Why ‘Pick the Best Technology’ is Poor and Impractical Advice

Talent Pool, Collaborative Scalability, Documentation, and Community Are Perhaps Better Metrics

Nicklas Millard
Principal Software Engineer
nicklas@mjukvare.com

January 31, 2025

I recently had a discussion with a developer who stubbornly claimed that you should always pick the best technology for the job.

I couldn’t disagree anymore.

To me, this is not only poor advice, it is also a deeply arrogant and misguided position to take.

The discussion was in the context of a founder asking if any developer would be willing to come work for him, in a paid software developer position on the project. The founder stated that he wants the application built in JavaScript and Postgresql. This sparked the question: *“why are the technologies decided in advance?”*

Now, say these two technologies were not the *best* choice for the application. Let’s imagine that something like C#, Java, or even C++ would have been better picks. Better picks purely from a performance standpoint – which was the stance of the developer with whom I had a discussion.

First off, is performance really something we want to concern ourselves with from the very beginning of a startup? Unless the startup is about providing optimization services or applications from the get-go, this aspect would be the least of my concerns. Secondly, there’s a high likelihood that a complete rewrite is required within a short timespan. This goes for both startups and internal corporate projects. The initial design is often the worst design. Even if you spend time thinking through the problem up front.

Secondly, the founder has skin in the game. The developer just shows up and gets paid if things turn out great or not, provided that the company has liquidity to pay a salary, obviously. As a developer, taking the stance that "we need to use the best technology" is arrogant and impractical. Say the argument is that picking another technology would perform 100x faster. Now, this stance not only dismisses the company's wishes; it also shows lacking understanding of what the company deems important. Who says ultimate performance is a key metric the company cares about? Also, how are the technologies practically weighed against each other?

Company needs change.

Picking a technology-stack is not a static choice that cannot be undone. As companies evolve, so do their tech-stack. One may start off with the LAMP stack if that is how the company could get a good working prototype in front of users fast. As the domain knowledge deepens and new technological challenges present themselves, the company may want to rethink the choice – or not.

Airbnb was initially built with Ruby, but also decided to go with JavaScript later on. Honey had skilled JavaScript developers and built their app in JavaScript, but transitioned to Swift for iOS development. Netflix had their own datacenters, running the databases themselves, and later migrated to AWS, as owning their datacenters was not a competitive advantage. These are just a few of many stories about evolving tech-stacks.

To me, this goes to show that picking the *best* technology up-front, and sticking with it, is not desirable, viable, nor practical to do. Things change, what was once considered the best may eventually fade and not even be an option anymore.

What really matters when picking a suitable tech-stack.

Talent pool and depth are important aspects of picking a technology. Choosing a technology with wide-enough talent pools insulates the project or company from sudden brain-drain, where the skilled developers go for new opportunities elsewhere. Also, the depth of knowledge in tech-stacks varies between regions in the world, and even within countries. If a tech-stack has a good talent pool with a skilled community of practitioners close by, then that might be the one you want to go for. On the other hand, consider a really strong technology for your use-case, but talent is sparse or has low adoption among senior developers, then the technology is quickly becoming liability. Code itself is already a liability, you don't need to make matters worse.

Familiarity is a great indicator of whether to pick a technology or not. Professionally, I would much rather work in a technology that I am deeply familiar with, where I know where to find help when stuck, where the good online documentation lies, and that I have gained mastery in. I have no issue with learning a new technology, but in the context of building production-grade software, I prioritize technology familiarity and company demands any day.

Collaborative scalability is another important consideration. If you build microservices or have a company with different development teams, allowing everything to be built in the most optimal tech-stack is likely to be detrimental to productivity, and confusing at best. Try to build five services in wildly different tech-stacks, maintain them and hire people for them over time. It's a mess. Simply pulling a developer from one team into another is not as straightforward either. The developer may be new to the technology, language, framework, and so on. And, being new to something typically means not being very productive or knowing what a smart decision looks like.

Well-documented technologies make life easy. I enjoy that. What I don't enjoy the frustration of having a deadline slowly creeps up as I'm still trying to figure out how things work, fix seemingly random bugs, or read out-dated and sparse documentation. If you're choosing between a nicely documented technology or a better technology, but lacking in documentation, I'd say go with the well-documented one, considering that both technologies can actually do the work.

A strong community is the lifeblood of any successful technology. Have you ever thought of learning a new technology, but found that it had been mostly abandoned and only a few die-hard fans were left hanging out at the forums? This is not a technology you would want to bet your investment on.

Do developers have an obligation to use the best technology?

I think not.

That is not to say, we should just accept ideas or direction that we consider poor or "bad".

We have a professional obligation to present our informed views, share our lived experiences, and point out challenges. If we are asked to perform a task that is likely to have negative implications, or is even downright illegal, you have a professional responsibility to make your manager aware. If still pushed to go forward, there are times you need to stand your ground – especially if asked to do anything knowingly illegal. But, if it's nothing more than "I'd strongly recommend C# over JavaScript", and your recommendation is not met, then it's not the time to throw an entitled millennial tantrum.

In summary, use what is suitable.

Choosing between technologies or tech-stacks is not a one-dimensional decision. Claiming performance is the overarching reason to do anything is oversimplification at best. Rather, think in terms of how familiar you are with a technology, if you can hire the right people, if the technology is well-documented, and people can efficiently work together on it.

The author

Nicklas Millard is a Principal Software Engineer and previously FinTech engineer at BankingCircle and Engineering Manager at Deloitte Consulting, where he was Engineering Community Lead of 30 developers and architects. He has worked on highly sensitive projects for the Ministry of Defence, the National Police, and other Danish government agencies. Nicklas holds Cosmic Top Secret NATO clearance. In addition to project work, he has created development courses for IDA and private companies. As a tech-writer, his work has been read over two million times.