Cutter
IT Journal

The Journal of
Information Technology Management

Vol. 24, No. 8
August 2011

"Some people get stuck on the word 'devops,' thinking that it's just about development and operations working together. Systems thinking advises us to *optimize the whole*; therefore devops must apply to the whole organization, not only the part between development and operations."

**— Patrick Debois,
Guest Editor**

Devops: A Software Revolution in the Making?

At Arm's Length

Automating the delivery process reduces the time development and operations have to spend together. This means that when dev and ops *must* interact, they can focus on the issues that really matter.

Arm in Arm

Development and operations often play cats and dogs during deployment. To stop the fighting, we must bring them closer together so they will start collaborating again.

CUTTER CONSORTIUM

Opening Statement

by Patrick Debois, Guest Editor

Despite all the great methodologies we have in IT, delivering a project to production still feels like going to war. Developers are nervous because they have the pressure of delivering new functionality to the customer as fast as possible. On the other side, operations resists making that change a reality because it knows change is a major cause of outages. So the usual finger-pointing begins when problems arise: "It's a development problem"; "Oh no, it's an operations problem." This tragic scene gets repeated over and over again in many companies, much to the frustration of management and the business, which is not able to predict releases and deliver business value as expected and required.

The problem is further amplified by two industry trends: agile development and large-scale/cloud infrastructure. Agile development by its nature embraces change and fosters small but frequent deployments to production. This higher frequency puts additional pressure on operations, which becomes even more of a bottleneck in the delivery process. What is remarkable is that even though agile actively seeks collaboration from all its stakeholders, most agile projects did not extend themselves toward the operations people. Continuous integration and the testing department served as the traditional buffer between development and operations, and in that buffer, pressure has also been building up.

The second driver, large-scale/cloud infrastructure, forced operations to change the way they managed their infrastructure. Traditional manual labor didn't cut it at the scale of thousands of machines. It's good to want to do small increments and frequent changes, but you need to have the tools to support that. Virtualization enabled ops to spin up new environments very quickly, and cloud made away with the resource problem. The real differentiator came from two concepts: configuration management and infrastructure as code. The first enables you to describe the desired state of an infrastructure through a domain-specific language, allowing you to both create and manage your infrastructure using the same tools. Infrastructure as code is a result of the increase of SaaS and APIs used to handle infrastructure. Embracing both concepts allows operations to automate a lot of its work — not only the initial provisioning of a new machine, but the whole lifecycle. This gave birth to the concept of agile infrastructure.

As a response to these two drivers, devops was born. A number of people got together and started a grass-roots movement that set out to remove the traditional boundaries between development and operations. Some consider this picking up where "traditional agile" left off. After all, software doesn't bring value unless it is deployed in production; otherwise, it's just inventory. To tackle the problem, devops encourages cross-silo collaboration *constantly*, not only when things fail.

Like exercise, the more you practice deployment to production, the better you will get at it.

Operations becomes a valued member of the traditional agile process with an equal voice. Far too often, SLAs negotiated with a customer work counter to the changes requested by the same customer. Now both functional *and* nonfunctional requirements are evaluated for their business priority. This moves the usual discussion on priorities back before development starts. As both development and operations serve the same customer, the needs of both are discussed at the same time.

Once priorities have been determined and work can start, developers *pair together* with operations people to get the job done. This pairing allows for better knowledge diffusion across the two traditionally separate teams. Issues such as stability, monitoring, and backup can be addressed immediately instead of being afterthoughts, and operations gets a better understanding of how the application works before it actually deploys it in production.

When it's hard, do it more often. Like exercise, the more you practice deployment to production, the better you will get at it. Thanks to automation, operations can introduce small changes much more quickly to the

infrastructure. Practicing deployment in development, testing, quality assurance, and production environments, and managing it from the same set of defined models in configuration management, results in more repeatable, predictable results.

As both development and operations are now responsible for the whole, lines are beginning to blur. Developers are allowed to push changes to production (after going through an extensive set of tests), and just like their operations counterparts, they wear pagers in case of emergencies. They are now aware of the pain that issues cause, resulting in feedback on how they program things.

People active in continuous integration were among the first to reach out to operations. In the first article in this issue, Jez Humble and Joanne Molesky elaborate on the concept of an "automated deployment pipeline," which extends the boundaries of software development to the deployment phase. All changes to a production environment follow the same flow, regardless of whether the changes are to code, infrastructure, or data. The deployment pipeline acts as sonar for issues: the further you get in the pipeline, the more robust the results get. And you want to track the issues as fast as possible, since the cost of fixing them goes up the further you go in the process.

Feedback is available to all people: those in operations learn what issues they might expect in production, while developers learn about the production environments. Nor is feedback only of a technical nature — management and the business can learn from production trial runs what customers really want and how they react. This moves product development even further into customer development and allows you to test-drive your ideas in almost real time.

In our second article, Eric Shamow shows that, like any other change, devops is clearly a discovery path — people and processes don't change overnight. A common pattern in devops adoption seems to be that organizational changes go hand in hand with the introduction of new tools. A tool by itself will not change anything, but the way you use a tool can make a difference in behavior, resulting in a cultural change. Therefore, driving change requires management support; otherwise, the impact of the change will be limited to some local optimization of the global process.

Shamow's story also shows that a lot of organizations were already taking devops-like approaches, so for them it is nothing new. The great thing with the term "devops" is that people can use it to label their stories. And thanks to this common label, we can search for those stories and learn from others about what works or not.

All the focus in devops on automation to reduce cycle time and enhance repeatability makes you wonder if we will all become obsolete and be replaced by tools. There is no denying that automation can be an effective cost cutter, as our next article, a case study by Larry Fitzpatrick and Michael Dillon, clearly demonstrates. But we need to recognize the real opportunity: all the time won by automation gains us more time to design and improve our work. This is where the competitive advantage comes in. People sometimes lightly say that their most valuable asset is people, but it's oh so true. They will give your business the edge it needs if you let them.

Agility in coding, agility in systems — it takes time and effort to nurture these ideas, but the results can be astonishing. The feedback cycle makes the whole difference in quality and results. With all this close collaboration, lines begin to blur. Will we all be developers? Does everybody deploy? In a multidisciplinary approach, you have either the option of *generalizing specialists* (the approach taken in the organization Fitzpatrick and Dillon discuss) or *specializing generalists*. Both can work, but if you are generalizing specialists, you get the extra depth of their specialization as well. This is sometimes referred to as having "T-shaped people" in the collaboration: people with in-depth knowledge (vertical) who are also able to understand the impact in a broad spectrum (horizontal). In the traditional organization, this would map to a matrix organization structure, similar to the one the authors describe. It's important to rotate people through different

projects and roles, and it's equally important for the specialists to touch base again with their respective specialist groups to catch up with new global changes.

Many of the initial devops stories originated in Web 2.0 companies, startups, or large-scale Internet applications. In contrast with more traditional enterprises, these organizations have a much lighter and more flexible management structure. It's worth nothing that several of the large Web 2.0 companies such as Amazon and Flickr didn't use the term "agile" but were working with a similar mindset. Small groups of both developers and operations people delivered new releases and understood that they were working toward the same goals for the same customers.

Given these origins, many people at first dismiss devops, thinking it will never work in their traditionally managed enterprise. They see it conflicting with existing frameworks like CMMI, agile, and Scrum. Consider, however, that most of the stories in this journal come from traditional enterprises, which should serve to debunk that myth. Traditional enterprises need to care even more about collaboration, as the number of people involved makes one-on-one communication more difficult. The term "devops" is only a stub for more global company collaboration.

Our next author, Bill Phifer, clearly shows there is no reason to throw existing good practices away. They might need to be adapted in execution, but nothing in them inherently conflicts with devops. Focusing particularly on CMMI for Development and ITIL V3, Phifer discusses how these two best practice models can be integrated to enable improved collaboration between development and operations. But as useful as this mapping of CMMI and ITIL functions to each other is, Phifer argues that it is not enough. Rather, the "result of effective integration between CMMI and ITIL as applied to the devops challenge is a framework that ensures IT is aligned with the needs of the business."

Indeed, we sometimes forget that the reason we do IT is for the business, and if the business doesn't earn money, we can be out of a job before we know it. As Phifer emphasizes, this relation to the business is an important aspect in devops: some people get stuck on the word "devops," thinking that it's just about development and operations working together. While this feedback loop is important, it must be seen as part of the complete system. Systems thinking advises us to *optimize the whole*; therefore, devops must apply to the whole organization, not only the part between development and operations.

In our final article, Dominica DeGrandis focuses on just this kind of systems thinking. She talks about the leadership changes that will be required to enable the "devops revolution" and outlines how statistical process control can be applied to devops to increase predictability and thus customer satisfaction. DeGrandis concludes by describing her experience at Corbis (the same company where David Anderson developed his ideas on applying Kanban to software development), recounting how devops-friendly practices like continuous integration and smaller, more frequent releases greatly improved the IT organization's ability to deliver business value.

The term "devops" is only a stub for more global company collaboration.

Only by providing positive results to the business and management can IT reverse its bad reputation and become a reliable partner again. In order to do that, we need to break through blockers in our thought process, and devops invites us to challenge traditional organizational barriers. The days of top-down control are over — devops is a grass-roots movement similar to other horizontal revolutions, such as Facebook. The role of management is changing: no longer just directive, it is taking a more supportive role, unleashing the power of the people on the floor to achieve awesome results.

Patrick Debois is a Senior Consultant with Cutter Consortium's Agile Product & Project Management *practice. To understand current IT organizations, he has made a habit of changing both his consultancy role and the domain in which he works: sometimes as a developer, manager, sys admin, tester, or even as the customer. During 15 years of consultancy, there is one thing that annoys him badly; it is the great divide between all these groups. But times are changing now: being a player on the market requires you to get the "battles" under control between these silos. Mr. Debois first presented concepts on agile infrastructure at* Agile 2008 *in Toronto, and in 2009 he organized the first* devopsdays *conference. Since then he has been promoting the notion of "devops" to exchange ideas between these different organizational groups and show how they can help each other achieve better results in business. He's a very active member of the agile and devops communities, sharing a lot information on devops through his blog http://jedi.be/blog and twitter (@patrickdebois). Mr. Debois can be reached at pdebois@cutter.com.*

Why Enterprises Must Adopt Devops to Enable Continuous Delivery

by Jez Humble and Joanne Molesky

> Roughly speaking those who work in connection with the [Automated Computing Engine] will be divided into its masters and its servants. Its masters will plan out instruction tables for it, thinking up deeper and deeper ways of using it. Its servants will feed it with cards as it calls for them. They will put right any parts that go wrong. They will assemble data that it requires. In fact the servants will take the place of limbs. As time goes on the calculator itself will take over the functions both of masters and of servants.
>
> — Alan Turing, "Lecture on the Automatic Computing Engine"[1]

Turing was — as usual — remarkably prescient in this quote, which predicts the dev/ops division. In this article, we will show that this divide acts to retard the delivery of high-quality, valuable software. We argue that the most effective way to provide value with IT systems — and to integrate IT with the business — is through the creation of cross-functional product teams that manage services throughout their lifecycle, along with automation of the process of building, testing, and deploying IT systems. We will discuss the implications of this strategy in terms of how IT organizations should be managed and show that this model provides benefits for IT governance not just in terms of improved service delivery, but also through better risk management.

THE PREDICAMENT OF IT OPERATIONS

IT organizations are facing a serious challenge. On the one hand, they must enable businesses to respond ever faster to changing market conditions, serving customers who use an increasing variety of devices. On the other hand, they are saddled with evermore complex systems that need to be integrated and maintained with a high degree of reliability and availability. The division between projects and operations has become a serious constraint both on the ability of businesses to get new functionality to market faster and, ironically, on the ability of IT to maintain stable, highly available, high-quality systems and services.

The way organizations traditionally deliver technology has been codified in the established discipline of project management. However, while the projects that create new systems are often successful, these projects usually end with the first major release of the system — the point at which it gets exposed to its users. At this stage the project team disbands, the system is thrown over the wall to operations, and making further changes involves either creating a new project or work by a "business as usual" team. (The flow of value in this model is shown in Figure 1.) This creates several problems:

- Many developers have never had to run the systems they have built, and thus they don't understand the tradeoffs involved in creating systems that are reliable, scalable, high performance, and high quality. Operations teams sometimes overcompensate for potential performance or availability problems by buying expensive kits that are ultimately never used.
- Operations teams are measured according to the stability of the systems they manage. Their rational response is to restrict deployment to production as much as possible so they don't have to suffer the instability that releases of poor-quality software inevitably generate. Thus, a vicious cycle is created, and an unhealthy resentment between project teams and operations teams is perpetuated.
- Because there are several disincentives for teams to release systems from early on in their lifecycle, solutions usually don't get near a production environment until close to release time. Operation teams' tight control over the build and release of physical servers slows the ability to test the functionality and deployment of solutions. This means that their full production readiness is usually not assessed until it is too late to change architectural decisions that affect stability and performance.
- The business receives little real feedback on whether what is being built is valuable until the first release, which is usually many months after project approval. Several studies over the years have shown that the biggest source of waste in software development is features that are developed but are never or rarely

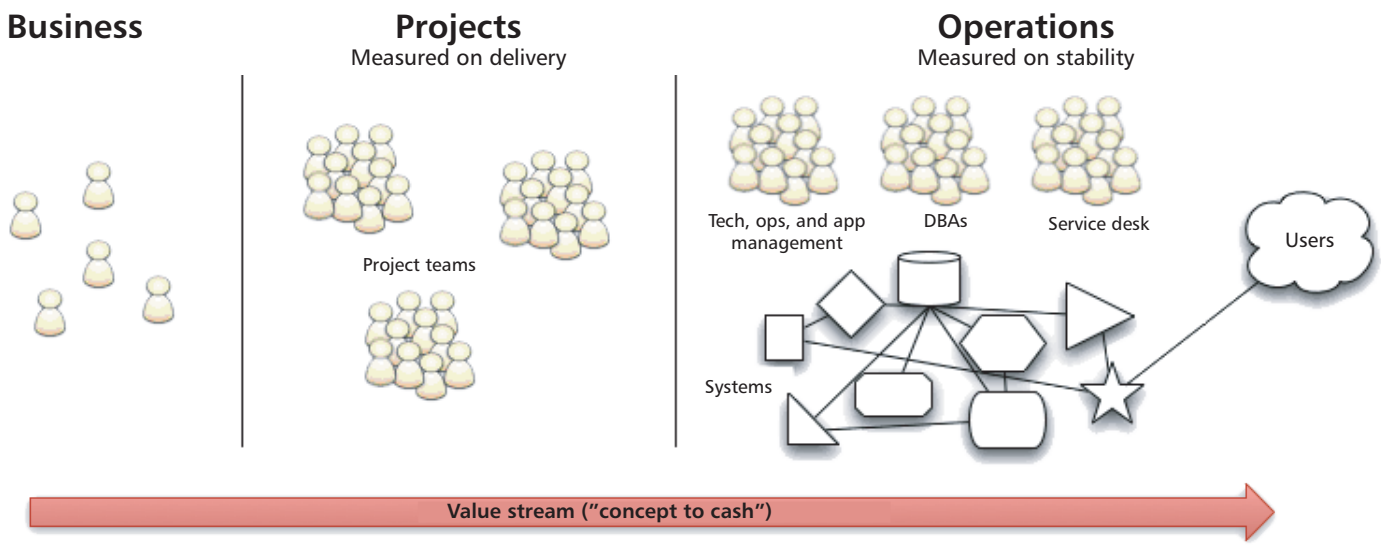

Figure 1 — The flow of value in project-based organizations.

used — a problem that is exacerbated by long release cycles.

- The funding model for projects versus operating expenses creates challenges in measuring the cost of any given system over its lifecycle. Thus, it is nearly impossible to measure the value provided to the business on a per-service basis.
- Due to the complexity of current systems, it is difficult to determine what should be decommissioned when a new system is up and running. The tendency is to let the old system run, creating additional costs and complexity that in turn drive up IT operating costs.

The upshot is that IT operations must maintain an ever-increasing variety of heterogeneous systems, while new projects add more. In most organizations, IT operations consumes by far the majority of the IT budget. If you can drive operating costs down by preventing and removing bloat within systems created by projects, you'd have more resources to focus on problem solving and continuous improvement of IT services.

DEVOPS: INTEGRATING PROJECT TEAMS AND IT OPERATIONS

Devops is about aligning the incentives of everybody involved in delivering software, with a particular emphasis on developers, testers, and operations personnel. A fundamental assumption of devops is that achieving both frequent, reliable deployments and a stable production environment is not a zero-sum game. Devops is an approach to fixing the first three problems listed above through culture, automation, measurement, and sharing.[2] We will address each of these aspects in turn.

Culture

In terms of culture, one important step is for operations to be involved in the design and transition (development and deployment) of systems. This principle is in fact stated in the ITIL V3 literature.[3] Representatives from IT operations should attend applicable inceptions, retrospectives, planning meetings, and showcases of project teams. Meanwhile, developers should rotate through operations teams, and representatives from project teams should have regular meetings with the IT operations people. When an incident occurs in production, a developer should be on call to assist in discovering the root cause of the incident and to help resolve it if necessary.

Automation

Automation of build, deployment, and testing is key to achieving low lead times and thus rapid feedback. Teams should implement a deployment pipeline[4] to achieve this. A deployment pipeline, as shown in Figure 2, is a single path to production for *all* changes to a given system, whether to code, infrastructure and environments, database schemas and reference data, or configuration. The deployment pipeline models your process for building, testing, and deploying your systems and is thus a manifestation of the part of your value stream from check-in to release. Using the deployment pipeline, each change to any system is validated to see if it is fit for release, passing through a comprehensive series of automated tests. If it is successful, it becomes available for push-button deployment (with approvals, if required) to testing, staging, and production environments.

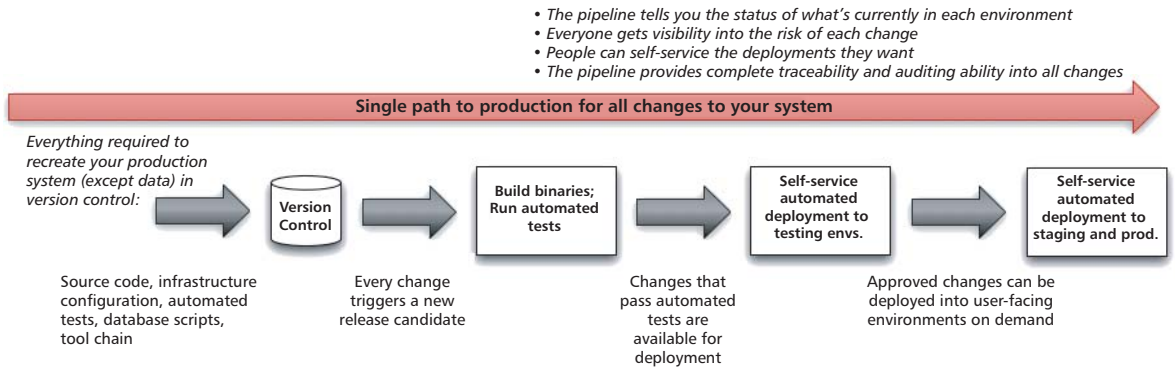

Figure 2 — The deployment pipeline.

Deployments should include the automated provisioning of all environments, which is where tools such as virtualization, IaaS/PaaS, and data center automation tools such as Puppet, Chef, and BladeLogic come in handy. With automated provisioning and management, all configuration and steps required to recreate the correct environment for the current service are stored and maintained in a central location. This also makes disaster recovery much simpler, provided you regularly back up the source information and your data.

One way to help developers focus on creating more stable systems might be to measure the effect of releases on the stability of the affected systems.

Measurement

Measurement includes monitoring high-level business metrics such as revenue or end-to-end transactions per unit time. At a lower level, it requires careful choice of key performance indicators, since people change their behavior according to how they are measured. For example, measuring developers according to test coverage can easily lead to many automated tests with no assertions. One way to help developers focus on creating more stable systems might be to measure the effect of releases on the stability of the affected systems. Make sure key metrics are presented on big, visible displays to everybody involved in delivering software so they can see how well they are doing.

In terms of process, a critical characteristic of your delivery process is lead time. Mary and Tom Poppendieck ask, "How long would it take your organization to deploy a change that involves just one single line of code? Do you do this on a repeatable, reliable basis?"[5]

Set a goal for this number, and work to identify and remove the bottlenecks in your delivery process. Often the biggest obstacle to delivering faster is the lengthy time required to provision and deploy to production-like environments for automated acceptance testing, showcases, and exploratory and usability testing, so this is a good place to start.

Sharing

Sharing operates at several levels. A simple but effective form of sharing is for development and operations teams to celebrate successful releases together. It also means sharing knowledge, such as making sure the relevant operations team knows what new functionality is coming their way as soon as possible, not on the day of the release. Sharing development tools and techniques to manage environments and infrastructure is also a key part of devops.

DIY Deployments

If you implemented all of the practices described above, testers and operations personnel would be able to self-service deployments of the required version of the system to their environments on demand, and developers would get rapid feedback on the production readiness of the systems they were creating. You would have the ability to perform deployments more frequently and have fewer incidents in production. By implementing continuous delivery, in which systems are production-ready and deployable throughout their lifecycle, you would also get rid of the "crunches" that characterize most projects as they move toward release day.

However, while the practices outlined above can help fix the new systems you have entering production, they don't help you fix the string and duct tape that is holding your existing production systems together. Let's turn our attention to that problem now.

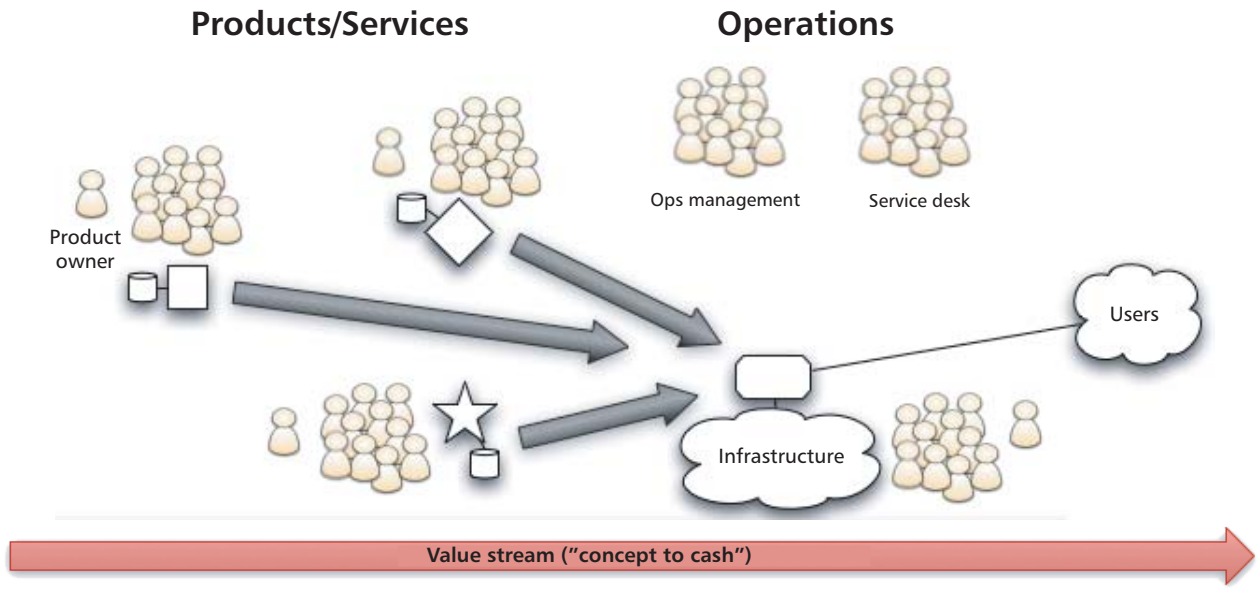

Figure 3 — The flow of value in a product development model.

DID RUBE GOLDBERG DRAW THE ARCHITECTURE DIAGRAM FOR YOUR PRODUCTION SYSTEMS?

We propose an old-fashioned method for simplifying your production systems, and it rests on an approach to managing the lifecycle of your services. Treat each strategic service like a product, managed end to end by a small team that has firsthand access to all of the information required to run and change the service (see Figure 3). Use the discipline of product management, rather than project management, to evolve your services. Product teams are completely cross-functional, including all personnel required to build and run the service. Each team should be able to calculate the cost of building and running the service and the value it delivers to the organization (preferably directly in terms of revenue).

There are many ways people can be organized to form product teams, but the key is to improve collaboration and share responsibilities for the overall quality of service delivered to the customers. As the teams come together and share, you will develop a knowledge base that allows you to make better decisions on what can be retired and when.

What is the role of a centrally run operations group in this model? The ITIL V3 framework divides the work of the operations group into four functions:[6]

1. The service desk
2. Technical management
3. IT operations management
4. Application management

Although all four functions have some relationship with application development teams, the two most heavily involved in interfacing with development teams are application management and IT infrastructure. The application management function is responsible for managing applications throughout their lifecycle. Technical management is also involved in the design, testing, release, and improvement of IT services, in addition to supporting the ongoing operation of the IT infrastructure.

In a product development approach, the central application management function goes away, subsumed into product teams. Nonroutine, application-specific requests to the service desk also go to the product teams. The technical management function remains but becomes focused on providing IaaS to product teams. The teams responsible for this work should also work as product teams.

To be clear, in this model there is *more* demand for the skills, experience, and mindset of operations people who are willing to work to improve systems, but less for those who create "works of art" — manually configured production systems that are impossible to reproduce or change without their personal knowledge and presence.

Once your organization has reached some level of maturity in terms of the basics of devops as described

in the previous section, you can start rearchitecting to reduce waste and unnecessary complexity. Select a service that is already in production but is still under active development and of strategic value to the business. Create a cross-functional product team to manage this service and create a new path to production, implemented using a deployment pipeline, for this service. When you are able to deploy to production using the deployment pipeline exclusively, you can remove the unused, redundant, and legacy infrastructure from your system.

Finally, we are not proposing that the entire service portfolio be managed this way. This methodology is suitable for building strategic systems where the cost allocation model is not artificial. For utility systems that are necessary for the organization but do not differentiate you in the market, COTS software is usually the correct solution. Some of the principles and practices we present here can be applied to these services to improve delivery, but a dependence on the product owner to complete change will restrict how much you can do and how fast you can go. Certainly, once changes are delivered by a COTS supplier, you can test and deploy the changes much faster if you have the ability to provision suitable test and production environments on demand using automation.

Many organizations attempt to create small teams, but they often make the mistake of splitting them functionally based on technology and not on product or service.

DEVOPS AT AMAZON: IF IT'S A TRENDY BUZZWORD, THEY'VE BEEN DOING IT FOR YEARS

In 2001 Amazon made a decision to take its "big ball of mud"[7] architecture and make it service-oriented. This involved not only changing the architecture of the company's entire system, but also its team organization. In a 2006 interview, Werner Vogels, CTO of Amazon, gives a classic statement not only of the essence of devops, but also of how to create product teams and create a tight feedback loop between users and the business:

> Another lesson we've learned is that it's not only the technology side that was improved by using services. The development and operational process has greatly benefited from it as well. The services model has been a key enabler in creating teams that can innovate quickly with a strong customer focus. Each service has a team associated with it, and that team is completely responsible for the service — from scoping out the functionality, to architecting it, to building it, and operating it.
>
> There is another lesson here: Giving developers operational responsibilities has greatly enhanced the quality of the services, both from a customer and a technology point of view. The traditional model is that you take your software to the wall that separates development and operations, and throw it over and then forget about it. Not at Amazon. You build it, you run it. This brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. This customer feedback loop is essential for improving the quality of the service.[8]

Many organizations attempt to create small teams, but they often make the mistake of splitting them functionally based on technology and not on product or service. Amazon, in designing its organizational structure, was careful to follow Conway's Law: "Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."[9]

IMPROVED RISK MANAGEMENT WITH CONTINUOUS DELIVERY

A common reason given for not trying devops and continuous delivery in IT shops is that this approach does not comply with industry standards and regulations. Two controls that are often cited are segregation of duties and change management.

Regulations and standards require organizations to prove they know what is happening and why, protect information and services, and perform accurate reporting. Most IT organizations are subject to some kind of regulation and implement controls in order to ensure they are in compliance. Controls are also essential to reducing the risk of having bad things happen that may affect the confidentiality, integrity, and availability of information.

Segregation of duties is a concept derived from the world of accounting to help prevent fraud and reduce the possibility of error. This control is required by regulations and standards such as SOX and PCI DSS. The relevant COBIT control states:

> **PO4.11 Segregation of Duties**
>
> Implement a division of roles and responsibilities that reduces the possibility for a single individual to compromise a critical process. Make sure that personnel are performing only authorized duties relevant to their respective jobs and positions.[10]

The spirit of the IT control is that one person should not have the ability to make preventable errors or introduce nefarious changes. At a basic level, you have checks and balances to make sure this doesn't happen. This control can be implemented many different ways; an extreme interpretation of this control by some organizations is that development, operations, and support functions need to be functionally and physically separated and cannot talk to each other or view each other's systems.

Those of us who have experienced working in these organizations know that this level of control only serves to increase cycle time, delay delivery of valuable functionality and bug fixes, reduce collaboration, and increase frustration levels for everyone. Furthermore, this type of separation actually increases the risk of error and fraud due to the lack of collaboration and understanding between teams. All IT teams should be able to talk and collaborate with each other on how to best reach the common goal of successful, stable deployments to production. If your IT teams don't talk and collaborate with each other throughout the service/product delivery lifecycle, bad things *will* happen.

A Better Way: The Automated Deployment Pipeline

Reducing the risk of error or fraud in the delivery process is better achieved through the use of an automated deployment pipeline as opposed to isolated and manual processes. It allows complete traceability from deployment back to source code and requirements. In a fully automated deployment pipeline, every command required to build, test, or deploy a piece of software is recorded, along with its output, when and on which machine it was run, and who authorized it. Automation also allows frequent, early, and comprehensive testing of changes to your systems — including validating conformance to regulations — as they move through the deployment pipeline.

This has three important effects:

1. Results are automatically documented and errors can be detected earlier, when they are cheaper to fix. The actual deployment to production with all associated changes has been tested before the real event, so everyone goes in with a high level of confidence that it will work. If you have to roll back for any reason, it is easier.
2. People downstream who need to approve or implement changes (e.g., change advisory board members, database administrators) can be automatically notified, at an appropriate frequency and level of detail, of what is coming their way. Thus, approvals can be performed electronically in a just-in-time fashion.
3. Automating all aspects of the pipeline, including provisioning and management of infrastructure, allows all environments to be locked down such that they can only be changed using automated processes approved by authorized personnel.

If your IT teams don't talk and collaborate with each other throughout the service/product delivery lifecycle, bad things *will* happen.

Thus, the stated goals of the change management process are achieved:[11]

- Responding to the customer's changing business requirements while maximizing value and reducing incidents, disruptions, and rework
- Responding to business and IT requests for change that will align the services with the business needs

As well as meeting the spirit of the control, this approach makes it possible to conform to the letter of the control. Segregation of duties is achieved by having your release management system run all commands within the deployment pipeline as a special user created for this purpose. Modern release management systems allow you to lock down who can perform any given action and will record who authorized what, and when they did so, for later auditing. Compensating controls (monitoring, alerts, and reviews) should also be applied to detect unauthorized changes.

IMPLEMENTING CONTINUOUS DELIVERY

Continuous delivery enables businesses to reduce cycle time so as to get faster feedback from users, reduce the risk and cost of deployments, get better visibility into the delivery process itself, and manage the risks of software delivery more effectively. At the highest level of maturity, continuous delivery means knowing that you can release your system on demand with virtually no technical risk. Deployments become non-events (because they are done on a regular basis), and all team members experience a steadier pace of work with less stress and overtime. IT waits for the business, instead of the other way around. Business risk is reduced because decisions are based on feedback from working software, not vaporware based on hypothesis. Thus, IT becomes integrated into the business.

Achieving these benefits within enterprises requires the discipline of devops: a culture of collaboration between

all team members; measurement of process, value, cost, and technical metrics; sharing of knowledge and tools; and regular retrospectives as an input to a process of continuous improvement.

From a risk and compliance perspective, continuous delivery is a more mature, efficient, and effective method for applying controls to meet regulatory requirements than the traditional combination of automated and manual activities, handoffs between teams, and last-minute heroics to get changes to work in production.

It is important not to underestimate the complexity of implementing continuous delivery. We have shown that objections to continuous delivery based on risk management concerns are the result of false reasoning and misinterpretation of IT frameworks and controls. Rather, the main barrier to implementation will be organizational. Success requires a culture that enables collaboration and understanding between the functional groups that deliver IT services.

The hardest part of implementing this change is to determine what will work best in your circumstances and where to begin. Start by mapping out the current deployment pipeline (path to production), engaging *everyone* who contributes to delivery to identify all items and activities required to make the service work. Measure the elapsed time and feedback cycles. Keep incrementalism and collaboration at the heart of everything you do — whether it's deployments or organizational change.

ACKNOWLEDGMENTS

The authors would like to thank Evan Bottcher, Tim Coote, Jim Fischer, Jim Highsmith, John Kordyback, and Ian Proctor for giving their feedback on an early draft of this article.

ENDNOTES

[1]Turing, Alan. *The Essential Turing*, edited by B. Jack Copeland. Oxford University Press, 2004.

[2]Willis, John. "What Devops Means to Me." Opscode, Inc., 16 July 2010 (www.opscode.com/blog/2010/07/16/what-devops-means-to-me).

[3]*ITIL Service Transition*. ITIL V3. Office of Government Commerce (OGC), 2007. (ITIL published a "Summary of Updates" in August 2011; see www.itil-officialsite.com.)

[4]Humble, Jez, and David Farley. "Continuous Delivery: Anatomy of the Deployment Pipeline." informIT, 7 September 2010 (www.informit.com/articles/article.aspx?p=1621865).

[5]Poppendieck, Mary, and Tom Poppendieck. *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley Professional, 2006.

[6]*ITIL Service Transition*. See 3.

[7]Foote, Brian, and Joseph Yoder. "Big Ball of Mud." In *Pattern Languages of Program Design 4*, edited by Neil Harrison, Brian Foote, and Hans Rohnert. Addison-Wesley, 2000.

[8]Vogels, Werner. "A Conversation with Werner Vogels." Interview by Jim Gray. *ACM Queue*, 30 June 2006.

[9]Conway, Melvin E. "How Do Committees Invent?" *Datamation*, Vol. 14, No. 5, April 1968, pp. 28-31.

[10]*COBIT 4.1*. IT Governance Institute (ITGI), 2007.

[11]*ITIL Service Transition*. See 3.

Jez Humble is a Principal at ThoughtWorks Studios and author of Continuous Delivery*, published in Martin Fowler's Signature Series (Addison-Wesley, 2010). He has worked with a variety of platforms and technologies, consulting for nonprofits, telecoms, and financial services and online retail companies. His focus is on helping organizations deliver valuable, high-quality software frequently and reliably through implementing effective engineering practices. Mr. Humble can be reached at jez@thoughtworks.com.*

Joanne Molesky is an IT governance professional and works as a Principal Consultant with ThoughtWorks Australia. Her work focuses on helping companies to reduce risk, optimize IT processes, and meet regulatory compliance through the application of agile methodologies and principles in delivering IT services. She is certified in ITIL and COBIT Foundations and holds CISA and CRISC designations through ISACA. Ms. Molesky can be reached at jmolesky@thoughtworks.com.

Devops at Advance Internet: How We Got in the Door

by Eric Shamow

Advance Internet is a medium-sized company with roughly 75 employees and around 1,000 servers devoted to a wide array of internally written and modified applications. When I arrived here in 2009, the company was in the midst of rolling out a new configuration management system (CMS) that had been in the works for roughly three years. Multiple deployment failures and poor information transfer had resulted in a complete breakdown of communication between operations and development. Developers were finding it increasingly difficult to get code onto test or production servers, or to get new servers provisioned. When they were deployed, it was often done incorrectly or without following release notes, and it often had to be redone several times. Operations had no information from development about when to expect releases, and when the releases did come, they would often be incomplete or inconsistent: they had configuration files in varied locations, contained poor or outdated release notes, and often depended on applications not installed from any repository. The entire operations team eventually left under various circumstances, and I was brought in as the new manager of systems operations to evaluate the situation, build a new team, and try to instill a culture of cooperation between development and operations.

I was not aware of "devops" as a movement at the time we started this project, although I became aware of it through my work and by reaching out to other professionals for suggestions and support. However, my experience discovering these problems and solving them from scratch led us straight down the devops path.

THE UNDERLYING ENVIRONMENT

There were several problems that became evident immediately. We began to tackle these issues individually, beginning with monitoring. At that time, the alerts we received were not relevant and timely, and metrics were unavailable, which led to a lot of guesswork and "from-the-gut" decision making. The build situation also needed to be fixed. New servers would be built by operations, be handed over to development for vetting, and then bounce back and forth in a three- or four-week cycle as corrections and modifications were made on both sides. By the end of this process, there was no clear record of who had done what, and the next build was inevitably condemned to failure. Even once we had gotten an idea of what went into a successful build, there was no automated means of repeating the process.

Monitoring and alerting turned out to be the easy part of the equation, but the one that would have the greatest impact on the tenor of the changes to come. Rather than attempting to fix the existing monitoring framework, we built a brand-new one using Nagios and PNP for graphing. We then developed an application severity matrix — based around the organization's list of known applications and their relative importance — in order to form the basis for a per-application escalation chain. With these established, we then met group by group with each team and the relevant stakeholders for the application or server in question and painstakingly listed which services needed to be monitored, how critical those services were, and when alerts should escalate.

The key piece of this was involving not just people responsible for monitoring the system, but the people who "owned" the system on the business side. Because they were participating in the discussion, they could help determine when a software engineer should be paged or when an alert could be ignored until morning, when they themselves should be paged, and how long to wait before senior management was notified. Once these meetings were held, we were able to move over 80% of the servers in our original alerting system to the new one. The remaining outliers enabled us to convert "unknown unknowns" into "known unknowns" — we now had a map of those servers with no clear owner and little documentation, so we knew which servers required the most focus. We also established a recurring quarterly schedule with the same stakeholders to review their application settings and determine whether we needed to make adjustments in thresholds and escalations.

Once this process was complete, access to our new Nagios install was given to everyone inside the technical organization, from the lowest-level support representative to the VP overseeing our group. We met with anyone who was interested and had several training classes

that demonstrated how to navigate the interface and how to build graph correlation pages. The goal was to open the metrics to the organization. Operations may not be able to keep its eye on everything at all times, so with more people with different interests looking at our data, problems were more likely to become apparent and decisions could be made and fact-checked immediately.

In the meantime, the configuration piece was moving much more slowly. Advance had long grown past the point where server deployments were manual, but what was in place was not much better than that manual deploy. There were two build systems in place — CONF REPO and HOMR — and I will stop to discuss them here purely as cautionary tales. Prior groups had clearly attempted to improve server deployment before and failed. The question I faced was, why? And how could I avoid repeating those mistakes?

Operations was trying to solve a discipline problem with a technological solution.

PREVIOUS ATTEMPTS AT A TOOLSET, AND MOVING FORWARD

In small-scale environments, making bulk configuration changes is not challenging. Servers are configured individually, and when a change must be made to multiple systems, administrators typically use a tool such as clusterSSH (which sends input from the user's workstation to multiple remote systems) or a shell script, which repeats the same changes on all of these systems sequentially. As environments become larger, however, it becomes more difficult to maintain systems in this way, as any changes made in an ad hoc fashion must be carefully documented and applied across the board. Therefore, in large-scale environments, it is common to use a configuration management system, which tracks changes and applies them automatically to designated servers.

The original configuration environment evident at the time of my arrival at Advance was CONF REPO. CONF REPO was in fact a heavily modified version of CFEngine. At some point in time, an inadequately tested change in CFEngine apparently caused a widespread systems outage, and the ops team responded by modifying CFEngine's default behavior. A script called cache_configs now attempted to generate a simulation of CFEngine's cache from configuration files and automatically copied this to each host. The host would then check what was in the new cache against what was on the filesystem, and it would then stop and send an alert rather than replace the files if there were any discrepancies. Although most of this functionality could have been achieved by CFEngine out of the box, the team had elected to rewrite large parts of it in the new CONF REPO framework, with the result that as CONF REPO's codebase diverged from CFEngine's, the framework became unmaintainable. CONF REPO was dropped as an active concern but never fully retired, so many hosts remained on this system.

This was followed by an attempt at writing a config management system from scratch, called HOMR. HOMR was really a wrapper around Subversion and functioned as a pull-only tool. Changes would be checked into HOMR, and then "homr forceconfigs" would be run on the client. This would pull down all configs and replace existing ones without warning. The result was that HOMR wouldn't be run on a host for weeks or months while individual admins tinkered with settings; then another admin on an entirely unrelated task (changing default SSH or NSCD settings, for instance) would run "homr forceconfigs" across the board and wipe out a bunch of servers' configs. This often resulted in production outages.

My evaluation of this situation was that operations was trying to solve a discipline problem with a technological solution. CFEngine, the originally chosen configuration management tool, is a well-respected and widely used system. Had it been used properly and left in place, Advance would have had a CMS years ago. Instead, due to poor discipline surrounding testing of changes, the system was abandoned and replaced with increasingly convoluted and difficult-to-maintain frameworks. This was not the way forward.

Although there was appeal in simply returning to CFEngine, there was enough mistrust of the tool around the organization that I instead elected to get a fresh start by bringing in Puppet as our new CMS. The idea was to create a greenfield "safe zone" of servers that were properly configured and managed and then to slowly expand that zone to include more and more of our environment.

Advance is a CentOS/RHEL shop, so I started by building a Cobbler server and cleaning up our existing Kickstart environment so that we had a reasonable install configuration. I then took the base configuration out of HOMR, translated it into Puppet, and set up a

bootstrap script in our Kickstart so that we could build Puppet machines with our base environment. We now had the ability to bring up new Puppet hosts. I wrapped the Puppet configuration directories in a Mercurial repository and shut off direct write access to this directory for the entire admin team — all changes would have to be made through Mercurial, ensuring accountability.

The next step was to start porting configs into configuration management (hereafter CM), building more hosts, and training my newly hired team of admins to use this system. More critically, I needed a way to get buy-in from suspicious developers and product owners (who had heard lots of stories of "new" configuration management systems before) so that I could lift up their existing systems, get their time and involvement in building CM classes reflecting their use, and ultimately rebuild their production environments to prove that we had in fact captured the correct configuration. This needed to work, it needed to be surrounded by accountability and visible metrics, and it needed to be done quickly.

BOOTSTRAPPING THE PROCESS

There were a few key pieces of learned wisdom I took into this next phase that helped ensure that it was a success. This was information I'd gathered during my years as a system administrator, and I knew that if I could leverage it, I'd get buy-in both from my new staff and from the rest of the organization. To wit:

- **Unplanned changes are the root of most failures.** Almost all organizations resist anything like change management until you can prove this to them, in black and white.
- **Engineers do a much better job of selling internal initiatives to each other than management does,** even if the manager in question is an engineer. When possible, train a trainer, then let him or her train the group.
- **Engineers like to play with fun stuff.** If you tie fun stuff to what they consider "boring stuff," they will put up with a certain amount of the "boring" to get at the "fun."
- **Organizations generally distrust operations if operations has a lousy track record.** The only way to fix this is to demonstrate repeatedly and over a long period of time that operations can execute to the business's priorities.

The "unplanned changes" principle was the first item to tackle and the easiest to prove within my team. Most engineers are at least somewhat aware of this phenomenon, even if they think that they themselves are never responsible for such problems. Clear metrics, graphs, and logs made this stunningly clear. Problems were almost always caused by changes that had been made outside the approved process, even if those problems didn't show up until a change made as part of the process ran up against the unplanned change weeks later. It took a lot longer to prove this to the business, because engineers will see the results in a log file and immediately draw the picture, whereas less technical stakeholders will require explanations and summary information. But over time, the concept is one that most people can understand intuitively; logs and audit trails made the case.

I was careful to pick a single person within the group to train on Puppet, a move that had a number of effects. It created something of a mystique around the tool ("Why does he get to play with it? I want in!"). It also enabled me to carefully shape that person's methodology to ensure that it was in line with my thinking, while giving him a lot of latitude to develop his own best practices and standards. This was a true collaboration, and it gave me a second person who could now answer CM questions as well as I could and who could train others.

Finally, we used the carrot approach and moved the most exciting, interesting new projects into CM. Anything new and shiny went there, regardless of its importance to the business. Want to build a new backup server for everybody's local Time Machine backups? Fine, but it's got to go in CM, and I want to see someone else deploy the final build to ensure it works. A few months of this, combined with scheduled training, meant that we quickly had a full group of people at least nominally versed in how Puppet worked, how to interact with Mercurial, and how to push changes out to servers.

At this point we were successful, but we had not yet achieved our goals. We could easily have failed here through the same kind of testing failure that brought down CFEngine within our group. The key step was scaling the toolset to keep pace with the number of people involved.

SCALING

Once we were prepared to move beyond two people making commits to our repository, we needed to worry about integration testing. At this point I identified two possible strategies for dealing with this:

1. Automated integration testing with a toolset such as Cucumber
2. Manual testing with a development environment

Although the first was clearly the optimal solution, the two weren't mutually exclusive. While the Cucumber solution would require a long period of bootstrapping and development, the development environment could be set up quickly and be used and tested immediately. We thus forked our efforts: I tasked my CM expert on the team with figuring out the development environment and how to get code moved around, and I began the longer-term work on the Cucumber solution.

> **Treat the code pool as a commons and make everyone responsible for growing it and keeping it clean.**

An issue that immediately raised itself was how to deal with potential code conflicts and the code repository. The team we had built was comfortable with scripting and low-level coding but had limited familiarity with code repositories. My initial plan had been to use typical developer tools such as branches and tags or perhaps a more sophisticated Mercurial-specific toolset revolving around Mercurial Queues. As we began to introduce more junior admins into the process, however, we found an enormous amount of confusion around merging branches, the appropriate time to merge, how to deal with conflicts, and other typical small-group issues when people encounter distributed change control for the first time. Rather than attempt to educate a large number of people on this quickly, we decided to build a simple semaphore-based system to prevent two people from working on the same CM module simultaneously.

Some will argue that this solution is suboptimal, but it allowed us to work around the branching issue during the early phases of deployment. There was never — and still isn't — any doubt that the "right" way to do this was with native repository technologies, but the key point for us was that not adhering to those enabled us to roll out a functioning environment with minimal conflicts and without the need to debug and train on complex merge issues. This kept our admins focused on the core elements of moving to real change control rather than tied up in the details of running it. The goal was to keep it simple initially to encourage more focus on getting systems into CM and writing good, clean CM entries.

DISCIPLINE AND CODING STANDARDS

Again drawing from lessons learned as a systems administrator and dealing with maintaining admin-written scripts, I began work on Puppet, operating from a series of core principles that informed decision making around the environment:

- **Coding standards are easy.** Many style guides are available online, but if your tool of choice doesn't have one, write your own. It does not have to be deeply complex, but your code should adhere to two principles: code must be clean and readable, and output must be similarly clean and machine parsable.
- **Development should be as generic as possible** so that nobody wastes time reinventing the wheel. If you are building a ticket server that runs behind Apache, take the time to build a generic Apache module that can be used by the entire team; if it uses MySQL, build that MySQL module. Treat the code pool as a commons and make everyone responsible for growing it and keeping it clean.
- **Rely on self-correction.** Once a critical mass is built and everyone is contributing to the same code base, peers won't tolerate bad code, and you as a manager won't have to execute as much oversight day-to-day. Oversight and periodic code review are still important, but it's much more important to have a strong group ethic that contributing "quick" but poorly written code to the repository is unacceptable. The more that peers find errors in each other's code and help to fix them, the more likely the system is to be stable as a whole and the better the code will likely integrate.
- **Enforce the use of a development environment.** If changes aren't put into place in a real, running CM environment and run in that environment to verify that they function properly, there is almost no point in doing devops — you are just abstracting the process of "doing it live." The key isn't just to put code through a code repository — it's to then do the same rigorous testing and deployment that developers must do before operations accepts its code as ready to go.

We did all of the above, to good effect. Within two to three months, everybody on the team had a few personal virtual machines hooked up to the development environment, which they could quickly redeploy to match real-world systems. Code was being contributed, shared, fixed, and extended by the entire group. When the CM system got upgraded and offered us new features we wanted to use, it took only very little training and some examples set by the specialist to enable the rest of the group to start working with the new tools.

CLOSING THE LOOP

None of the things we did to this point would have been helpful if we hadn't achieved the goal of improved communication and better deployment. So how did the company respond to these changes?

Most of the concerns from management and other groups were focused on time to deliver new builds and server updates, the accuracy of those builds, and communication with development about the processes it needed to follow to get material onto a server. Prior to our changeover to devops-style configuration management, the CMS project was a particular sore point. Rebuilding an individual server took an average of two weeks and two full days of admin time. After the change, a single engineer could do that same build in half a day — with active attention not fully on the build — with absolute precision. As a result, we were able to rebuild entire CMS environments in a day. Because developers worked with us to manage configuration files (only operations has commit rights, but we will often grant developers limited read access), it became very clear what needed to be in a package. And to facilitate faster development, for nonproduction or QA environments such as development and load, we used the Puppet ensure => latest directive for key packages. Consequently, developers only needed to check the newest version of their application package into a repository, and within 30 minutes Puppet would install it to their environments. In the meantime, our publicly available and transparent metrics enabled them to measure the results of these changes and ensure that changes were consistent across environments.

These developments had two immediate effects. First, the folks on the old CMS project started talking loudly about how well our project was going, and soon other product managers whose applications suffered from a situation similar to the former CMS project began to request that their servers move into our new CM. We soon had a queue longer than we could handle of projects fighting to be moved into the new environment. For a while, my ability to move servers in lagged behind my ability to train my staff in doing so.

Second, senior management immediately noticed the improvement in turnaround time and the disappearance of debugging time due to environment inconsistency. They immediately mandated that we roll our new CM out to as many environments as possible, with the goal of fully moving all of Advance into full CM by year's end. Management recognized the importance of this approach and was willing to make it a business rather than a technological priority.

I should not underemphasize the role of PR in making this project a success. A tremendous amount of time went into meeting with product owners, project managers, development teams, and other stakeholders to assure them about what was changing in their new environment. I was also willing to bend a lot of rules in less critical areas to get what we wanted in place; we made some short-term engineering compromises to assuage doubts in exchange for the ability to gain better control of the system in its entirety. For instance, we would often waive our rules barring code releases on Fridays, or we would handle less urgent requests with an out-of-band response normally reserved for emergencies to ensure that the developers would feel a sense of mutual cooperation. The gamble was that allowing a few expediencies would enable us to establish and maintain trust with the developers so we could accomplish our larger goals. That gamble has paid off.

Puppet is now fully entrenched in every major application stack, and it is at roughly 60% penetration within our environment. The hosts not yet in the system are generally development or low-use production hosts that for a variety of reasons haven't been ported; we continue slow progress on these. We have also begun to train engineers attached to the development teams in the system's use, so that they can build modules for us that we can vet and deploy for new environments. This gives the development teams increased insight into and control over deployments, while allowing us to maintain oversight and standards compliance. We have also established some parts of our Mercurial repositories where developers can directly commit files (e.g., often-changing configuration files) themselves, while still leveraging our system and change history.

NEXT STEPS

We're not finished yet — in fact we're just getting started. Exposure to the devops community has left me full of fresh ideas for where to go next, but I would say that our top three priorities are integration and unit testing, elastic deployments, and visibility and monitoring improvements.

For integration and unit testing, we have already selected Cucumber and have begun writing a test suite, with the goal that all modifications to our Puppet environment will have to pass these tests in order to be committed. This suite would not be in lieu of, but rather in addition to, our development environment. The ultimate goal here is to enable development teams or product owners to write agile-style "user

stories" in Cucumber, which developers can translate into RSpec and we can then run to ensure that CM classes do what they assert they can do. The integration testing piece is complete; unit tests will take considerably more time, but we will roll these out over the next six months.

Elastic deployment is ready to go but has not been implemented. Discussion of this is beyond the scope of this article, but the summary is that we use a combination of an automated deploy script, Cobbler, Puppet, and Nagios to determine thresholds for startup and shutdown and then automatically deploy hosts around some concrete business rules. The tools are in place, and we are now looking at viable candidate environments for test deployment.

Finally, visibility and monitoring, like security, are not a destination but a process. We continue to "sharpen the saw" by examining alternate monitoring tools, different ways to slice and improve metrics, automating simple statistical analysis such as moving trend lines, and developing dashboards. We are currently working on a number of dashboard components that would allow technically minded developers and even senior management to put together correlations and information from the servers on the fly.

Devops is now entrenched in the culture and fully supported by management. Now that we've gotten it in the door, it's up to us to use it to drive our environment further and even more tightly couple operations with development. We're on the same team … now we share the same information and tools.

FOR THOSE BEGINNING THE JOURNEY

If you are reading this and haven't yet begun the transition to a functioning devops environment, the length of the road may seem daunting. There are many organization-specific choices we made that may not be appropriate for you, but the key is not to get hung up on the details. I would offer the following thoughts to anyone looking to implement a solution similar to ours:

- Above all else, focus on the culture change and remember that you are working in a business environment. This change in culture will only occur if there is total commitment both from the driver of the change and the majority of the people who will have to participate in the new culture.
- If you must give — and be prepared to give, because at the end of the day, there will come an emergency more critical than your adherence to protocol — have a procedure in place for integrating those changes once the crisis is over. Don't allow management or other teams to see devops as something that prevents critical change; rather, emphasize the seriousness of bypassing it and then facilitate the changes the stakeholders need while working within your own system for addressing exceptions.
- Don't worry about the specific toolset — let the tools you choose select themselves based on the culture and skills of the group you have. Choosing the perfect configuration management system for a team whose strengths are not in the same area as the tool will not provide the best results, and the best-in-breed solution may not be the best for you. Keep everyone comfortable using the system and they will be more willing to use it; tools can always be changed once procedures are set.
- Your own transparency will open up other groups. The culture of information hiding in IT can be toxic and is done for various reasons, many of which are unintentional. Whether you are on the "dev" or the "ops" side of the divide, ensure that — regardless of the transparency of the other side — your information, your metrics, your documentation, and your procedures are kept open and available for view. Be open and willing to discuss changes in procedure with people that are affected by them. If you allow the devops change to be just another procedural change, it will fail. The change must fit the contours of your organization, which means give and take on both sides. If there has been a rift, you will need to give first and often to establish trust. The trust you earn at this point will buy you the leeway to do the previously unthinkable once the project really starts rolling.

At the time of writing, Eric Shamow was Manager of Systems Operations for Advance Internet. Mr. Shamow has led and worked in a wide variety of operations teams for the past 10 years. He has spent time in university environments and private industry, worked in a Wall Street startup, and at the number three comScore-rated news site. Mr. Shamow has managed a variety of Linux and Unix environments, as well as several large Windows-based deployments. He recently began work as a Professional Services Engineer for Puppet Labs.

Mr. Shamow has extensive experience in automating and documenting large-scale operational environments and has long worked to bring metrics-driven transparency to his teams. He has a paper due to be presented at USENIX's LISA '11 *on his work building an elastic virtual environment. Mr. Shamow can be reached at eric@opsrealist.com.*

The Business Case for Devops: A Five-Year Retrospective

by Lawrence Fitzpatrick and Michael Dillon

THE JOURNEY BEGINS

In 2005, a large organization with a fully outsourced IT operation and a sizable application development footprint was relying primarily on manpower for many aspects of development support, engineering, and operations. Having failed to leverage automation, its development, deployment, and operations latencies were measured in months, and manual change controls led to repeated break/fix cycles. Five years into the underperforming outsourcing contract, the organization lacked confidence that the outsourcer could execute on several key initiatives, so it decided to take back control and insource its application development. Today, the speed, capacity, and reliability of the operation is very different, due in large part to a "devops" group that grew within the new development organization, functioning at the juncture of applications and operations.

The devops movement (see Table 1) grew out of an understanding of the interdependence and importance of both the development and operations disciplines in meeting an organization's goal of rapidly producing software products and services. In this organization, a devops group emerged during the aforementioned insourcing of the company's application development[1] capabilities. Out of a desire to achieve consistency, efficiency, visibility, and predictability in development, and to support the transition to iterative/agile methods, the organization consolidated its development support resources from multiple development teams into a centralized Development Services Group (DSG). The DSG's mission was to standardize practice, automate development, and drive staffing efficiencies for software configuration management (SCM)/build, quality assurance (testing), and systems analysis activities.

From its inception five years ago, the DSG has led a transformation that delivers value far beyond initial expectations. Serving a development community of over 500, the group operates a centralized build farm that handles 25,000 software builds a year and an automated deployment stack that executes 1,500 software deployments a year. In addition to improving reliability, velocity, and transparency, this foundation has also been used to automate test frameworks, provisioning, monitoring, and security assurance. Most significantly, the group has captured cost savings and avoidance for the entire IT organization.

Bumps in the Road

This journey was particularly challenging — and perhaps different from other devops stories — due to the existence of an outsourcer that was highly resistant to change and frequently cloaked itself in the terms of the contract. This single-vendor contract covered substantially all of the IT function from operations to maintenance and development. Fortunately, certain contractual terms (activated by outsourcer failures) allowed the organization to begin to take control of development. Minor grumbling proved little hindrance when the new development organization took back control of development tool support (SCM, build, issue tracking, etc.), but the resistance escalated substantially when other functions proved inadequate.

Within two years of insourcing application development, development cycle times had been reduced from months to weeks. But this development acceleration exposed a major impedance mismatch between operations and development teams. Inefficiencies in the outsourced operation's ticket handling, system provisioning in both development and production environments, deployment, change management, and monitoring were exposed as major bottlenecks as well as a source of tension between the applications and operations

Table 1 — "Devops" in a Nutshell

Increased collaboration between operations and development
Reduced cycle times for operations activities (e.g., provisioning, deployment, change controls)
Extreme focus on automation in tools and processes
Fostering continuous improvement as both a means to the above and as a strategy to adapt to increasingly rapid changes happening in IT

organizations. Driven by increasing demands for velocity and the inability of the existing operational model to service these needs, the DSG took on some capability historically reserved for operations.

A number of obstacles dogged the organization as traditionally operational roles were augmented and transformed. The infrastructure groups feared loss of control without loss of accountability and felt constrained by the terms of the outsourcing contract. Finance was concerned that the organization was "paying twice" for the same services. Compliance watchdogs didn't know how to interpret DSG's role in the new landscape. Isolated development teams fought the loss of their "oilers"[2] to automation.

This dual-reporting tension was necessary to ensure that both the project's and the larger organization's goals were met.

Focusing on Results

By focusing on project delivery, sticking to solving problems that impacted development productivity, and paying close attention to cultural issues, the DSG was able to sidestep objections and deliver results. Experienced leaders in the group knew the devops approach would yield agility, flexibility, and cost benefits, but they had no way to quantify those benefits *a priori*. Now, with five years of experience and metrics behind them, the DSG has been able to substantiate the value proposition. Their experience and lessons learned are reflected in the key decision points and guiding principles that follow.

GUIDING PRINCIPLES FOR THE DSG

In a period of 15 months between 2006-2007, the newly formed DSG grew within a development organization that went from 50 people and eight projects to 175 people and 25 projects. Within another two years, the development organization's size had tripled again. In order to maintain efficiency and effectiveness during this rapid growth, the DSG focused on three principles:

1. Centralize
2. Remain project-focused
3. Automate wherever possible

Not only did these principles keep the group focused, but they also reinforced the value of the devops approach throughout the organization. Corporate stakeholders intuitively accepted the value of centralization and automation for cost containment, and project stakeholders accepted the sharing of project resources in exchange for the emphasis on project delivery, skill, and automation.

Centralize Capabilities as Practices

An obvious way to achieve economies of scale and consistency is to centralize. The DSG initially consolidated practices around SCM, testing, and systems analysis by hiring a practice lead for each area. The practice lead was responsible for delivering skilled staff to project teams, managing staff, and developing best practices and procedures to be used across projects. Initially, project leadership resisted allowing the practice leads to hire for their teams, but by letting the experts hire their own kind, the practice leads took more ownership of project problems and recruited more qualified staff.

Remain Project-Focused — A Hybrid Matrix

In order to neutralize the tendency of centralized organizations to lose touch with those they serve, the DSG staff was organizationally managed by their respective practice leads but were funded by the project teams, embedded into the project teams, and operationally managed by the respective project manager or other project leadership (see Figure 1). This dual-reporting tension — a practice boss (from DSG) and an operational boss (from the project) — was necessary to ensure that both the project's and the larger organization's goals were met. This structure reinforced important values: a commitment to delivery, a bidirectional conduit to harness innovation from the floor and introduce it back through the practice to other project teams, the ability to hold to standards in the face of delivery pressure, and visibility into project team risk and performance through multiple management channels. It also helped resolve funding issues common to shared services groups: project teams were required to fund their own DSG resources, even though they were recruited and managed by DSG, while simultaneously holding the practices to delivery expectations.

Automate and Manage to Metrics

While the automation of development procedure and the generation of performance metrics tend to drive efficiency and consistency, development teams often fail to automate. Their reluctance is understandable — among the things that preclude automation are a lack of money for tools, project urgencies that trump the investment of time needed to code procedures, or a bias toward hiring nonprogrammers into support roles in order to save

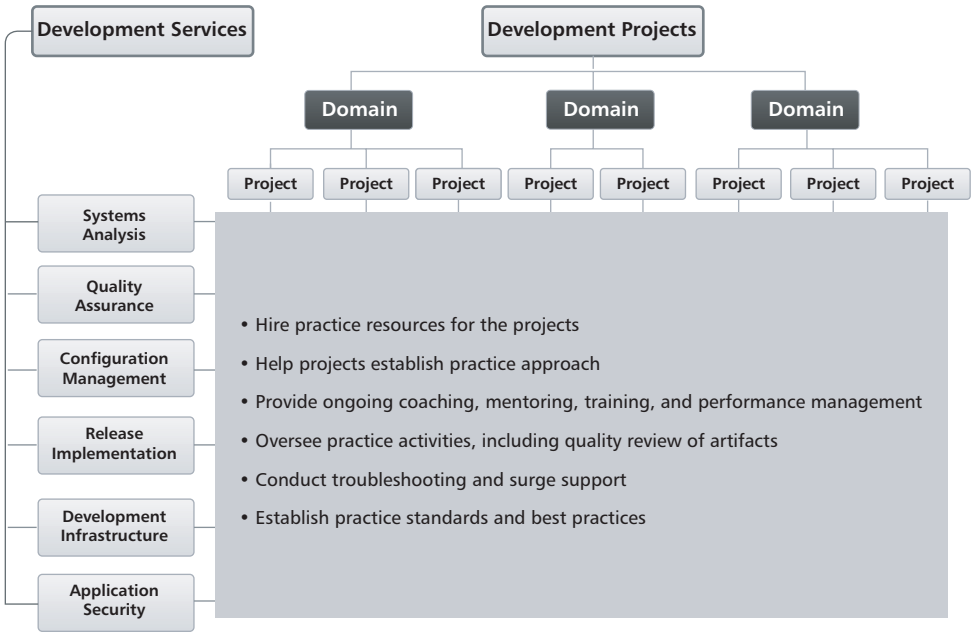

Figure 1 — The DSG engagement strategy.

money. The DSG eliminated these obstacles by investing in tools, affirming the importance and expectation of automation (tracked by metrics and published in dashboards), and ensuring that skilled professionals were hired for crucial service roles. All this was provided as a service to the delivery teams.

EARLY "WINS" AND PRACTICE LEADERSHIP

Because the DSG represented a significant change to the organization's structure (i.e., by introducing matrix practices), the group needed to show stakeholders early successes. A focus on recruiting technical specialists became the single most important step on the road to building DSG credibility and acceptance. The DSG hired staff with a particular profile: a high affinity for automation, a strong sense of service to the development community, a considerable track record of delivery, and expertise in their practice area.

Practice staff were targeted to key projects that had influence across dimensions such as platform (.NET, Java), size, maturity, and importance to the organization.[3] Despite being placed and managed from the center, the embedded practice staff was held accountable for delivery as part of the development team in order to defuse sensitivity to remote control from the outside. This focus on accountability helped to mitigate the key weakness of a pure matrix organization — the drop-in engagement model — which doesn't breed trust or enable the tight communication loops common to high-functioning teams.

As practice staff merged into the development streams of key projects, their unique positions allowed them to observe patterns across the organization that were hindering development and to identify opportunities to introduce improvements (see Table 2). The opportunities and solutions initially identified by each practice were captured in a "state of the practice" roadmap — a set of actionable initiatives determined by attempting to solve delivery issues with the project teams. These roadmaps were used to get concurrence and set expectations with management and peers and to hold practice leads accountable for improvements to their practice.

By building a track record of problem solving, both project-centered and enterprise-wide, the practice leads were ultimately able to move services out of development teams and into the center. Key examples of this were the centralized automated build infrastructure, automated test frameworks, and, later, monitoring

Table 2 — The Lifecycle of a Practice

Need	Identify a common capability whose lack of consistency or maturity is an obstacle to goals.
Lead	Recruit a practice lead to take charge of this capability in a hybrid matrix structure.
Breed	Execute by staffing the capability for real project needs.
Read	Build a roadmap of improvements, informed by a "state of the practice" assessment.
Seed	Harvest capability and back-fill to other portfolio projects.

and automated deployment capability. This harvesting of solutions that can be matured and sustained, then reintroducing them back into the rest of the enterprise portfolio, had additional benefits. It enabled flexibility, allowing the DSG to adjust staff appropriate to the ebb and flow of the organization's needs, and it formed the basis for standardization and intellectual property retention in a company that had a high ratio of transient contractors.

CAPABILITIES, SERVICE, AND VALUE

Three practices made up the initial DSG: SCM, quality assurance (testing), and systems analysis. While quality assurance and systems analysis practices had a significant positive impact on development teams, their impact on operations was limited. The SCM practice, however, played a key role in the transformation of the development/operations juncture, both directly (by driving automation) and indirectly (by supporting additional areas that led to further operational transformations — release implementation, application security, and development infrastructure). Two factors contributed to the SCM practice's pivotal role in devops innovation:

1. A tool platform useful both to development and operations
2. The consolidation of interaction between development and operations

The SCM toolset provides integration opportunities for capabilities beyond code management and build. Some of these are development-focused (e.g., test automation, issue tracking, and code metrics), while many of them are also operations-focused (e.g., deployment and change management). Figure 2 highlights the pivotal role of the automated build platform.

In addition, SCM activity provides an opportunity to facilitate and focus the interaction between development and operations. Prior to centralization, the interaction between development teams and operations was strained and frequently erupted into conflict. The explosion of development activity overwhelmed the few operations staff assigned to coordinate environment changes. Left to fend for themselves, and without clear procedures or an understanding of operational concerns, development teams made matters worse. Because the DSG spoke both dialects — development and operations — it was able to step into the middle to relieve the tension. Operations appreciated having a single, disciplined point of contact to development teams, and the development teams appreciated having responsive, high-level services. This confluence of support created an opportunity for real operational change.

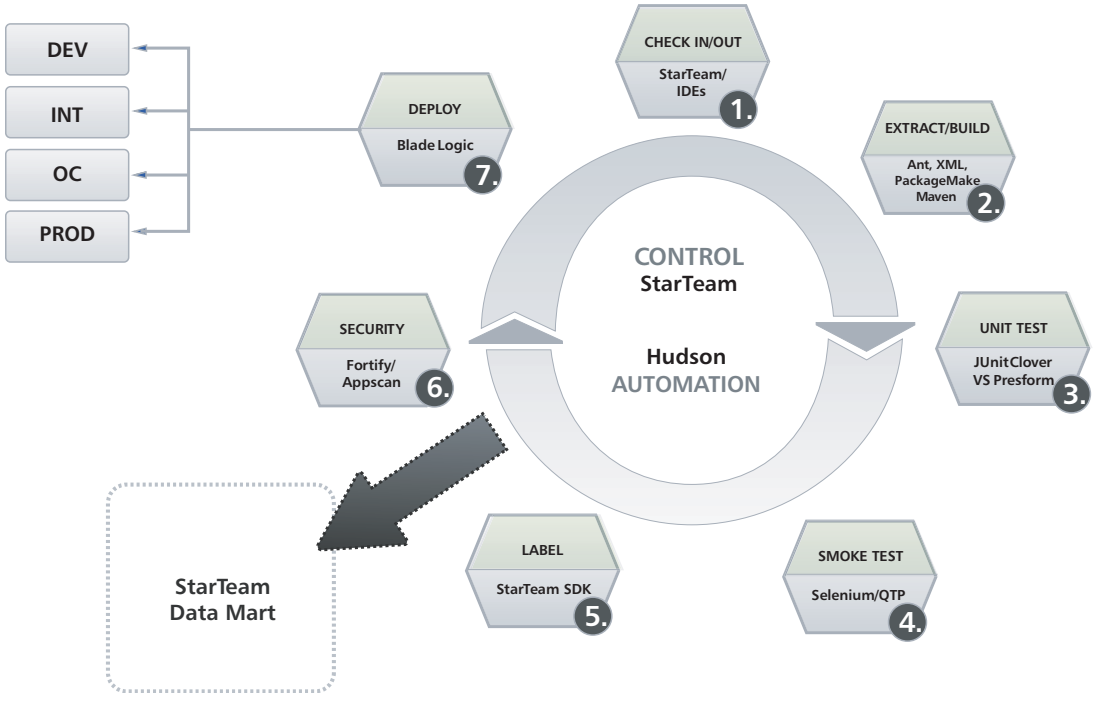

Figure 2 — Automated build as a hub for other development and operations support activities.

INNOVATION AT THE DEVELOPMENT/OPERATIONS BOUNDARY

With an automation mindset and the weight (priority and resources) of a number of business-critical development projects, the development organization, with its DSG, gained significant leverage to drive development-style change into operations. The following sections highlight *some* of the more significant impacts of the practices at the operations boundary.

Software Configuration Management

SCM provides the safe house for software created as the result of portfolio investment. Control over software configuration provides transparency through build and code-change metrics. A strong SCM capability enables rapid deployment of new projects and serves as a cornerstone for interaction with the operations boundary. Failure to perform sound SCM leads to fragmentation and loss of control of the corporate asset, as each team operates in a vacuum. In order to ensure development soundness, the DSG had to improve configuration management, but gaining control of SCM drove another important innovation: the creation of an automated build infrastructure.

Before the DSG took control of SCM, the configuration management function was sclerosed inside the outsourced operations group, and the configuration management engineer (CME) had become an administrator to facilitate the numerous tickets necessary to generate builds and deployments within the tightly controlled production environments. In the new embedded technical resource model of the DSG, the SCM practice worked inside the development team to build and deploy software, relieving the rest of the project team to focus on delivery. This "build as a service" offered by the practice lead encouraged teams to migrate their standalone efforts to the center in a short period of time and facilitated the development of an automated software build infrastructure (see Table 3). Within a year, the service had broad penetration within the portfolio. The standardized build structure and metadata enabled metrics generation from a data warehouse without imposing a burden on teams. Initially, the metrics platform provided tracking of the build from inception to staging, and it was later extended to catch metadata from other automation efforts, such as unit test and test case code coverage. This gave the organization unprecedented transparency into the development activities of multiple projects.

Centralizing SCM offered both qualitative and quantitative value (see Table 4). An initial investment of US $500,000 launched the capability, and other than the practice lead, all costs were borne by the projects being served. Taking into account only project setup, this practice yielded annual cost avoidance of over $1 million per year, not counting indirect savings to project teams related to service delays. In addition, consolidating this practice drove efficiencies in hardware acquisition and maintenance by avoiding project-specific build plants and captured staff efficiencies. By restructuring SCM, the CM full-time employee (FTE)–to-project ratio increased by a factor of five. Estimated using staffing ratios, the savings to the organization from 2008 onward are greater than $5 million per year.

Application Security

The flexibility and agility afforded by having a centralized, automation-focused SCM group that serviced all of development enabled the rapid addition of new capabilities, such as application security, at a cost that was substantially less than alternative options.

In response to increasing awareness of data security concerns, changing legislation related to protection of personally identifiable information (PII), and lessons learned in the financial services industry, the operations organization proactively sought to mitigate the risk of security flaws in the substantial quantity of custom code that ran the business. External consultants

Table 3 — Characteristics of the SCM Practice

Labor	2%-3% of development labor budget
Activities	Approximately 50 concurrent projects, 500-plus developers, 12 million LOC, 9,000 builds, 350 releases, several organization-wide initiatives
Artifacts	CM plan, build manifest, metrics, standard automated build, and CM structure
Tools	StarTeam, Hudson, JUnit, Ant, Clover, VSPerfMon, Maven
Metrics	Build time, KLOC, test results/coverage, deploy time

Table 4 — SCM/Build Metrics

	Before	After
Effort per instance setup (30/yr)	3 person-months	0.5 person-months
Annual cost per instance setup	$1.5 million	$250,000
Projects per CM FTE	2:1	9:1
SCM as percentage of application development staff	10%	2%

proposed identifying and remediating security flaws with standalone solutions that were expensive and failed to provide a means of institutionalizing security defect prevention because they relied on experts to both assess and perform the remediation.

Instead, the DSG partnered with the organization's Corporate Information Security Office (CISO) to establish a "bureau model" tied to the SCM platform. Automatic code scanning[4] was integrated into the build system, and two FTE application security experts tailored the scanning to match the conditions in the code base and the company's security policy. Issues identified by scanning were screened by the experts and fed into the standard issue-tracking system. The CISO could then prioritize issues for project teams, and the security experts were available to help teams handle complex remediation.

The bureau model enabled the DSG to provide application security remediation for the 31 critical corporate applications at almost one-quarter of the cost of the solutions security consultants had proposed (see Table 5). In addition, because of the high degree of automation, DSG staff were able to partner with the CISO, giving them improved visibility and participation in the process and allowing them to fulfill their mission with little administrative effort. Finally, the centralized application security practice had a considerable impact on application development teams. Because these teams were accountable for fixing security defects detected as a result of the security scanning, their security knowledge increased naturally out of the desire to avoid problems. Thus, application security practice was institutionalized as a routine part of the development process at a sustainable cost.

Table 5 — Application Security

	Standalone	Integrated
Total annual cost	$1.9 million	$540,000
FTE	9	2

Table 6 — Requests for Change (RFCs) Metrics

	Before Portal	After Portal
Emails	34,000	3,800
Effort hours	1,750	400

Change Management Interface

Rapid organizational growth and the proliferation of development teams inexperienced with change management processes in the production environments also led to problems. In one example caused by broken change notification procedures, a scheduled software change would have inadvertently taken a 250-person unit offline. So much information flooded the system that it was nearly impossible to monitor everything. Each of the 1,800 change tickets per year for the upper environments generated 12-15 broadcast emails to coordinate dozens of development teams and six different infrastructure teams. Larger application teams had staff dedicated to watching the email stream for changes affecting their applications and servers, but less robust teams often were surprised.

As a service to the application teams, the SCM practice developed a portal that took a daily data feed from the production ticketing system and aggregated key information around upcoming releases, application contacts, and server inventory into an approval dashboard. By displaying only relevant tickets for each project contact, the dashboard saved time and avoided mistakes (see Table 6). This portal application helped solidify the DSG's position as intrinsic to the organization's success.

Release Implementation

Concurrent to the implementation of the build infrastructure and ticketing portal, a number of project teams were investing in automating their software deployments. As build automation increased the rate of software delivery, the services offered by the outsourced operation were unable to meet installation demand for volume, speed, and reliability. The DSG formed a nascent release implementation practice by pulling key release staff and practices from the application team that was most advanced in this regard. After consolidating release services for a number of development teams, this new practice began an initiative to introduce an automated deployment framework as a service offering. The outsourced operations release team responsible for deployments was hesitant to make any changes but was eventually recruited to the effort.

Within six months, the first proof of concept was in place using BMC's BladeLogic product, which was subsequently selected as the organization's platform for automated deployments. In addition to enabling concurrent code drops to multiple environments, the process consolidated all activity into one ticket and six release teams into one. A release manager is assigned to, and gains familiarity with, a significant portion of

the application portfolio, thus eliminating development team labor effort. Because the operations team was able to reduce its staff from 12 to eight positions, an initial investment of approximately $350,000 and an annual support cost of $300,000 are more than covered by cost savings (see Table 7). This infrastructure and team now support over 1,500 deployments per year without staff increase — a feat that could not have been achieved without automation.

Table 7 — Deployments: 2,000 per Year

	Before	After
Effort per application	1 day	20 min
Annual cost	$1.7 million	$85,000
Release team size	12	8

RESOURCE FLEXIBILITY: ABSORBING A MERGER

The consistency and level of automation implemented by the DSG were tested when one of the company's business units acquired and merged with a similar business unit from another company, which had similar but different business processes and was supported by a different application portfolio. During the two-year initiative to merge more than 80 applications into the existing portfolio, the development organization recruited an additional 125 FTEs and executed an additional 30 projects concurrently.

The practice structure distributed the hiring load and shortened the on-boarding time of new staff as a result of the standardization and automation of all back-end activity. Most critically, the operations boundary was previously a massive time sink for new teams, but with the DSG practices established and automated, such teams could focus on business reconciliation and wasted no time on operational tasks. Just as importantly, the organization easily scaled back staff in response to the reduced post-merger workload without loss of productivity or knowledge.

TAKEAWAYS

Given the nature of the IT organization in question prior to 2005, a devops approach driven from within the application development organization was the only viable path to achieving the stability, predictability, cost savings, and productivity gains realized in this case study. The conventional organization structure of separate "apps" and "ops" hindered operations from recognizing and acting on their responsibility to adopt emergent capabilities to better support development. The concept of such hard and fast boundaries is inherently broken and counterproductive to IT's mission to support the business.

At the time, operations was limited by its contract with a large outsourcer that was marching to a cadence from a different era and had repeatedly demonstrated outright hostility to change. The inhouse infrastructure and operations organization wanted to help but had its hands tied by its limited staff, small budget, and oversight role toward the outsourcer. In addition, this group had a compliance burden that discouraged innovation at small scales — any solution they implemented had to scale instantly to the entire enterprise or risk audit findings. Furthermore, because they were not accountable for the success of (and were at arm's length from) application delivery, they didn't experience the consequences of high-latency and low-consistency legacy operational procedures. Therefore, they had no compelling reason to adjust their or the outsourcer's practices.

The conventional organization structure of separate "apps" and "ops" hindered operations from recognizing and acting on their responsibility to adopt emergent capabilities to better support development.

By contrast, the development organization was less encumbered by the status quo, was directly accountable for software delivery, and was so severely impacted by operations and infrastructure latencies that it was in its interest to fix the bottlenecks. The development group also had the labor budget and flexibility to hire hands-on development engineers to solve problems. As a trump card, the development group could lever the political weight of the business units it supported to apply additional pressure to force change. Given the corporate aversion to risk, the DSG could justify a centralized approach on the basis of providing consistency and predictability to development. Fortuitously, the consolidated development practices provided a focal point to connect the application delivery organization with operations. The DSG's emphasis on automation and scalability meshed well with operations for capabilities that were natural extensions of the development support activities.

When the organization's journey began, development leadership was convinced that a centralized approach

was the right path toward large-scale, reliable delivery and was allowed the time necessary to quantify the benefits. The DSG promised consistency and transparency, consolidated points of control, risk reduction, and economies of scale, and it was able to attain those goals in a relatively short time. We cannot overstate the importance of the hybrid matrix organization model to maintaining a primary focus on delivering for projects, while at the same time improving practice across the whole organization and putting downward pressure on the tendency of both project teams and centralized services groups to bloat their ranks.

Quantitatively, the DSG realized significant results. The group has:

- Enabled increased throughput without increased costs
- Maintained low startup costs for new and evolving capabilities
- Decreased latency and administrative burden
- Reduced error rates
- Realized both license and infrastructure savings
- Gained labor efficiencies by sharing resources across projects and replacing humans with automation

The DSG also realized qualitative results by:

- Sharpening hiring skills
- Standardizing procedure
- Maintaining flexibility in scaling up and down as needed with minimal loss of discipline
- Increasing operational stability
- Providing transparency, which frequently mitigated risks and averted costly failures

While the economies of scale offered by the DSG's relentless focus on centralization, project results, and automation are considerable, perhaps the greatest value to the organization of automation at the engineering and operational boundary is increased development productivity. A leading outside consultant benchmarked[5] a number of the organization's projects in 2008 and again in 2011. The results show that these methods contribute to a substantial difference in performance (see Table 8). In 2008, the benchmarks showed projects hovering around industry norms for size, cost, and quality. The 2011 numbers contrast markedly, showing factor-of-two improvements in cost and effort and a five-fold improvement in quality.

Table 8 — Fraction of Industry Norms

Project cost	55%
Schedule	50%
Effort	65%
QA defects	20%

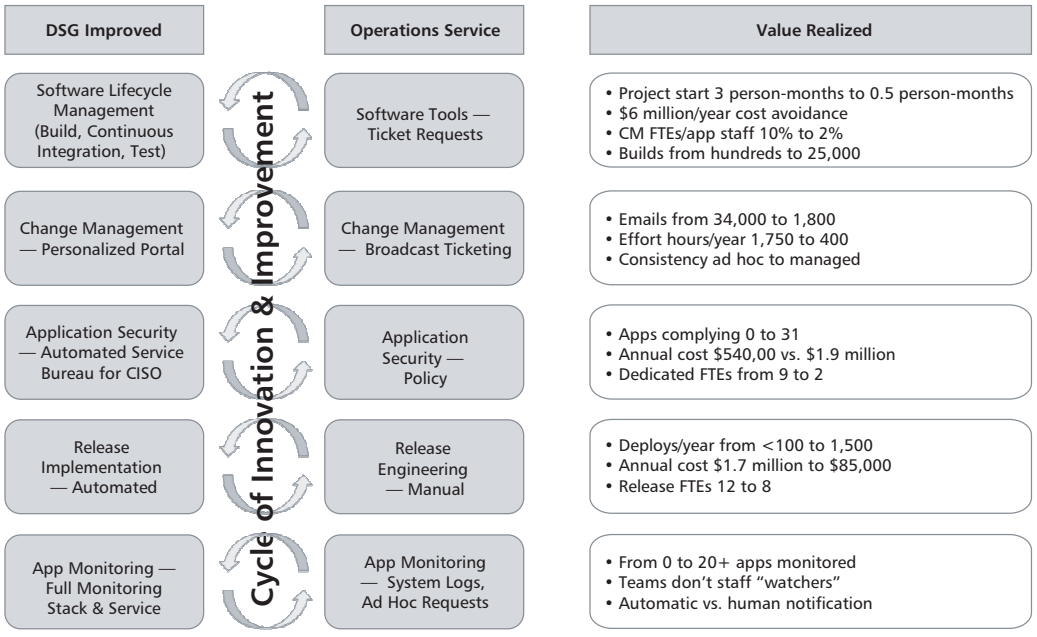

Figure 3 — Some of the inefficient and inadequate operations services improved by DSG and the value realized.

The case for investing in devops in 2005, when this development organization formed its DSG, was based on development dogma and speculation. Since then, this devops group has collected a number of concrete measurements of its incremental value (see Figure 3). The greatest value by far is its contribution to a doubling of productivity for this company, which spent many tens of millions of dollars per year in new development during this time. We hope that this case study will provide support to others seeking to justify adopting devops practices and that, in short order, devops will become the norm within the industry.

ENDNOTES

[1]Fitzpatrick, Lawrence. "How Craftsmanship Survives Explosive Growth." *Cutter IT Journal*, Vol. 23, No. 4, 2010.

[2]During the industrial era, an army of workers wielding oilcans constantly tended to heavy machinery in order to forestall breakdown. One can draw a parallel to operations and maintenance personnel performing manual break/fix production work.

[3]This model is similar to the military approach of assembling unique capabilities from various units to form teams for specific missions.

[4]A number of vendors provide tools for scanning source code for security flaws.

[5]Mah, Michael. "IT Organization, Benchmark Thyself." *IT Metrics Strategies*, Vol. 6, No. 3, 2002, pp. 1-6.

Lawrence Fitzpatrick has a record of software delivery and transformational software management that spans academia and government, commercial software product development, and corporate IT over 30 years. Currently, Mr. Fitzpatrick is President of Computech, Inc., a leading application development and consulting firm headquartered in Bethesda, Maryland, USA. Until mid-2009, Mr. Fitzpatrick served as Senior VP of the Financial Industry Regulatory Authority (FINRA). During a time of significant change in the financial sector, he grew and ran an application development team of over 500 individuals and was accountable for the delivery of numerous key systems. Prior to joining FINRA in 2002, Mr. Fitzpatrick led delivery of commercial software products as VP at Open Text Software (OTEX) and as CTO at Personal Library Software. Mr. Fitzpatrick has degrees from the University of Virginia and Georgetown University. Mr. Fitzpatrick can be reached at lfitzpatrick@computechinc.com.

Michael Dillon has led government and corporate IT and application delivery over the past 20 years. For the last five-plus years, Mr. Dillon has served as Associate VP in charge of a creating, growing, and managing the development services group for a leading financial services regulatory firm. Prior to his present position, Mr. Dillon built and ran multidisciplinary matrix teams that supported application development needs in the government services arena and led a number of quality functions within government and the airline industry. Mr. Dillon can be reached at mchldillon@comcast.net.

Next-Generation Process Integration:
CMMI and ITIL Do Devops

by Bill Phifer

Computer operations used to be a lot simpler. I recall my first several IT programming jobs, where the computer room was only a set of stairs away. You could actually watch your test job running and even mount the data tapes for it yourself. A production problem could be handled in person (even if you had to get out of bed and drive to the office first) and was usually diagnosed quickly from a system return code or through a printed core dump to find the offending instructions.

THE GROWTH OF IT COMPLEXITY

Today, that's all changed. You may be modifying a computer application that was developed on one platform in India, was implemented by a California release team, now runs in a complex distributed global network of servers, and is monitored from Argentina. You probably have little idea about the physical runtime environments for the executable, but you still need to ensure that your business customer's service levels for availability are maintained.

There are many more challenges now for developers, starting with the cold reality that you probably aren't allowed access to your data center or have never even seen it. The silos and relationships between developers and operations have never been more separate and distant. Cloud computing further abstracts the connection points between infrastructure and applications by adding more virtualization.

These may not represent the biggest problems, however. Solutions often are not designed for the multi-tiered architectures, complex technologies, and platforms in the runtime environment. They are developed without input from infrastructure and operations teams and as a result often run inefficiently and are difficult to maintain. There are few cross-domain experts who are familiar with both development and operations, and fewer still who can also fairly represent the business. Development and operations teams also have conflicting priorities — developers are evaluated on their ability to deliver more enhancements, more releases, and more application components, while operations staff are driven to reduce and eliminate outages.[1] These are among the many reasons why the devops movement is timely and relevant, since it aims to break down the barriers that stop organizations from effectively delivering working software to the business at the pace of the business.[2]

Historically, many major outages and implementation failures result from a lack of communication between applications and infrastructure groups that don't participate effectively within a complete service management framework. Other examples of integration failures include:[3]

- Communication gaps between delivery groups
- Critical errors that are not resolved in a timely manner or are even ignored
- End-to-end change management that is not effective or is nonexistent
- Poor transition from development to production
- Apps and infrastructure engineering that don't sync, don't collaborate, and don't learn from one another
- Infrastructure that does not always have a strong project management culture, being more reactive than proactive
- No central governance for strategy, architecture, and standards

Best practice reference models provide discipline to development teams (e.g., CMMI for Development [CMMI-DEV]) and to service management/operations (e.g., ITIL V3), but no one model crosses the chasm. CMMI-DEV is not descriptive enough of the operational environment, and ITIL V3 does not really attempt to address application development.

CMMI AND ITIL PLAY TOGETHER FOR DEVOPS

Devops is concerned with addressing this challenge through better collaboration between development and operations, but how? To begin, we can take a longer look at best practice models such as CMMI-DEV and ITIL V3.

The CMMI for Development[4] provides support for the software development lifecycle (SDLC) through disciplines that include requirements development, estimating and project planning, product quality, configuration management, and other processes such as measurement and training. However, the SDLC and best practices in this case end at product integration and validation testing within CMMI. This model is strong in systems and software engineering, but weak in transition.

This is essentially where ITIL V3 picks up in sufficiently describing service design and transition into runtime and subsequent operations for the completed application. ITIL was written as a means of enabling and managing services that add value to the customer's business by providing a common, proven approach. The complementary aspects of these two models are evident, but do they dance together? I would answer "yes." Each has strengths, but also weaknesses, as indicated in Figure 1. But taken together, they make each other more effective.

Looking at CMMI and ITIL V3 processes and disciplines together allows us to exploit the strengths of each model, but the synergies are most effective when their best practices are fully integrated. What this means is that applications must be developed not only to support intended business processes, but also with full consideration of what's needed to effectively and efficiently sustain the application over its life. To do that, clearly IT operations must be engaged much earlier in development and there must be a system-wide culture change, giving development and operations more visibility into each other's activities. This is not just a tooling or communication question, but an attitude adjustment. This is one view of the devops movement.

Understanding the possibilities for process integration between development and operations is most pronounced within the engineering process areas of the CMMI for Development.[5] These five areas — Requirements Development, Technical Solution, Verification, Validation, and Product Integration — provide a framework for applying and integrating service management processes from ITIL V3 for the greatest benefit. Figure 2 describes a straightforward SDLC based on CMMI engineering process areas and indicates relevant integration points for applicable ITIL V3 processes.

IT organizations must effectively perform this process integration in order to ensure that applications are designed and built to run in the complex IT environment demanded by global and Internet business.

DEVOPS BEGINS WITH REQUIREMENTS DETERMINATION

The devops movement has clearly stressed a need for collaboration and integration[6] — specifically that development and operations should design, develop, build, and deploy software together.[7] This may be easy to say, but it is more difficult to accomplish. What must change first is that there must be new and detailed bilateral communication between development and operations during the requirements determination stage — not as a final activity within deployment. And let's add the business unit itself, since in addition to business requirements, IT must elicit and negotiate the following among the business, application development, and

Figure 1 — CMMI and ITIL scope targets and strengths.

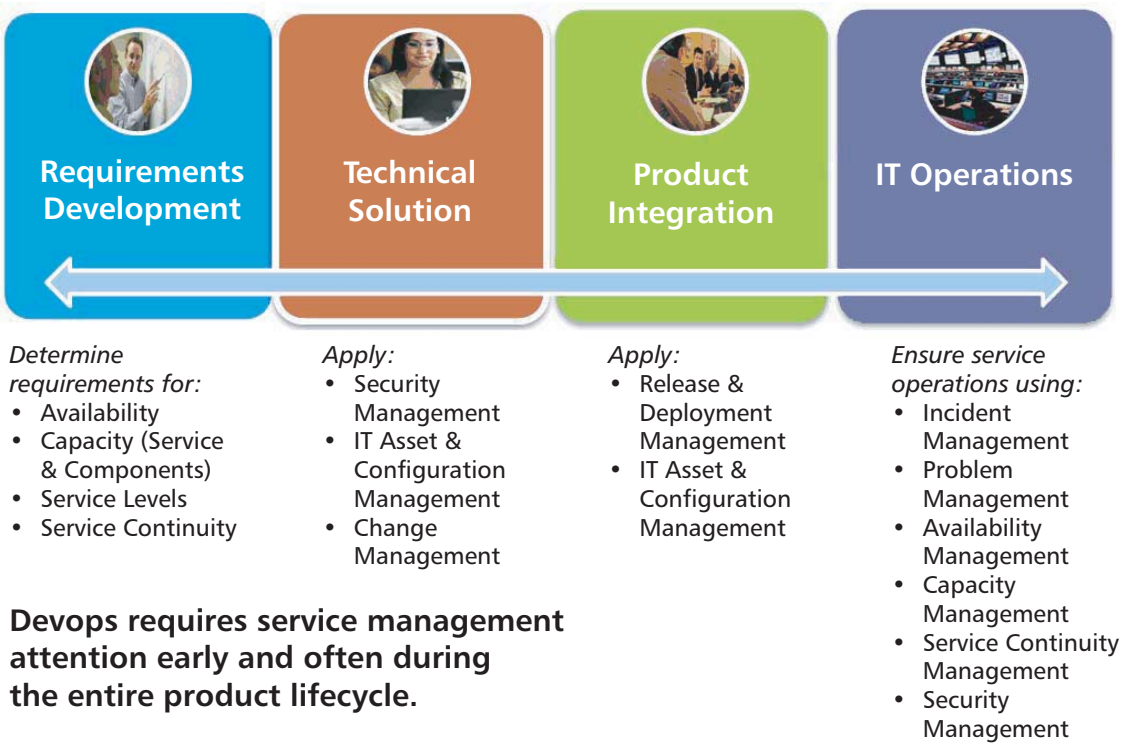

Figure 2 — CMMI engineering lifecycle with key ITIL interactions.

infrastructure operations, using several ITIL V3 processes[8] to drive the discussion:

- **Availability Management.** Understand and plan for the resources needed to ensure the application performs at levels specified or needed by the business. What hours and days must the service supported by the application be available? What resources and components (experts, knowledge, networks, environments, service desks, etc.) are required to maintain and sustain it? An availability assessment with operations may be required to understand what is needed, but clearly the requirements have to include these types of needs.
- **Capacity Management.** As is the case with availability, there are three levels of requirements — for business, service, and component capacity. The business capacity is largely determined by the current and targeted market and drives the required service capacity needed by IT to support the business. At the application level as part of requirements gathering, IT must understand the capacity needed to deploy and maintain the service, including component capacity. How many users? How many database records? What network bandwidth? How many virtual or physical servers? In short, the requirements must specify the existing infrastructure capacity necessary to satisfactorily support the application.
- **Service-Level Management.** Service-level requirements in support of business objectives should also be included in the requirements gathering and identified as constraints on the design. New applications and major changes can easily impact system response times, system availability windows, and even service desk response and resolution if training and documentation requirements are not established up front. This is the time to work with operations to ensure that IT component capacity is sufficient to achieve agreed-upon service levels, or these may have to be renegotiated.
- **IT Security Management.** Security has to be built in; responding to attacks, unauthorized accesses, and theft of data and intellectual property after they occur is expensive and embarrassing. IT must define security requirements and preventive measures to resist vulnerabilities and threats, as well as meet security and privacy compliance regulations such as HIPAA and PCI DSS — all prior to design and build.
- **IT Service Continuity Management.** We all understand that today's business applications run in a complex and widely distributed but interconnected and largely virtual environment. As seen recently with some highly publicized public cloud outages, applications are perpetually exposed to risks of service interruption; in the global enterprise, these

types of problems are unacceptable. In conjunction with understanding availability needs, application developers and operations personnel also need to understand critical points of possible failure and set requirements to design appropriate mitigations such as warning trigger points, automated backups of critical data, and built-in redundancies into the architecture and application when appropriate.

DEVOPS CONTINUES WITH DESIGN, BUILD, AND TEST

Still following the CMMI for Development lifecycle, the remaining engineering process areas (Requirements Management, Technical Solution, Product Integration, Verification, and Validation)[9] also require devops-style collaboration among the business, application developers, and operations, as follows:

- **Change Management.** Few complete sets of requirements or design solutions survive the SDLC intact. Changes happen — repeatedly — and this is only one factor spawning the agile movement. Application design solutions must undergo change review by infrastructure engineering and operations groups to ensure that they do not adversely impact current operations and that the appropriate runtime infrastructure is available with the necessary capacity to support the change and achieve required service levels. The implementation should also be included in what ITIL calls a "Forward Schedule of Change" to ensure coordination with other planned infrastructure change and maintenance activities.
- **Capacity Management and Availability Management.** As the design evolves, additional assessments may be required for capacity and availability through modeling, simulations, prototypes, and design walkthroughs to ensure that the target infrastructure can sustain the application in operational steady state. The closer the integrated system test environment can mimic the live environment, the better. It is unrealistic to expect that system performance with large volumes of data will be the same for a global distributed application as it is in a small clustered data center or environment.
- **Configuration Management.** ITIL V3 takes this process well beyond the basic change and version control discipline expected by CMMI, which focuses primarily on applications. Here Configuration Management defines the relationship between all the components involved in providing each service the business consumes. For this, applications staff must provide missing data and ensure that it is maintained.
- **Release and Deployment Management.** Operations staff hates failed implementations and having to back out and reschedule changes. They also are unhappy being unable to effectively support newly deployed applications due to lack of runtime instructions and shared knowledge. The solution to this problem is early and frequent involvement of operations staff in the planning stages of major new releases.[10] Release and Deployment Management takes a system view well beyond what CMMI typically expects, suggesting that release documents and objects be provided and traceable back to corresponding change records.

Operations staff hate failed implementations and having to back out and reschedule changes. The solution to this problem is early and frequent involvement of operations staff in the planning stages of major new releases.

DEVOPS DEPENDS ON COLLABORATION IN SERVICE OPERATIONS

The purpose of IT operations is to coordinate and carry out the activities and processes required to deliver and manage IT-based services at agreed levels for the business.[11] The applications staff clearly has a role to play in this to ensure that IT helps the business achieve its objectives. In this phase of the lifecycle, devops principles are not so much concerned with how the inherent ITIL V3 service operations processes are implemented as the extent to which developers are involved.

In several processes in this phase, developers can provide operations with additional knowledge and understanding that is not currently offered or required. For example, the following information should be included as part of changes and releases:[12]

- Application components with their interdependencies
- Directories to locate and store binary files and application-related files
- Scripts to run as part of installation, or to start or stop the application
- Scripts to check hardware and software configuration of target systems before deployment or installation
- Operational and support requirements
- SLA targets and requirements related to the application

- Specification of metrics and events that can be retrieved from the application and that indicate the performance status of the application
- Specification of the details required to track an application's major transactions

Event Management could benefit from having applications personnel actively monitoring normal operations and detecting and escalating exception conditions. Further, these individuals can help with the development and installation of monitoring agents and sensors to trigger response. Similarly, Incident Management and Problem Management require application developer involvement to quickly restore service and also to assist in analyzing the root cause of problems, so as to resolve them permanently and prevent future problems. Other ITIL processes already mentioned, such as Configuration Management and Change Management, are fully enhanced when developers are completely engaged and the devops principles are supported.

DEVOPS-RELATED QUESTIONS FOR DEVELOPERS

- Has operations been included in your change advisory board meetings, and is your application change included in the Forward Schedule of Change in order to coordinate with operations maintenance and other change activities?
- Have you determined the required availability of your application to the business and ensured the availability of resources (such as databases and licenses, as well as operations staff with appropriate knowledge) to support it?
- What capacity requirements (storage, processor, network, database size, etc.) do you require from infrastructure engineering, and have you used a simulation model or other approach to confirm them?
- What possible impact will your developed application (or major modification) have on negotiated service levels? Will these need to be modified?
- Have you considered security and privacy vulnerabilities? Are there credit card numbers (PCI DSS) or patient records (HIPAA) involved? Will offsite backups of this data need to be encrypted?
- Has operations reviewed your release and deployment plans?

KNOWLEDGE MANAGEMENT DRIVES DEVOPS

Knowledge management is the process responsible for gathering, analyzing, storing, and especially sharing knowledge and information within the organization.[13] It is a tenet of devops and requires an open culture for sharing knowledge and the infrastructure to support it. The implication is that processes supporting devops must include activities directing interaction between application developers and operations, so they can work together to effectively support the business. This also requires identifying knowledge gaps and improving processes to include necessary knowledge sharing, critical cross-silo reviews, and training.

CMMI FOR SERVICES VS. ITIL V3

Many process experts may be looking to the SEI's CMMI for Services (CMMI-SVC), V1.3[14] as a single guide to combining the strengths of CMMI with those of ITIL in a way that also supports devops principles. While several of the ITIL V3 processes are represented within this model, it is my opinion that CMMI-SVC currently does not provide sufficient guidance to address the devops issues discussed here. However, since ITIL V3 has only limited support for the software development lifecycle and CMMI-DEV does not address service management, CMMI-SVC may offer at least a limited model that could be useful for appraisal and evaluation. The approach I recommended is to use guidance from CMMI-DEV for application development and ITIL V3 for service management and service operations, and consider CMMI-SVC as a reference model for evaluation.

EFFECTIVE CMMI-ITIL INTEGRATION YIELDS BUSINESS-IT ALIGNMENT

While developing applications following the CMMI model may often be helpful in managing development environments and projects from a software- or systems engineering–centric view, it is entirely more effective when complemented by an IT management model such as ITIL. However, the simple act of mapping the CMMI and ITIL functions to each other provides only a conceptual view and doesn't get to what is needed at the execution level. After all, an IT organization could be rated at CMMI Level 5 and have practices fully aligned with ITIL, yet still have operational problems that cause it to fall far short of achieving the required value for the business.

The result of effective integration between CMMI and ITIL as applied to the devops challenge is a framework that ensures IT is aligned with the needs of the business. This framework is supported in application development through applications that are built on CMMI engineering principles but are integrated with ITIL V3 service management processes. The latter are applied to ensure that the applications can be effectively deployed and managed in the targeted operational environment and be sustained with ITIL Service Operations disciplines. The outcome is a service lifecycle that leverages the individual strengths of each model — CMMI for systems and software engineering and ITIL for clear business alignment, effective transition, and sustained operation of the IT environment. An integrated lifecycle and associated collaboration provide for the effective sharing (both push and pull) of information between application development and operations, allowing both groups to approach IT complexity and rapid change together in a consistent manner.[15]

FINAL THOUGHTS AND ACTIONS

As I have described, the key elements of devops to drive collaboration between development and operations functions can be satisfied through implementation of additional process elements as part of the requirements gathering and design/build phases of the SDLC. Success also requires effective two-way communication between these groups and the business. IT organizations would be well advised to train both developers and operations in ITIL V3 processes and redesign the SDLC to ensure that runtime environment factors and operational considerations are addressed early (and often) in the lifecycle, beginning with requirements. In addition, successful organizations will build collaboration networks and consider the use of matrix teams of developers and operations staff, especially in release and deployment planning.

ENDNOTES

[1]Logue, Clyde. "Bridging the DevOps Gap" CM Crossroads Webcast Series, 29 July 2010 (www.youtube.com/watch?v=ZtewOtFKF4w).

[2]West, Dave. "It's Time to Take Agile to the Next Level." Forrester Research, 25 March 2011.

[3]*ITIL Service Transition*. ITIL V3. Office of Government Commerce (OGC), 2007. (ITIL published a "Summary of Updates" in August 2011; see www.itil-officialsite.com.)

[4]Chrissis, Mary Beth, Mike Konrad, and Sandra Shrum. *CMMI for Development: Guidelines for Process Integration and Product Improvement*. 3rd edition. Addison-Wesley Professional, 2011.

[5]Chrissis et al. See 4.

[6]"DevOps." Wikipedia (http://en.wikipedia.org/wiki/DevOps).

[7]Portelli, Bill. "DevOps 101." *CM Journal*, Vol. 9, No. 2, February 2011.

[8]*ITIL Service Design*. ITIL V3. Office of Government Commerce (OGC), 2007. (ITIL published a "Summary of Updates" in August 2011; see www.itil-officialsite.com.)

[9]Chrissis et al. See 4.

[10]*ITIL Service Transition*. See 3.

[11]*ITIL Service Operation*. ITIL V3. Office of Government Commerce (OGC), 2007. (ITIL published a "Summary of Updates" in August 2011; see www.itil-officialsite.com.)

[12]*ITIL Service Operation*. See 11.

[13]*ITIL Service Transition*. See 3.

[14]Forrester, Eileen, Brandon Buteau, and Sandra Shrum. *CMMI for Services*. Addison-Wesley Professional, 2010.

[15]West. See 2.

Bill Phifer is a Fellow at HP Enterprise Services, with responsibility for service assurance and service management strategies for Applications Management Services. He is an SEI-certified CMMI Lead Appraiser with over 35 years in IT, including 18 years in software process implementation and improvement, measurement, service management, and project management. Mr. Phifer is also a Lead Evaluator for Carnegie Mellon University's eSourcing Capability Model for Service Providers (eSCM-SP), with an interest in sourcing best practices. He is a regular presenter at IT industry conferences and seminars such as SEI's SEPG, *itSMF's* USA Fusion, *and those by IEEE.*

Mr. Phifer is currently involved in research and architecture of end-to-end IT lifecycle process integration between applications and infrastructure using multiple industry reference models and standards, such as CMMI, ITIL/ISO 20000, eSCM, ISO 9000, ISO 27001, and COBIT. He is a member of the Philadelphia Software Process Improvement Network, itSMF Lehigh Valley Local Interest Group, and a former president and current board member of the Delaware Valley Science Council. Mr. Phifer can be reached at bill.phifer@hp.com.

Devops: So You Say You Want a Revolution?

by Dominica DeGrandis

Coined in Belgium during a 2009 data center migration, the term "devops" sprang from an attempt to apply agile techniques to operations ("ops") activities. With ops on the receiving end of faster and more frequent deliveries from development teams, the pressure to keep up necessitated faster and more efficient practices. And now, with a cloud of thousands of servers to manage, the inefficiencies of the past would clearly no longer suffice.

The realization that the world of building, deploying, and maintaining environments could benefit significantly from using a new approach caught on rapidly. The devops momentum accelerated when professionals working in the bowels of software delivery found community support.

The "revolution in the making" is a shift from a focus on separate departments working independently to an organization-wide collaboration — a "systems thinking" approach. It's about addressing all the work as a whole, versus looking only at the bits and pieces. It's about work flowing across functions versus lurking in silos.

As part of a systems thinking approach, devops is about respect, cooperation, and trust among the individuals who do the work versus management-driven control. A systems thinking approach requires a change in behavior from management. In this article, I will explain why this change is valuable and describe the kind of leadership required to make the shift to follow the revolution. I will also discuss how using statistical process control as a mechanism for better decision making can help devops teams drive out variability in their processes and improve customer satisfaction.

THE PROBLEM: LITTLE-PICTURE THINKING

When an individual team within a company focuses only on optimizing its own area, team members are often oblivious to what's needed for the company as a whole to be successful. Each team member doing a terrific job in his or her own cubicle doesn't cut it when their work has to be handled, or mishandled, by someone else before it moves on to its ultimate use.

Consider a not-so-hypothetical example. A developer (let's call him Steve) proclaims his work is done when his code is checked into source control. In this instance, however, it turns out that his code, which worked just fine in the development and test environment, doesn't work in the staging environment.

Invariably, the problem wasn't discovered until the day before a release, and Steve (who has already begun work on another project) looks up to find the project manager, plus one or two others, gathered around his desk. Steve's neighbors become all ears as the scene plays out something like this: "The build isn't working in staging, Steve, and the error log indicates a missing file." To which Steve replies, "It works in dev. Doesn't staging have the latest version of …?"

The huddle outside Steve's cubicle grows until it's determined that changes must be made to allow for backward compatibility and that another build must be deployed to staging. QA and ops will have to work late (again) to test the new build and stage it for production before the release *tomorrow*. This is a problem, and an even bigger problem, long term, is that this kind of waste spawns disgust from professionals looking to take pride in their work. Ops is disgusted with the development team for the lack of backward compatibility in the code. The development team is disgusted with ops for the lack of consistency between the dev, test, and staging environments. QA is disgusted with both developers *and* ops for not considering these problems to begin with. And the project manager is disgusted, claiming developers and ops don't talk to each other and that "herding cats" or "babysitting" might be better terms for describing his or her role. This all eats away at team morale.

THE SOLUTION: SYSTEMS THINKING

Imagine a company very much like Steve's — we'll call it "WeNeedHelp.com" — and then further imagine that its heads of software engineering and IT operations agree to try a different approach, pursuing incremental evolutionary change (driven by customer demand) across their

departments. Analyzing customer demand across functional boundaries would enable teams to step back and objectively look at their productivity from the "outside in." They would fully describe where their work originates, the types of work, the arrival rate, and the expectations from upstream and downstream customers.

Problems that are hurting the business, but were previously ignored, would surface. If WeNeedHelp.com customers have been complaining that sustainment changes (noncritical changes to production) aren't being delivered, a demand analysis would make visible the fact that sustainment work is not getting done and that WeNeedHelp.com has no adequate "service delivery" capability for sustainment work. Perhaps it's due to a perceived higher value on new work from another business unit. Or perhaps there is no real avenue for sustainment or maintenance fixes to be worked (other than employees burning the midnight oil). Shining a light on the services that teams provide to each other and the blockages preventing them from doing so would create an incentive to use a service delivery approach to improving productivity and customer satisfaction.

Looking at sources of internal team dissatisfaction, WeNeedHelp.com might discover that ops is being supplied with confusing and inadequate information that makes it hard for them to do their job. And it may find that adopting explicit policies that define sufficient handoff information between teams solves that problem.

Considering demand would reveal variability that randomizes the process and prevents work from being delivered on time. Perhaps the codeline branching strategy used at WeNeedHelp.com has crippled the ability to merge certain changes until other changes are also ready for delivery. Using a systems thinking approach, it would become acceptable to experiment with new codeline strategies (as long as they were survivable).

That would be just the start. The team would then gather data to find ways to understand the capability of the system and how it operates. This data would expose risks, bottlenecks (or not), and economic overheads. It would set expectations for objective, data-driven management. As WeNeedHelp.com teams watch metrics trend, they could make well-informed, high-quality decisions.

Maybe the rate at which developers are fixing bugs is trending faster than the rate that ops can deliver the fixes to production. It may at first appear that ops is the bottleneck, when in reality the problem lies with a tightly coupled, complex system architecture that requires nothing less than a miracle to maintain. A systems thinking approach will show that problems are typically inherent in the system and not the people. To deliver business value early and often, WeNeedHelp.com will likely need to invest in an automated regression tool rather than continue to overload its testers.

As the WeNeedHelp.com dev and ops teams organizationally focus on common goals, they would seek out and receive feedback to keep aligned. They would work closely together and perhaps merge into one devops team. The devops principle of "optimize the whole" would spread through organization-level continuous improvement.

The ability to deploy quickly and correctly begins with involving *from the very beginning* those responsible for deployment.

THE BENEFITS OF SHIFTING TO A DEVOPS APPROACH

Optimizing the whole allows companies to achieve their goals of increasing revenue, driving down costs, and keeping good people. Let's look more closely at how devops can get us there.

Increasing Revenue

Increasing revenue requires releasing our product or service to market faster, which requires ever faster, evermore reliable software build and deployment methods. This begins with a thorough understanding of the interdependencies among build artifacts, database schemas, environment configurations, and the infrastructure they sit on.

Acquiring the knowledge needed to interact with and design a speedy and accurate deployment process takes time — time from a team that understands both software architecture and infrastructure. Dividing this work among traditional functional teams often leads to inadequate system design decisions that are discovered too late.

By merging smaller teams into larger teams focused on common goals, devops facilitates a broader understanding of the idiosyncrasies throughout the system and minimizes handoffs from team to team. By working together, early on, devops teams can foresee problems and design a well-thought-through deployment process. The ability to deploy quickly and correctly begins with involving *from the very beginning* those responsible for deployment.

Driving Down Costs

Driving down costs by reducing rework requires building quality into the process in the first place. Because they perform builds and deployments early and often, far away from the intensity of release night, devops teams using a systems thinking approach have time to fine-tune scripts, discover and solve problems, and cross-train team members. Imagine the increase in quality when staff has the capacity to work on improvements instead of chiefly interruptions and rework. Imagine releases evolving from chaotic middle-of-the-night ordeals into daily non-events.

With developers scoring points for making changes happen and ops guys scoring points for keeping things stable, is it any wonder they find working together for the good of the company difficult?

Wealthfront, an investment advisor whose founder was named one of the "Best Young Tech Entrepreneurs 2011" by *Bloomberg Businessweek*,[1] has just *one* team of engineers that collectively owns and is responsible for *all* processes involving development, testing, deployment, and monitoring. The entire team owns quality. They can deploy to production in less than 10 minutes, and they do so, on average, 30 times a day! In an SEC-regulated industry, their full regression suite runs in less than five minutes.[2] *Fast Company* included Wealthfront among its "10 Most Innovative Companies in Finance," noting that the Wealthfront website "has attracted more than $100 million in assets, and its managers, which undergo an intensive selection process, have collectively outperformed the S&P 500 by 6%."[3]

Keeping Good People

More than once I've seen a team of highly trained professionals crumble because of ongoing systemic frustrations that paralyze the team's effectiveness and pulverize its gung ho spirit. Management will know that this is happening when it discovers team members are moving on, perhaps to the competition — and invariably it's the best who are the first to go.

Keeping good people, by enabling them to take pride in their work, depends on the opportunity to master one's work. More than enormous sums of money, the ability to conquer the job is a huge motivator. When 32-year-old point guard Mike Bibby gave up US $6 million to join the Miami Heat late in his career, it was because he wanted a chance to play for the championship.[4] He wanted to belong to a really good team.

We all want to belong to a really good team, but this requires us to be really good at what we do. A devops team using a systems thinking approach allows for continuous improvement of critical skills that results in increased expertise. As at WeNeedHelp.com, this does not occur because of any grand project plan. It just happens. By implementing these changes in an incremental, evolutionary fashion, solutions will be found that no one would have earlier been able to plan for.

LEADERSHIP: THE ESSENTIAL INGREDIENT OF DEVOPS

This all sounds good, yet the devops revolution can't happen without good leadership — and good leadership often seems to be in woefully short supply. Some managers appear incompetent in part due to the structure of the corporation, with CEO fiduciary responsibility focused solely on short-term shareholder profit. Some management is slanted toward new projects (instead of much-needed maintenance work) because new projects can be written off as capital expenditures. Some management breeds contention between teams with annual merit ratings favoring certain teams over others. With developers scoring points for making changes happen and ops guys scoring points for keeping things stable, is it any wonder they find working together for the good of the company difficult? Good teamwork may help the company, but it provides few tangible points on an individual's annual merit review.

If you think I'm overstating the case, consider that Amazon once celebrated heartily after a software release crashed its website. Responsible for website availability at the time, Jesse Robbins (now CEO of Opscode) voiced a concern that the release in question would bring the site down. But an Amazon VP pushed forward anyway, believing the company's stock price would go up *regardless* of website availability. The software was deployed and — as Robbins predicted — brought the site down. It took two days to stabilize the site and get it back up, but as the VP had foreseen, the stock price rose along with order volume. At the end of that year, Amazon rewarded developers for the new and profitable functionality *and* penalized ops for the outage![5]

Can cross-teamwork find a spot on merit reviews? If not, maybe individual merit reviews should be abolished. After all, the benefits of dumping merit ratings have been discussed for over 50 years, beginning with W. Edwards Deming in his 14 principles for transformation.[6]

Another ineffective management technique that does more harm than good to the devops movement is pressuring staff into working during the wee hours of the night on top of their regular day job. Working long hours may come with trendy bragging rights these days, implying strength and power. But as *Wall Street Journal* health writer Melinda Beck says, "Genuine 'short sleepers' only need four hours of sleep per night, but they comprise only 1% to 3% of the total population."[7] So for the 97%-99% of us who aren't short sleepers, working the wee hours brings sleep deprivation and mistakes.

When production breaks in the middle of the night, it's typically someone from operations who gets paged to troubleshoot the issue. In problematic systems, ops can get paged a lot, sometimes working night after night (supporting an unstable system) on top of their day job. In teams that practice devops — where ops personnel, architects, and developers collaborate on system architecture *and* where developers carry duty pagers as often as ops staff do — production failures will likely diminish, reducing middle-of-the-night interruptions.

Management by Fear Breeds Distrust; Good Leadership Drives Out Fear

It's hard to put forth your best effort when you are worried about public humiliation or losing your job. Remarks like "Well, if we don't get this project delivered by October, we'll all be looking for new jobs" or "There are a lot of people out there who would *love* to have your job" result in people keeping ideas to themselves. Collaboration, not withholding information, is what's needed in the devops revolution.

I once worked with a team where the director announced to her staff, "Do not send any communications out to other teams without my knowledge." To me this screamed, "I don't trust you to do the right thing," and it resulted in a pivotal moment when more than one résumé got updated in preparation for departure.

How do you know if people are fearful? Look at reports — inflated figures are a sure signal that people are telling you what you want to hear instead of the truth you need to know. Sifting through a wordy progress report crafted by someone who has carefully chosen words so as not to offend is time-consuming and rarely tells you what is really going on.

Making the shift to a devops systems thinking approach requires trust. Trust is essential for influencing change, and gaining trust takes time. Sociologists have learned that trust is event-driven and that small, frequent gestures or events enhance trust more than larger, grand gestures made only occasionally. Trust is also asymmetrical in that a single act of bad faith can destroy it, and repairing the damage, if possible at all, will take many small acts completed with competence and delivered as promised.

For some, moving to a systems thinking approach will require a revolution in management behavior. This revolution consists of:

- An emphasis on quality, which encourages craftsmanship and pride of workmanship
- Establishing avenues to enable collaborative working
- A properly designed process, which enables people to work effectively without becoming overloaded
- A feeling among workers that their actions are directly connected to business success (or failure)
- Management that addresses systemic problems by changing processes and policies, not by blaming or replacing people
- Management that demonstrates trust in the workers, and those same workers learning to trust their managers (via the quality of management actions)
- Management that drives out fear to increase productivity and innovation

These are management imperatives that are essential to making the shift to devops.

IMPROVING THE QUALITY OF DECISION MAKING

While effective leadership is absolutely essential to the devops revolution, it is not sufficient. High-quality decision making is the second half of the equation.

One means of improving decision making is the SPD (Study-Plan-Do) model, which is used for carrying out continuous improvement. It was inspired by W. Edwards Deming's popular PDSA (Plan-Do-Study-Act) cycle[8] and John Seddon's three-step approach for managing change, CPD (Check-Plan-Do).[9] The SPD model teaches that first you must Study. Studying the "what and why" of the current performance of a system leads to understanding possible improvements and what prevents them from being achieved. Next is "Plan," or identifying what needs to change in order to improve performance and what steps can be taken to get predictable results. This is followed by "Do," or executing the plan by taking small steps in controlled circumstances. Both Deming and Seddon stress that we must study capability and seek

to match capability against demand in order to provide satisfactory service.[10, 11] Statistical process control (SPC) charts are a mechanism for the Study part of the Deming cycle and Seddon CPD (see sidebar).

Removing Variability Results in Greater Predictability

SPC charts can be applied to devops where predictability of "routine" work, such as system maintenance and deployments, has a direct connection to the organization's immediate profitability. SPC charts provide a means for understanding the capability of a team or organization and the variation in their tasks and deliverable work items.

To be meaningful and useful, an SPC chart should be created for work items of variable size processed by a team or department. For example, measuring cycle time across a set of typical ops tasks doesn't tell us much when they are of widely varying sizes. Changing user permissions might take only two minutes, but upgrading databases to the next version might take weeks. Combining cycle times for these would skew results, making it difficult to put any predictability into the process. Breaking out the SPC chart by work item and size — so that, for example, database upgrades all appear on the same chart — helps avoid convoluted metrics. Exceptionally long cycle times for database upgrades would then stand out as special causes and would act as talking points for retrospectives and ops reviews.

By studying and then removing variability in the system, we enable a devops team to become more predictable. SPC charts help us to see variability that we want to act upon as managers. They help us demonstrate that capability is matched with demand and expectations. Using SPC charts as part of the SPD cycle inevitably leads us to improved customer satisfaction.

SPC CHARTS

Statistical process control charts (see Figure A) involve plotting events to reveal which ones fall outside of acceptable levels. The levels are demarcated with an upper control limit (UCL) and a lower control limit (LCL). The points that fall outside of the control limits are called "special" or "assignable cause" variation. These tend to be external to the system or process. The term "assignable" means that they can be easily identified and are often pointed to by members of the team. A list of special cause variations should be addressed with risk management — mitigation, reduction, and contingency planning. The variation among the points within the control limits (as in Figure A) is known as "common" or "chance cause" variation. This tends to be internal to the system and is therefore, in theory, affected by the process in use and under the control of local managers. Common cause variation can be affected by policy changes, individual skills, process design, training, recruitment policies, and many more aspects of the workplace that are under management control.

The lesson here is that the control chart can be used to see capability and to react when it is poorly matched with expectations. Control charts show where the process is unstable and hence totally unpredictable.

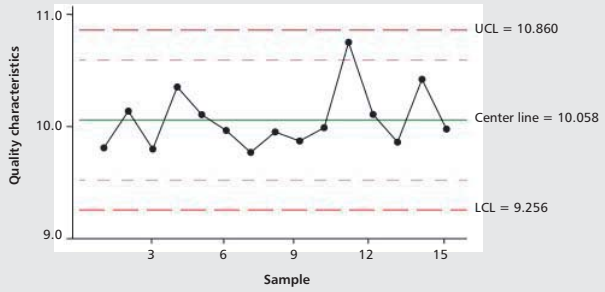

Figure A — Control chart showing common cause variation.

PRECISION AND QUALITY DRIVE DOWN COST

It sounds counterintuitive, but a focus on cutting costs actually results in higher costs. We've seen evidence of this with companies that relied heavily on manual testing.

When I worked at Corbis (a privately held, Seattle-based company that licenses digital images), we relied entirely on manual build tests during deployment *for years* — the argument being that automated test tools were too expensive. The reality was that each build required, on average, 25 minutes of manual testing before being released to QA for integration testing. Full regression testing rarely occurred because it took too long. This added rework when previous production fixes were overwritten with newer builds. A contributing factor to deployment issues was the large batch size of the changes incorporated into new project builds, which created too much work-in-progress (WIP).

With a complex system prone to design and code issues, a new build sometimes ate up days of troubleshooting by database developers, architects, sys admins, build engineers, and testers. Imagine the cost of that!

The risks involved with the lack of automated testing eventually diminished over time, due in part to the following improvements:

- Continuous integration (although a major cost) was implemented, allowing developers to immediately see the impact of their code changes and fix problems on the spot in the development environment.
- WIP was limited to available capacity, resulting in less context switching.
- The release cadence increased from quarterly to biweekly. This resulted in fewer changes (again, less WIP) per build, allowing each change to be tested more thoroughly.
- Smaller, more frequent changes reduced merge and integration issues.

The improvements listed above were not made as part of a project plan or an IT governance initiative, but as part of an ongoing management focus on driving incremental improvements continuously as a regular part of operating our business. Organizationally, we were focused on delivering business value. Teams were aligned around business goals and individuals were empowered with information on business risk and value, supported by a management team that encouraged them to *optimize the whole* system through collaboration with peers rather than act locally and independently. This was a significant leap from the previous management approach, and it demonstrated the powerful concepts behind the devops movement.

SUMMING UP

A systems thinking approach to devops results in the ability to increase revenue, drive down cost, and keep good people. These valuable results are achieved by building quality into devops activities *from the beginning* and establishing proper motivations to drive out resentment and fear, enabling continuous improvement and increased expertise. These attributes, indeed, make for a devops revolution.

In addition, managers can use the Study-Plan-Do model to balance the capability of their organization against the demand. They can improve the quality of their decisions by understanding variability through SPC charts. They can take actions to change the system (the policies and procedures and practices used) or to deal appropriately with special variations through risk management.

This is how IT organizations should be run. Better managers are systems thinkers who make better decisions and produce better results.

ENDNOTES

[1]"Best Young Tech Entrepreneurs 2011." *Bloomberg Businessweek*, 17 May 2011 (http://images.businessweek.com/slideshows/20110516/best-young-tech-entrepreneurs-2011/slides/4).

[2]Perez, Pascal-Louis. "Continuous Deployment in an SEC-Regulated Environment – SLL Conf." Wealthfront Engineering, 25 May 2011.

[3]Macsai, Dan. "The 10 Most Innovative Companies in Finance." *Fast Company*, 14 March 2011 (www.fastcompany.com/1738549/the-10-most-innovative-companies-in-finance).

[4]Salter, Chuck. "The World's Greatest Chemistry Experiment." *Fast Company*, No. 155, May 2011, pp. 78-84.

[5]Logan, Martin J. "DevOps Culture Hacks." DevOps.com, 8 March 2011 (http://devops.com/2011/03/08/devops-culture-hacks).

[6]Deming, W. Edwards. *The New Economics for Industry, Government, Education*. 2nd edition. The MIT Press, 2000.

[7]Beck, Melinda. "The Sleepless Elite." *Wall Street Journal*, 5 April 2011.

[8]Deming. See 6.

[9]Seddon, John. *Freedom From Command and Control: Rethinking Management for Lean Service*. Productivity Press, 2005.

[10]Deming. See 6.

[11]Seddon. See 9.

Dominica DeGrandis is an Associate with David J. Anderson & Associates, specializing in Kanban for IT operations and devops. Ms. DeGrandis spent her first 15 years in software engineering deeply embedded in development teams performing builds, deployments, and environment maintenance. Adept at leading teams performing configuration management and release management, she found a passion for improving the way development and operations teams work together. Committed to doing the right thing, Ms. DeGrandis studies sustainability in business, community, and global social matters. She is based in Seattle, Washington, USA, and holds a BS in information computer sciences from the University of Hawaii. Ms. DeGrandis can be reached at dominica@djandersonassociates.com.

Cutter Membership
The Ultimate Access to the Experts

Cutter Consortium Membership opens up multiple avenues to interact with Cutter's experts to brainstorm and gain guidance to transform your organization and boost success.

Like everything business technology, one size does not fit all.

That's why we encourage you to choose the Membership that's right for your organization. Whether you choose Digital Transformation & Innovation, Enterprise-wide, Practice-specific, or CIO Membership you'll see a strong return.

Contact us at +1 781 648 8700 or sales@cutter.com to arrange a sample inquiry call with a Cutter expert and see for yourself how quickly your return on Membership can be realized.

What Do You Get from Cutter Membership?

- Get guidance in leveraging new strategies, emerging technologies, and business management practices to enable digital transformation and boost competitive advantage
- Learn how to mine data to create new products and services and improve customer experience
- Get input on how to reduce expenses through more cost-effective strategies
- Gain insights and get ideas on achieving sustainable innovation, successful change management, and prudent risk management
- Get coaching and insights on leadership and team-building practices that boost productivity
- Discover vendor-agnostic advice to ensure unbiased purchasing decisions

How Does Cutter Membership Deliver Benefits?

- Get answers to your specific questions by taking advantage of **inquiry privileges.**
- Ensure your organization is on the right path with **regular strategy meetings for your team with a Cutter expert.**
- Move your entire organization up the learning curve with "seat for everyone" **unlimited access to Cutter's research.**
- Discuss cutting-edge strategies with your peers and Cutter's experts during **virtual roundtables** and **online peer-to-peer events.**
- Immerse yourself in new ideas and bounce concerns off Cutter's experts with **a complimentary seat at the Cutter Summit.** Train your entire team: take advantage of **additional discounted seats.**
- Get guidance in digitally transforming your business, learn new leadership skills, navigate organizational change, and more with **add-on consulting, training, and exec ed.**

CUTTER CONSORTIUM
Access to the Experts

About Cutter Consortium

Cutter Consortium is a truly unique IT advisory firm, comprising a group of more than 100 internationally recognized experts who have come together to offer content, consulting, and training to our clients. These experts are committed to delivering top-level, critical, and objective advice. They have done, and are doing, groundbreaking work in organizations worldwide, helping companies deal with issues in the core areas of software development and agile project management, enterprise architecture, business technology trends and strategies, enterprise risk management, metrics, and sourcing.

Cutter offers a different value proposition than other IT research firms: We give you Access to the Experts. You get practitioners' points of view, derived from hands-on experience with the same critical issues you are facing, not the perspective of a desk-bound analyst who can only make predictions and observations on what's happening in the marketplace. With Cutter Consortium, you get the best practices and lessons learned from the world's leading experts, experts who are implementing these techniques at companies like yours right now.

Cutter's clients are able to tap into its expertise in a variety of formats, including content via online advisory services and journals, mentoring, workshops, training, and consulting. And by customizing our information products and training/consulting services, you get the solutions you need, while staying within your budget.

Cutter Consortium's philosophy is that there is no single right solution for all enterprises, or all departments within one enterprise, or even all projects within a department. Cutter believes that the complexity of the business technology issues confronting corporations today demands multiple detailed perspectives from which a company can view its opportunities and risks in order to make the right strategic and tactical decisions. The simplistic pronouncements other analyst firms make do not take into account the unique situation of each organization. This is another reason to present the several sides to each issue: to enable clients to determine the course of action that best fits their unique situation.

For more information, contact Cutter Consortium at +1 781 648 8700 or sales@cutter.com.

The Cutter Business Technology Council

The Cutter Business Technology Council was established by Cutter Consortium to help spot emerging trends in IT, digital technology, and the marketplace. Its members are IT specialists whose ideas have become important building blocks of today's wide-band, digitally connected, global economy. This brain trust includes:

- Rob Austin
- Ron Blitstein
- Christine Davis
- Tom DeMarco
- Lynne Ellyn
- Israel Gat
- Tim Lister
- Lou Mazzucchelli
- Ken Orr
- Robert D. Scott