Exploring Maturity in the DevOps Entry Point

The second part of a series that highlights most effective first steps for shedding inefficient and ineffective information technology practices.

Al DuPree

Thu, 03/28/2019 - 10:00

The first part of this series covered how information technology practices were heavily influenced by the interests and the practices of early market participants. How early market players like IBM and Arthur Andersen helped to shape how we approach application development and technology deployment even today. And how those early influences did not always result in the most effective methods.
This part will discuss the most effective first steps for shedding those inefficient and ineffective practices. How adopting DevOps represents an exceptional first step and how that early phase of adoption should focus on process maturity before deploying performance metrics.

**The Challenge DevOps Addresses**

John Willis and Damon Edwards in 2010 coined the term CAMS for Culture, Automation, Measurement and Sharing, as a standard definition or framework for DevOps. And as "standard definitions" or "frameworks" go, this one encompasses all the necessary elements.

Thinking carefully about the four words above, you’ll notice a lack of technology names, little mention of specific skillsets, no concern with "fill-in-the-blank-as-a-service." As described thus far, DevOps has far more to do with challenging how we organize the IT function and what kinds of incentives we provide than it has to do with what technologies we select.

DevOps starts by challenging the current system of incentives — a fundamental pillar of any purpose-driven enterprise’s culture. For software developers to date, the incentive framework within which they operate can be summed up as being recognized and rewarded for "providing a specific (often changing) set of capabilities within an agreed upon time frame with a set amount of resources." That’s a long way of saying "on time, within budget."
If you deliver within the confines of those parameters (or game the system, or both) then you get the prize. So then what are the incentives for the folks charged with evaluating the quality of the software produced? Number of software bugs found or number of test cases produced. Find/produce a lot of either or both? Once again, rewards will be forthcoming. And what about the incentives provided to the folks responsible for the care and feeding of infrastructure components? In large part, they can be summed up in a single word: stability. Infrastructure (like well-behaved children) is supposed to be obedient, doing what’s instructed without question or failure. And that is in line with its predecessors and the originators of the model that we labor under: development, QA and infrastructure are "separate and apart" from each other, each with its own separate criteria of what constitutes success operating largely in their own realms. And it’s possible to extend the conversation to virtually any of the generic functional areas found in IT organizations, from the help desk to the security apparatus.

So when separate incentive bundles create conflict between the IT tribes? There’s no real clear path to resolution. When, for example, the developers are anxious to declare something “complete” while QA is intent upon continuing to pick through the code in search of errors, while the infrastructure team chooses to declare an outage due to the arrival of critical firmware, there are few ways to avoid open warfare. Or when the folks concerned with security are included on the back end of a development or deployment effort and decide to halt deployment because of security concerns. Or when the folks concerned with user support and assistance decide to continuously refer problems and errors back into the developer/QA queues because they haven’t been “in the loop” during development nor have they any training with the newly developed product.

DevOps isn’t then so much a “methodology” for getting out of the quagmire outlined above as it is guidance for instigating and directing a change in the organizational culture that creates the quagmire in the first place. DevOps accomplishes this first by establishing within the culture an imperative to share (another word from above). Share the results, share the responsibility, share the credit and, above all, share both the success and the failure.

The DevOps Foundation
The underlying software development methodology that DevOps depends on is Agile. And it’s Agile that lays the groundwork to address the issues for the current set of governing incentives.

First, by forcing reconsideration of the scope and occurrence of products provided, Agile dictates smaller and more focused product releases provided more frequently. The most obvious initial consequence of this is that success is built in smaller increments as opposed to a single grand event. So-called "waterfall" approaches, made popular by Andersen Consulting (now Accenture) and others of what was then the "big eight" accounting firms were rooted in the approaches of their audit practices of the time. Arguably there is a narrow range of development and deployment efforts where such approaches are appropriate and beneficial (financial systems come to mind).

By and large, however, Agile allows organizations to more effectively manage the change that new systems represent. Equally as critical, frequent smaller increments allow development organizations to fail (and recover) fast, as opposed to the catastrophic failures associated with large releases or rollouts of the past.

Such an approach also improves the overall quality of releases by narrowing the focus of what’s to be tested and corrected. The more frequent release cycle provides the ability to determine how and where to apply resources as your project progresses. And finally, such an approach allows for a degree of experimentation that didn’t exist before. In short, Agile eliminates the old notion and version of “on time” and “on budget” by radically altering the time frames, redefining the scope of what’s considered a deliverable, and providing flexibility in resource commitment.

The DevOps approach extends Agile’s reorganization of developer time and resources to the entire IT organization using the software product and its series of releases as the focal point. DevOps dictates that the entirety of the organization must be represented, active and involved in this process because part of the cultural change that DevOps is intending to address is the idea of shared (there’s that word again) responsibility for outcomes.
Gene Kim, an early contributor to the definition of DevOps, captured this concept in what he calls "systems thinking and flow," part of his "three ways" defined in a series of blog posts in 2012. The idea is simple, “This is about considering the performance of the whole system, not just an individual component or department.” The simple premise of the DevOps culture change is: it’s all of our product and therefore all of our responsibility. Failure isn’t fatal. But finger-pointing is.

So, unlike a standalone methodology, which provides a series of prescribed procedures and rules in a designated order to achieve a desired outcome, DevOps provides a set of principles that resemble more a strategy than they do specific tactics. A roadmap for challenging old assumptions and changing shared values. For most organizations, once the premise of DevOps has been accepted, the most pressing question is: Where to begin?

To make culture changes, the place to start is not with technology. It’s with how you organize and develop your people.

**Ground Zero: Do you have a PMO?**

In fact, the most impactful and visible first action may be to rename the function. "DevOps Office" perhaps? If you don’t have a PMO, then there’s no renaming necessary; start your DevOps office from day one.

PMOs as they stand are, at the very least, a common repository for the plans of all active and soon-to-be active projects. As such, they represent the best possible entry point for a DevOps adoption. In theory, you have a complete scan of the organization’s activities and therefore the capacity to select what activities to begin with and which teams should represent the vanguard of this effort. So this is the place that you can begin to reshape how the organization approaches, talks about and organizes its work. And most important of all, it’s where you can place specific personnel with specialized business skills who are critical to the overall success of the effort (more about those people in the next part of the series). This is more than a simple renaming exercise. This is the first step in what could be a singularly significant cultural shift. And we’ll talk, in detail, about the tactics necessary for that shift also in the next part of the series.
Before we get to that, it’s important to drive home the idea that such a vast and significant cultural change, if it’s to be effective, must also be subjected to measurement. If we don’t measure, how can we possibly know to what extent we are successful?

The Role of the Capability Maturity Model

Back in 1988, the Software Engineering Institute at Carnegie-Mellon University in Pittsburgh published the first version of its Capabilities Maturity Model, or CMM, and subsequently updated the model to CMM Integration (CMMI). What it was intended to capture was how completely an organization has adopted a set of practices, which can, when employed, consistently produce desired outcomes. There are a number of capability maturity models available, and there have been several academic studies that evaluate both the efficacy of maturity models as well as their specific features.

One of the first orders of business then, in the newly named DevOps Office, is to research and adopt a capability maturity model. And ultimately this is a singularly critical step, so don’t ignore it. Why? That answer is twofold: First and foremost, the evaluation of how well or poorly a process is being taken up by an organization should be separate from the activity of evaluating any singular result of that process. But secondly, the critical personnel that you will be installing in the DevOps office will be able to significantly leverage such a tool. More on that in part three.

The final installment of this series will cover the human roles that are critical to the success of a DevOps adoption effort. We’re going to add to that a set of numbers that you can use to measure how successful you are in creating and deploying product. And we’re going to give you some practical ways to communicate to management as well as the rest of the organization exactly what you’re doing and why.
This was adapted from an original blog post by Al DuPree, who is an information systems professional with international experience in strategic planning, software application development and delivery, data center operations and information technology related service delivery. His past experiences include chief of innovation and solutions division at the National Institute of Standards and Technology and CIO at the Congressional Budget Office. You can hear more from DuPree in an upcoming episode of our new podcast, The Agile Advocate.