Mark Schwartz

Napoleon, DevOps, and Delivering Business Value

DevOps Enterprise Summit Las Vegas 2018

25 NW 23rd Pl, Suite 6314
Portland, OR 97210

The contents of this eBook are a transcript of the complete presentation given by Mark Schwartz, "Napoleon, DevOps, and Delivering Business Value" at the DevOps Enterprise Summit Las Vegas 2018. To view the original presentation, please visit https://www.youtube.com/watch?v=KR7Y8IUgyyA.

eBook published 2019 by IT Revolution Press.

For further information on this or any other books and materials produced by IT Revolution Press please visit our website at ITRevolution.com

NAPOLEON, DEVOPS, AND DELIVERING BUSINESS VALUE

Mark Schwartz

DevOps Enterprise Summit Las Vegas 2018

I am an enterprise strategist at AWS. What that means is I'm part of a small team of ex-CIOs and CTOs of large enterprises. What we do is we go talk to our customers who are large enterprises and who are in the midst of a transformation moving to the cloud, adopting DevOps, whatever it is, and are hitting impediments of some sort. Usually, it's non-technical impediments, it's cultural change and organizational change and financial issues and bureaucracy that they have to overcome. In fact, what I just said describes pretty much every large enterprise customer that we talk to. We try to give them some strategies based on our experience and based on what we see other customers do.

I'm going to talk a little bit about what I think is the key impediment that everybody faces. So, before I joined AWS, I was the CIO at US Citizenship and Immigration Services, which is part of the Department of Homeland Security. Before that I was CIO of a company in San Francisco, and CEO of a software company in Oakland. I think what happened to me is one day I was reading a newspaper article about all of the IT problems in Homeland Security. Being the problem solver that I am, my reaction immediately was, I can fix that, let me at it. We can do this.

Somehow all the stars aligned. I wound up at USCIS and about a year ago, it suddenly hit me, we were done. Everything was fixed. The government was running like clockwork, and it was probably time to move on to the next opportunity, so I wound up at AWS. Along the way, I wrote a couple of books, published a couple of books, that you might have seen, and I am just finishing off the third book. We're going to have some preliminary copies available tomorrow. It's called *War and Peace in IT*. As the title would suggest, it's a sequel to Tolstoy's novel, and I'm trying to bring out his critical messages on DevOps, which I thought maybe required a sequel to really articulate.

But let me tell you a little bit of a story from *War and Peace* to set the stage for what I want to say about some of the impediments that large enterprises face when they're adopting DevOps. In this novel, what's happening is Napoleon is invading Russia. That's kind of the through line through the whole thing. Napoleon, great military leader, and his army are coming into Russia, and the Russians keep withdrawing, and his army pushes deeper, and the Russians withdraw a little bit more. Then finally the big climactic scene comes when they're finally going to have their big battle: the Battle of Borodino. What happens at Borodino goes something like this.

You have Napoleon the day before the battle, and he's sort of surveying the ground with his key reports, direct reports. He's giving some instructions: you guys go over there and do this, and you guys go over there and do that. And he's setting the scene for the battle. Of course, each of the generals, or whatever their rank is, they think they know better than Napoleon, so they're mostly not listening to him. They decide to do some other things that don't really fit together very well. But then when the battle starts, Napoleon withdraws to a fort that he had taken the day before about a mile away in Shevardino. He's at this fort a mile away while the battle is going on and he's issuing commands.

The way that this works is messengers ride up from the battle, and they tell Napoleon what's going on. He makes a decision, gives an order, and the messengers go back to the front with the order. Napoleon can't actually see what's going on in the battle because he's a mile away and it's a misty day—there's lots of mist and there's smoke all over the battlefield once the battle starts, and there are actually gullies in between. So he can't see anything. He has to wait for the messengers to come. So the messengers come with a message and by the time they get to Napoleon, the news that they're bringing is completely outdated. In one very memorable moment, they ride up and say, “We’ve taken the bridge at Borodino. Do you order the army to cross it?” Napoleon says, “Yes, have them cross it and form into ranks on the other side.”

What he doesn’t know is that not only have the Russians taken it back but they’ve burned it, there is no bridge anymore. This actually happened the moment the messenger had left to go tell Napoleon. So Tolstoy’s point—and this is partly a novel and partly all about history—but Tolstoy’s point is that Napoleon has nothing to do whatsoever with what’s actually happening on the battlefield. He thinks he’s issuing orders but none of it actually means anything. So what is Tolstoy trying to tell us about DevOps? I think the point here is you have a long decision cycle—Napoleon’s decision cycle is really long because he has to wait for the messengers to come and go back—but the action that’s actually happening has a very short cycle time. It’s moving very quickly. There’s a sort of an impedance mismatch here.

He has these long decision-making cycles, but things are moving really quickly. There’s no way you can actually control something that’s moving quickly if you’ve got this big decision cycle. So if you look at DevOps in an enterprise, you have a business situation that’s changing very rapidly, that’s full of complexity, uncertainty, rapid change. You have a process, a technical process, DevOps, that works religiously to shorten lead times. And it’s embedded in a business process that is nothing of the sort, that has a really long lead time. DevOps more or less takes us from the point where we write code and commit it to the point where it gets deployed and gets into users hands, and, as the last presentation here mentioned, monitoring and telemetry kick in and prove that it’s actually working. But it’s from code commit essentially to use in production.

But the process for making business changes is this big thing. Generally, it starts with something like realizing you have business needs or mission needs; figuring out what they are; putting a group of them together into a big project or a big program; building a business case for that big program; passing it through an investment management process or a governance process, a steering committee, whatever; finding the resources who are going to work on it . . . Eventually, you get to the point where a code is written and deployed, and then you also have a bunch of downstream stuff to realize the value, or harvest the value, out of these changes that you’ve made.

So what you have is from concept to deployment or concept to cash, in some cases, but idea to impact you have this really long lead time. This is Napoleon on his hill. He’s making decisions about what needs to happen, what’s going to be valuable, and by the time it gets implemented, everything's changed already. No matter how much you speed up the part that DevOps covers, it doesn't make such a big difference for the big project work, at least, when you're in sort of a maintenance mode and you have rapidly churning requirements perhaps. But in a lot of cases in a big enterprise, you're dealing with big capital investments that need to go through this big drawn-out process to get to the point where you can actually start writing some code.

So that's the area that really interests me. What can we do about all of this? Because, really, to take full advantage of what DevOps can give us, we need to speed up the entire lead time, we need to somehow shrink it. Because that's what's going to let us maintain our nimbleness, our agility, our innovation where we can try experiments and make decisions based on the results. So my interest is a big up front fuzzy front end and what can we do about it. So first idea in my train of thought here is, the way we set up this big process is designed for a world that's predictable or at least reasonably predictable. We happen to be in an environment as Napoleon was, an environment that's filled with uncertainty, complexity, rapid change.

The mental model for how you make decisions, how you make investment decisions in particular, in an environment where things are reasonably predictable is very different from the mental model you use when things are changing very rapidly and things are uncertain. So the old scheme was to put together a business case. And your business case involves projecting revenues and costs based on this new IT system you're building. Everyone understands that you can't exactly project your revenues and costs, but the assumption is that if you put a stake in the ground and you put a number there, you're going to be somewhere close to reality. There's going to be some variance.

What if any accurate projection was so uncertain that a confidence interval of 60% would require plus or minus 200% from your estimate? In other words, this is going to make us $10 million of revenue maybe, plus or minus $40 million or something like that. The more extreme your uncertainty is, the less meaningful a business case is. The environment that we're in, I think, is one of enough uncertainty that trying to project revenues and costs and make decisions based on them is not the right mental model. When you're in that mental model, when you're making your decisions based on a business case, it's really important that when you actually execute the project, you execute it exactly as you planned it. Because the business case is based on your exact plans.

So if the business case says, this is going to make us $10 million a year and it's going to cost $1 million to build, well, when you execute that project you'd better spend $1 million and you better get back $10 million. Execution according to plan is highly valued. But in a world of uncertainty, where you know that things are not going to go the way that you plan, it's a bad idea to pretend that they are. That's just willfully misleading yourself and making bad decisions.

So we thought of this oversight process, governance process, that is based on assumptions about the old world that don't quite tie to the new world. If you think about why we have all of those processes, I would submit that there are really two reasons, two end goals.

One of them is to reduce risk of the actual execution, and the second is to make the best decisions about where to put your resources. You have an opportunity cost if you're going to invest in this project rather than that project. Well, you better have a sense of how much return you're going to get from both, and the return from this one has to be bigger. So we set up these processes of building business cases and reviewing them and trying to align with strategic objectives and all of those things in order to mitigate risk and allocate capital to the appropriate investments.

Well, in Napoleon's world, making those decisions in advance about exactly how you're going to conduct the battle probably is not that effective. It's not effective at mitigating risk, it's not effective about using the right deployments of your resources to conduct the battle, it just doesn't work very well. In an environment of uncertainty where unexpected events are going to happen and you know that they are, this sounds at first like a highly risky environment, and the traditional way of thinking about risk would call it that. Things are going crazy, there's disruption in all the industries, and geopolitical things, and all of this stuff. The change is just going to be constant over the next few years. That sounds pretty risky. It sounds pretty risky if you're still thinking in terms of business cases and trying to execute exactly on that business case.

But the fact that unexpected things are going to happen is not necessarily a hazard, it could be an opportunity as well. Something unexpected happens: either it destroys you or it makes you better. The difference between those, whether the unexpected is going to lead to something good or lead to something bad, is your ability to seize an opportunity out of that change. In other words, it depends on your agility, your speed, your flexibility, and your inventiveness when you see what actually happens. The best way to manage risk in the kind of environment that we're in is to make sure that anything unexpected that happens can be turned it into an opportunity. We have the agility and the inventiveness to be able to turn it into a positive thing.

Risk, in a sense, in today's environment is the same as not being agile. The agility I'm talking about is that entire value stream, where you have to notice the opportunity, figure out what you're going to do, justify it, apply the capital resources and everything else, and do all of that quickly and agilely. So how can we rework this long value stream in a way that promotes speed, agility, flexibility, and innovation or inventiveness? Well, I can think of three basic models. I'm going to walk through the three models. This is not to say that these are the only three models, these are just the ones that come to mind when I think about it. Actually, I'll tell you all three, at least a name for each so that we can correspond them.

The first one is what some people would call the product model, second is what I'm going to call the budget model, and the third is the objective model. So product, budget, objective. The product model says we're going to reduce the time that this takes by having fast decision-making cycles because we've de-centralize them into a product team. This is very much the way that AWS works, for example. So I'm very familiar with the model now. In AWS, we offer 125 or so different services, and there's a team responsible for each of those services, and the team maintains its own roadmap. It is influenced by central input, let's say, but the team has control of its roadmap and works with customers to figure out what they need, and then the team decides what it's going to do about those customer needs. Ninety to ninety-five percent of our product roadmap is drawn directly from requests from customers, so it's a process that's optimized for that. Let's listen to the customers, make decisions. The decisions don't have to go to a central authority, the governance is all done locally.

So that's a product type model. Now, the central concerns do have some influence on it. For example, there's an imperative to all of the service teams to reduce their costs and pass the savings onto customers. Each service team can interpret that however they want or need to. They figure out how to reduce costs. But there's this high level objective that gets passed on to the teams. That's what a product model might look like.

A budget model, to me this is interesting. Let me back up a tiny bit on this one. I think we've had this myth in the IT community for a while that we have two kinds of costs in our IT budget. We have keeping the lights on costs and we have innovation costs. All of us, I think, say innovation is where we want to spend our money, maintenance is not where we want to spend our money, keeping the lights on. I disagree, actually. I think a lot of what goes by the name of keeping the lights on, a lot of that maintenance stuff, is actually innovation work. It's actually what's advancing the business. It's actually what's changing the IT systems to keep them consistent with what the business needs.

You don't have to maintain software, by the way. You have to maintain a car to make it keep functioning the way it always did. Software keeps doing exactly what it did when you bought it. You don't have to put money into it to make it keep doing that. The problem is, you never want software to keep doing what it did when you bought it. You want to keep changing it as your business changes. So the maintenance spend or the keeping the lights on spend, to a large extent, is remaking the decision that this is the right software for you to use and making changes to it as you need to.

I say that as a backdrop to my budget model, because usually the maintenance side of things is done out of sort of a budget: we're going to put this much money in keeping the lights on and then we'll figure out how to spend it, as opposed to innovation money which is usually a large capital expense that has to go through a governance process.

So why not treat everything that we do as just ongoing maintenance of our IT assets? Sometimes it involves building or buying new systems, sometimes it involves making changes to an existing system. More often today, it's refactoring existing systems. Maybe a strangler pattern, breaking off pieces and doing something with them. But it doesn't really matter which of those it is, and it doesn't matter to the business users how you're getting them the capabilities. It could be from an existing application or a new one. In fact, if you're building on an existing application, it's a very effective way to spend your money because you've already got something there, every dollar is probably getting you more.

But either way, you've got this big legacy estate of IT, and you're constantly changing it to keep up with what the business needs, and you do that through a budget, and the budget is passed down through an organizational structure. That means that the amount of money that makes its way to whoever's spending it, they have control over how it's spent in their budget. So you don't need these long cycles of asking: Is it okay if we add this feature and this feature? You don't have to go to a steering committee for that. So the budget model potentially is a way to speed up your time to decisions around your DevOps initiative.

It's the third model that I think is really interesting, and something that we tried out at USCIS that I think was really effective. Since then, I'm hearing about more organizations using this. I call it the objective model, and I'll give you an example of how it's used because it's easier to see that way. At USCIS, one of the systems that we were in charge of was E-Verify, which is the application that employers can use to make sure that their employees are eligible to work in the United States. We anticipated that at some point soon for political reasons, that's going to have to scale up like crazy. At some point, congress is going to say all employers need to use this. Right now, it's just a very small set of employers that do. So we were going to have to scale the thing like crazy. And we realized that it wouldn't scale, not because of technical reasons, because it's in the cloud now, we have elasticity in the Cloud, it can scale.

The problem was the human part of the system. So E-Verify at the time we started this could in an automated way handle 98.6% of the cases. That's great, except the other 1.4% of the cases a human being had to look at. And if we suddenly got a lot more cases, we wouldn't have enough people to do that. So an objective we had was to increase this 98.6% to a higher number. A second objective was a human being who was adjudicating these cases could do about seventy cases a day. We said, we need that number to go way up. We need to develop software or something that's going to let them adjudicate more cases every day. A third was when companies signed up to use E-Verify, we had a shopping cart abandonment problem. About 40% of them actually made it through the whole process, the rest did not. They got stuck somewhere. So we said we wanted that number to go closer to 100%.

All together, we realized that we had five big business objectives in this project: 98.6%, make it go up; seventy cases a day, make it go up; 40% finishing the registration process, make it go up. Then two others that I won't talk about. So instead of the way you would normally do this, in the traditional way of making investment decisions, you would translate those objectives into a bunch of requirements. Then you would put all those requirements into a bundle, build a business case for it, and then try to execute those requirements.

This is a little strange if you think about it, because adding those requirements actually adds risk to the project. What I mean is, if you're just going to take those objectives and try to execute them, that's one thing. But if you've now said, those objectives turn into these requirements and that's going to have the impact we want, you've now added the risk that your requirements are not the right ones. What you really want is to *accomplish the objectives*. The old project way of thinking said what you want is to *finish off* these requirements, but you've added a risk in between. You've also tied the hands of the innovative people who you want to be thinking of good solutions by adding these requirements.

So what we chose to do in this case was instead of creating requirements, we just took those objectives and passed them to the teams directly. So we would create a team that was a cross-functional team, this might sound familiar from DevOps. It included developers, operations and infrastructure people, testers, security people, and it also included business operations people. So, truly cross-functional team. It had business skills, it had technical skills. Then we said, seventy cases a day, make it go up. And that's the only instruction we gave. We didn't say, here's a bunch of requirements, execute the requirements. That team then owned making seventy go up, and they could do it in whatever way they could figure out that would make that number increase.

In fact, we told them specifically if you can do this without writing any software, just changing business processes, go right ahead and do that. If you want to write some software...whatever it is that's going to make that number go up, all we care about is the number. And since you've got the cloud and you've got DevOps tooling and the process all set up, you should be able to start producing functionality tomorrow. So in two weeks, tell us what that number is now. Seventy now. In two weeks, tell us what number you've gotten it up to, and then every two weeks after that we're going to review it and see what number you've gotten it up to. Every time, we're going to remake the decision to keep investing in this based on what we see.

So after a month, we could see that the number of cases per day was going up, and we said to the oversight body that was responsible for the investment: Look, the number is going up. We think we should continue investing in this objective. What do you think? And they said, yeah, sounds good.

With the shopping cart abandonment rate problem, remember it was 40%, it went up, it went up, and then it started plateauing. In our bi-weekly discussions with the team that owned that objective, we asked what kinds of things they were doing to make the number go up. What they told us made a lot of sense, but it just didn't seem to be budging. So when we went back to the investment committee, we said, here's what we're seeing. It did this. We're trying everything, it's not helping. We suggest that we stop investing in this objective and put the money on something else.

They thought that was pretty strange, because no project ever returns money, but that was . . . the point of this process was that because we were going to constantly be reviewing it, the central body still had control over it in every relevant sense—in fact, more control than if we had started with a big set of requirements—and could constantly remake the decision about whether the spending was going well and divert resources. So ultimately, this project was continued. I think they just officially ended the project. It was planned to be about a four-year project, after about two and a half years the team said: We're done. We've accomplished these five objectives to the extent that they can be accomplished, so let's just return the rest of the money.

This, I think, is the perfect model for the age of Napoleon, where you have all this complexity and uncertainty by decentralizing the decision-making to the teams but agreeing on an objective that could be used for control centrally. Everybody was aligned, all of the controls were in place around the investment, the team could be innovative and could be testing hypotheses, and could continue to invest in the things that were working. Their accountability was to accomplish the objectives, which is what the business case was built on in the first place.

So the three models that I've given are all ways to stop this big, long investment decision cycle, make it much shorter by decentralizing control over the decisions so they don't have to go to Napoleon who's a mile away and come back again. But, the central authority still has control over it in every meaningful sense and can manage the risk of it.

So the three models again were: Product team that owns its own roadmap but has influence from the center. The budget model, where funds are allocated through a hierarchy until they reach a team which now has the funds or really in most cases the number of teams, the production capacity, they're going to decide how to use it, and then the central authority can reallocate funds and make changes based on how well it's being spent. Then the third model, where what's cascaded from the center is just an objective, and the team then has the freedom within that objective to do whatever it is able to find that will accomplish the objective. Essentially, the requirements can vary, which is really what we want in an agile world.

So these aren't the only possible models, but I wanted to throw them out there as ways to think about the problem. But the problem is we still have to shrink this long cycle if we're going to take full advantage of the DevOps short cycle time and the flexibility that we get from it.

Going back to Napoleon, I think the important thing to realize is shrinking cycle times is not about doing things faster. His army is on the field; it's going to take as long as it takes. It's not about the speed; it's about the quality of the decisions. Napoleon can't make good decisions from where he is because the cycle time for his decisions is out of sync with the cycle time that things are actually happening.

I think when you put DevOps into an enterprise, it's the same concern. It's not just about how quickly you can get stuff to market, although that's important, it's about how can your central organization still have good control and still make good decisions while the action is moving really fast both in the business context and in the DevOps process. So fast cycle time here, got to make this cycle time fast so that you can make really good decisions, so that you can lead your army to success as Napoleon didn't really do in Russia in the end. That I think is what Tolstoy has to teach us about DevOps. So thank you all.

ABOUT THE AUTHOR

Mark Schwartz is an iconoclastic CIO and a playful crafter of ideas, an inveterate purveyor of lucubratory prose. He has been an IT leader in organizations small and large, public, private, and nonprofit.

As an Enterprise Strategist for Amazon Web Services, he uses his CIO experience to advise the world's largest companies on the obvious: time to move to the cloud, guys. As the CIO of US Citizenship and Immigration Services, he provoked the federal government into adopting Agile and DevOps practices. He is pretty sure that when he was the CIO of Intrax Cultural Exchange, he was the first person ever to use business intelligence and supply chain analytics to place au pairs with the right host families.

Mark speaks frequently on innovation, change leadership, bureaucratic implications of DevOps, and using Agile practices in low-trust environments. With a BS in computer science from Yale, a master's in philosophy from Yale, and an MBA from Wharton, Mark is either an expert on business value and IT or just confused and much poorer.

Mark is the author of *The Art of Business Value* and *A Seat at the Table*, and the winner of a *Computerworld* Premier 100 award, an Amazon Elite 100 award, a Federal Computer Week Fed 100 award, and a *CIO Magazine* CIO 100 award. He lives in Boston, Massachusetts.