automated tests, reducing outages further and allowing the whole company to deploy twice as often. Let your boss dream of being the one to tell the CEO that they've managed to cut outages in half and double deployments. And then mention that, based on your research, you believe you can deliver this future world using Terraform.

There's no guarantee that your boss will say yes, but your odds are quite a bit higher with this approach. And your odds get even better if you work incrementally.

Work Incrementally

One of the most important lessons I've learned in my career is that most large software projects fail. Whereas roughly 3 out of 4 small IT projects (less than $1 million) are completed successfully, only 1 out of 10 large projects (greater than $10 million) are completed on time and on budget, and more than one-third of large projects are never completed at all.[1]

This is why I always get worried when I see a team try to not only adopt IaC but to do so all at once, across a huge amount of infrastructure, across every team, and often as part of an even bigger initiative. I can't help but shake my head when I see the CEO or CTO of a large company give marching orders that everything must be migrated to the cloud, the old datacenters must be shut down, and that everyone will "do DevOps" (whatever that means), all within six months. I'm not exaggerating when I say that I've seen this pattern several dozen times, and without exception, every single one of these initiatives has failed. Inevitably, two to three years later, every one of these companies is still working on the migration, the old datacenter is still running, and no one can tell whether they are really doing DevOps.

If you want to successfully adopt IaC, or if you want to succeed at any other type of migration project, the only sane way to do it is incrementally. The key to *incrementalism* is not just splitting up the work into a series of small steps but splitting up the work in such a way that every step brings its own value—even if the later steps never happen.

To understand why this is so important, consider the opposite, *false incrementalism*.[2] Suppose that you do a huge migration project, broken up into several small steps, but the project doesn't offer any real value until the very final step is completed. For example, the first step is to rewrite the frontend, but you don't launch it, because it relies on a new backend. Then, you rewrite the backend, but you don't launch that either, because it doesn't work until data is migrated to a new data store. And then, finally, the last step is to do the data migration. Only after this last step do you finally launch everything and begin realizing any value from doing all this work. Waiting until the very end of a project to get any value is a big risk. If that project is canceled or put on hold or significantly changed partway through, you might get zero value out of it, despite a lot of investment.

In fact, this is exactly what happens with many large migration projects. The project is big to begin with, and like most software projects, it takes much longer than expected. During that time, market conditions change, or the original stakeholders lose patience (e.g., the CEO was OK with spending three months to clean up tech debt, but after 12 months, it's time to begin shipping new products), and the project ends up getting canceled before completion. With false incrementalism, this gives you the worst possible outcome: you've paid a huge cost and received absolutely nothing in return.

Therefore, incrementalism is essential. You want each part of the project to deliver some value so that even if the project doesn't finish, no matter what step you got to, it was still worth doing. The best way to accomplish this is to focus on solving one, small, concrete problem at a time. For example, instead of trying to do a "big bang" migration to the cloud, try to identify one, small, specific app or team that is struggling, and work to migrate just them. Or instead of trying to do a "big bang" move to "DevOps," try to identify a single, small, concrete problem (e.g., outages during deployment) and put in place a solution for that specific problem (e.g., automate the most problematic deployment with Terraform).

If you can get a quick win by fixing one real, concrete problem right away, and making one team successful, you'll begin to build momentum. That