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

 ©2011 Cutter Information LLC

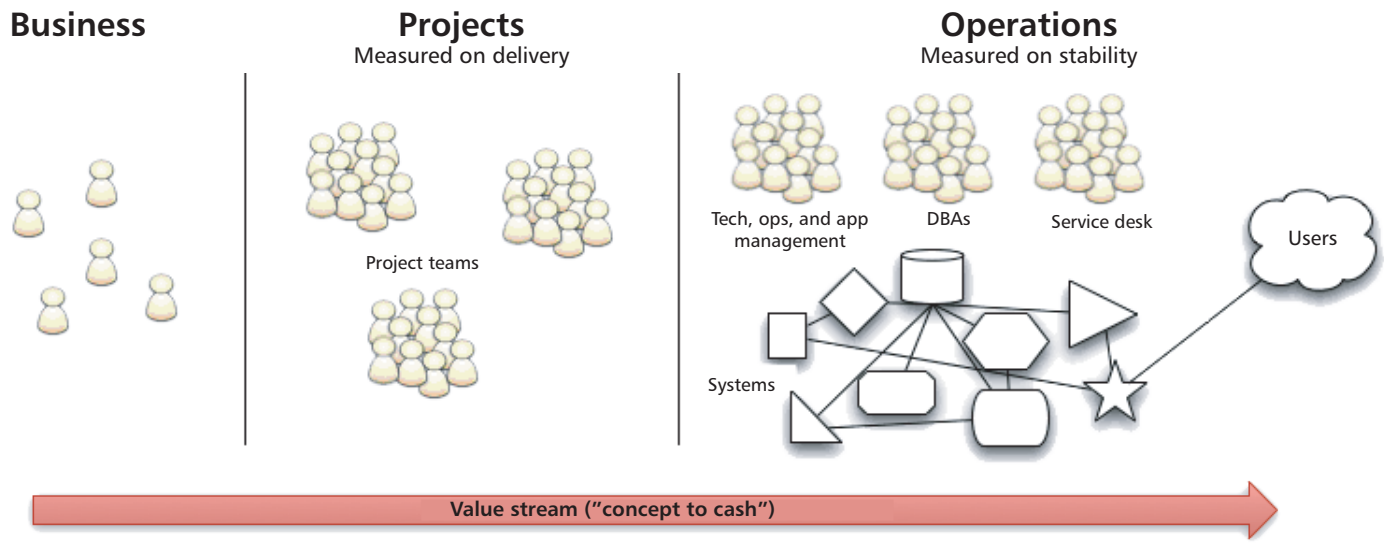

Figure 1 — The flow of value in project-based organizations.

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.

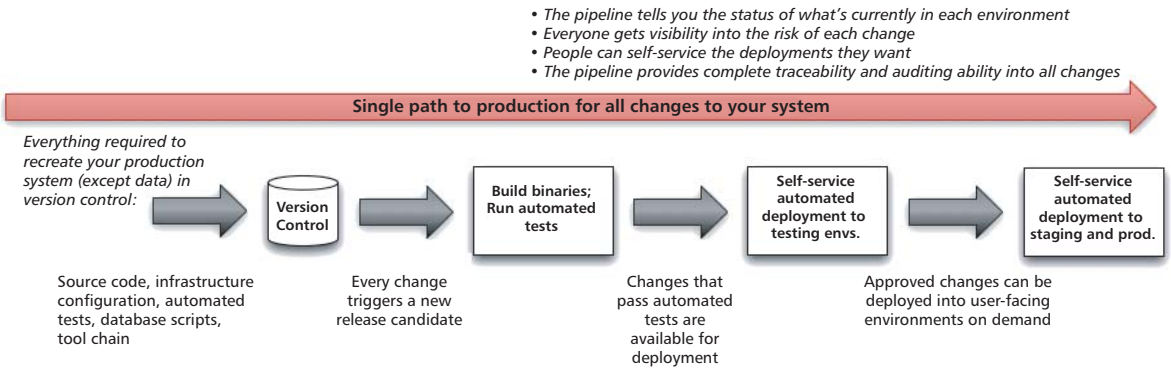

Figure 2 — The deployment pipeline.

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.

©2011 Cutter Information LLC

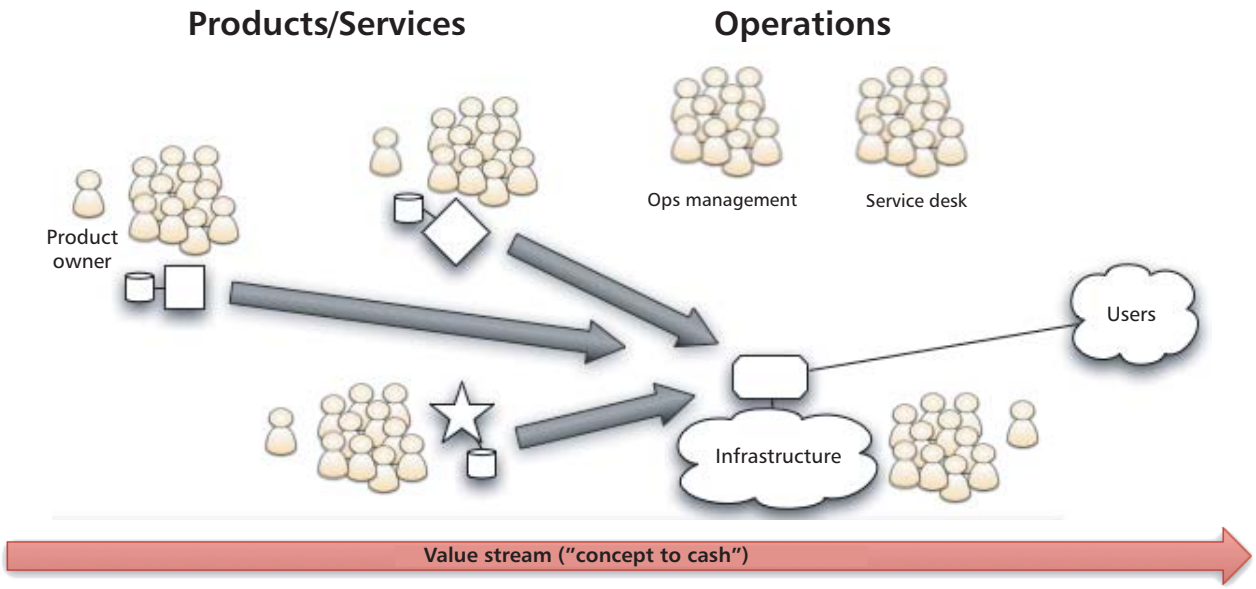

Figure 3 — The flow of value in a product development model.

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]

©2011 Cutter Information LLC

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.

©2011 Cutter Information LLC