Cutter IT Journal

The Journal of Information Technology Management

Vol. 24, No. 12
December 2011

"There is nothing to suggest that devops can't work in an enterprise. The main challenge will be reevaluating the existing people and processes from a constant collaboration viewpoint, keeping the business goals in mind."

— Patrick Debois, Guest Editor

Embedding Devops in the Enterprise

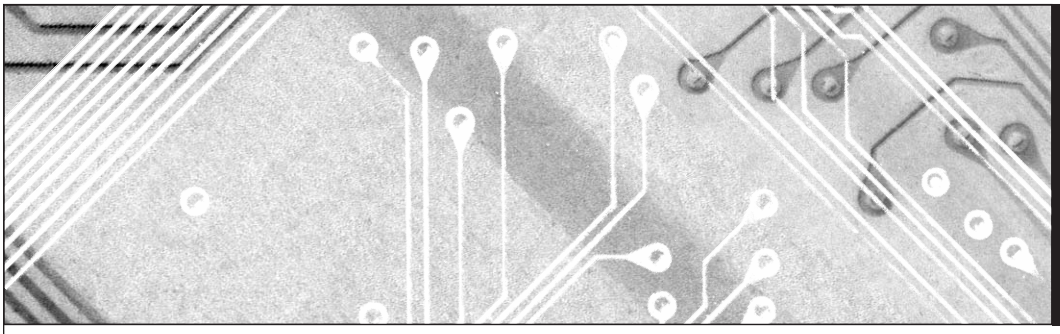
Big Enterprises

Cross-silo communication is harder in large enterprises, but they're the ones that need it the most.

Small Changes

Bad ITIL implementations often resist change because of risks, but making frequent, small changes can reduce the risks.

CUTTER CONSORTIUM



Cutter IT Journal

About Cutter IT Journal

Part of Cutter Consortium's mission is to foster debate and dialogue on the business technology issues challenging enterprises today, helping organizations leverage IT for competitive advantage and business success. Cutter's philosophy is that most of the issues that managers face are complex enough to merit examination that goes beyond simple pronouncements. Founded in 1987 as *American Programmer* by Ed Yourdon, *Cutter IT Journal* is one of Cutter's key venues for debate.

The monthly *Cutter IT Journal* and its companion *Cutter IT Advisor* offer a variety of perspectives on the issues you're dealing with today. Armed with opinion, data, and advice, you'll be able to make the best decisions, employ the best practices, and choose the right strategies for your organization.

Unlike academic journals, *Cutter IT Journal* doesn't water down or delay its coverage of timely issues with lengthy peer reviews. Each month, our expert Guest Editor delivers articles by internationally known IT practitioners that include case studies, research findings, and experience-based opinion on the IT topics enterprises face today — not issues you were dealing with six months ago, or those that are so esoteric you might not ever need to learn from others' experiences. No other journal brings together so many cutting-edge thinkers or lets them speak so bluntly.

Cutter IT Journal subscribers consider the *Journal* a "consultancy in print" and liken each month's issue to the impassioned debates they participate in at the end of a day at a conference.

Every facet of IT — application integration, security, portfolio management, and testing, to name a few — plays a role in the success or failure of your organization's IT efforts. Only *Cutter IT Journal* and *Cutter IT Advisor* deliver a comprehensive treatment of these critical issues and help you make informed decisions about the strategies that can improve IT's performance.

Cutter IT Journal is unique in that it is written by IT professionals — people like you who face the same challenges and are under the same pressures to get the job done. *Cutter IT Journal* brings you frank, honest accounts of what works, what doesn't, and why.

Put your IT concerns in a business context. Discover the best ways to pitch new ideas to executive management. Ensure the success of your IT organization in an economy that encourages outsourcing and intense international competition. Avoid the common pitfalls and work smarter while under tighter constraints. You'll learn how to do all this and more when you subscribe to *Cutter IT Journal*.

Cutter IT Journal®

Cutter Business Technology Council: Rob Austin, Ron Blitstein, Christine Davis, Tom DeMarco, Lynne Ellyn, Israel Gat, Tim Lister, Lou Mazzucchelli, Ken Orr, and Robert D. Scott

Editor Emeritus: Ed Yourdon
Publisher: Karen Fine Coburn
Group Publisher: Chris Generali
Managing Editor: Karen Pasley
Production Editor: Linda M. Dias
Client Services: service@cutter.com

Cutter IT Journal® is published 12 times a year by Cutter Information LLC, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA (Tel: +1 781 648 8700; Fax: +1 781 648 8707; Email: citjeditorial@cutter.com; Website: www.cutter.com; Twitter: @cuttertweets; Facebook: Cutter Consortium). Print ISSN: 1522-7383; online/electronic ISSN: 1554-5946.

Subscription rates are US $485 a year in North America, US $585 elsewhere, payable to Cutter Information LLC. Reprints, bulk purchases, past issues, and multiple subscription and site license rates are available on request.

Opening Statement

by Patrick Debois, Guest Editor

EMBEDDING DEVOPS IN THE ENTERPRISE

With the term "devops" picking up steam, vendors are now (re)branding their tools as devops tools. Similar to unit-test tools that supported an agile workflow, the current discussion on deployment automation supports the devops ideas. Even though tools have their merits,[1] after reading the August 2011 issue of *Cutter IT Journal* — "Devops: A Software Revolution in the Making" — it should be clear that tools are merely one aspect of devops and must be complemented with other aspects. The nice thing about tools is that they give you something concrete to discuss, as compared to the more intangible notion of "culture." Within large enterprises, tools are probably the easy part. Therefore, in this issue, we would like to focus on the harder aspects, like "people and processes," or as the Agile Manifesto puts it, "Individuals and interactions."

As agile practitioners Ken Howard and Barry Rogers have observed, "If your team has dug itself into a hole, the process ain't gonna pick up the shovel."[2] It all starts with great people. Getting people to collaborate is pretty easy in small, startup-like organizations, but it's a whole different ball game in an enterprise setting. Ernest Mueller kicks off this issue by looking beyond technical skills and gives sane advice on how to pick individuals and form teams in order to improve devops collaboration. You should not select just the right mix of cross-functional technical skills, but more important, you should screen for people's willingness to collaborate. Managers, you need to support this collaboration beyond the traditional handover point between development and operations. It needs to happen during the entire lifecycle — from the moment you start thinking about new ideas all the way to running the implemented idea.

Once this team starts picking up steam, it will most certainly hit boundaries in your existing enterprise processes. The agile culture of small and frequent changes seems to be orthogonal to the culture of the Information Technology Infrastructure Library (ITIL). As Cutter Senior Consultant Bill Keyworth correctly points out in the second article, many ITIL implementations have focused on the process for IT operations' sake. This is the point that usually causes the friction, and working with a devops mindset is likely to amplify it. To address this, version 3 of ITIL's set of best practices introduced the concept of the Application Management Lifecycle: acknowledging that interaction between dev and ops needs to be permanent. Instead of dismissing ITIL as a bureaucratic approach, learn about its real intent and how it complements devops collaboration.

Getting people to collaborate is pretty easy in small, startup-like organizations, but it's a whole different ball game in an enterprise setting.

On the other side of the traditional dev/ops fence, agile process methodologies like Scrum and Kanban also need to move out of their comfort zones. Seasoned agilist Scott Ambler points out that agile needs to become more enterprise aware and discusses an approach he calls "Disciplined Agile Delivery." Elements such as release and deployment should be integral parts of your agile vision and your daily activities. Keep a focus on the real business requirements not merely by developing software, but by providing a complete solution that can run stably in production. Ambler mirrors Mueller's and Keyworth's thoughts on the importance of addressing both functional and nonfunctional requirements throughout the complete workflow: from inception through construction and transition.

Constant collaboration with good governance is key. To support this, you need metrics and facts. Ambler rightly points out another important aspect besides people, process, and tools: data integration. Large amounts of useful information remain locked up within silos. Think

about log files, monitoring information, and configuration. By applying the concept of open data within your company, internal information should be freely available to all people engaged in the complete lifecycle.

Part of the "secret sauce"[3] for listening to customer feedback is monitoring and collecting metrics. Alexis Lê-Quôc expresses this need in his article "Metrics-Driven Devops." This is another of the great boundary objects between development and operations. As Lê-Quôc points out, you need to take away the friction of collecting metrics. Having the correct APIs allows you to automate part of the process. Friction discussions — such as "Why did it crash?" or "How many customers use this feature? — become less emotional. It puts the engineering back into IT. By identifying the "actionable" metrics — those between lower-level technical metrics like memory and CPU usage and high-level business metrics like page views and conversion rates — you create a framework where people from their unique enterprise perspectives can keep relating to the business goals.

The previous advice might sound great, but it might also sound intimidating. Shifting an enterprise into a new direction is like turning an oil tanker — it takes a while to turn, but once on the right course, it can be very powerful. Concluding this issue, Kief Morris provides a case study that shows us how to go in small steps. Start by introducing the minimum viable approach to get the collaboration flow going between business, development, QA, and operations. This will reduce the cost of change and risks. But please don't stop there: use the feedback channel at the end of the chain to improve your understanding of the process and make improvements accordingly. A powerful positive loop will result, creating a win-win situation for development, operations, QA, and — not least — the customer.

To conclude, I would say that there is nothing to suggest that devops can't work in an enterprise. It can coexist with both agile processes and ITIL. But like any other change, it will take time and requires you to take the existing ecosystem of process and people into account. The main challenge will be reevaluating the existing people and processes from a constant collaboration viewpoint, keeping the business goals in mind. If people question collaboration, don't measure collaboration for its own sake — look at the actual results and facts. And if it doesn't work, learn and improve.

ENDNOTES

[1]Debois, Patrick. "Devops, Tools, Fools, and Other Smart Things." Presented at *GOTO Aarhus Conference 2011*, Aarhus, Denmark, 10-12 October 2011(www.slideshare.net/jedi4ever/devops-tools-fools-and-other-smart-things).

[2]Ken Howard and Barry Rogers. "Beyond Process and Tools: People Issues in Agile Software." Interview by Matthew Heusser. informIT, 14 April 2011 (www.informit.com/articles/article.aspx?p=1701933).

[3]Allspaw, John, and Jesse Robbins. *Web Operations: Keeping the Data on Time*. O'Reilly Media, 2010.

Patrick Debois is a Cutter Consortium Senior Consultant for the Agile Product & Project Management *practice. In order 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, system 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 in 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.*

Executive Education +

SUMMIT 2012

Cutter Consortium

2–4 April 2012 | Cambridge, MA, USA

Immerse Yourself in Agile at Cutter Consortium's Summit: Executive Education+

Agile, the software method that was conceived as a way to cope with change, is itself dramatically changing. Immerse yourself in Agile at Cutter Consortium's Summit: Executive Education+ and find out how these changes can benefit your organization, and what you need to do to make that happen.

Agile Practice Director Israel Gat has assembled an impressive team of Cutter consultants to present, including: Patrick Debois, the "godfather of devops"; Lean software guru Jim Sutton; Agile software development lifecycle developer Hubert Smits; and Israel himself, the architect of the Agile transformation at BMC.

In addition to the stellar Agile program, you'll benefit from the world-class executive education the Summit is known for — keynotes, interactive case studies, and exercises on leadership and teaming delivered by Cutter Fellows and distinguished business school faculty, as well as interactive work sessions built around emerging topics and technologies that matter to business technology leaders. You'll enjoy (and join in on!) raucous panel debates; networking at lunches, breaks, and entertaining evening events; and get one-on-one guidance and input from expert presenters and participants.

What Leaders Need to Know About Devops
Patrick Debois, "the godfather of devops"

Agile 2.0: Change Is Changing!
Israel Gat, architect of Agile transformation at BMC

Want to be Radical? Here's How.
Hubert Smits, developer of Agile software development lifecycle

Reclaiming Business Glory Through the Lean Worldview
Jim Sutton, Lean software guru

Teaming in Action
Group exercises in teaming with Cutter Senior Consultant Alan MacCormack

Can You Achieve Operational Excellence?
Interactive work session with Bill Keyworth

Teaming: How Organizations Learn and Compete in the Knowledge Economy
Keynote by Amy C. Edmondson

Harder than I Thought: Adventures of a 21st-Century CEO
Keynote by Cutter Fellow Richard L. Nolan

CIO Roundtable
Forum with Cutter Fellow and Practice Director Ron Blitstein

Our best offers for *Summit 2012* end soon! BOGO: Buy 1 seat for $1995 and bring a colleague free (save $1995); single seat $1595 (save $400!)

See the full agenda at www.cutter.com/summit.html

Devops and the People Who Practice It:
Winning Their Hearts and Minds

by Ernest Mueller

> The ultimate victory will depend on the hearts and minds of the people who actually live out there.
>
> — US President Lyndon Baines Johnson

On the dev2ops blog (one of the primary locations for seminal devops thought), Alex Honor states his chosen methodology as "People over Process over Tools."[1]

This is of course a riff on the Agile Manifesto's value statement of "Individuals and interactions over processes and tools."[2] I believe that devops is at its heart an extension of agile as applied to include operations,[3] so this makes sense as an analogous principle. Why, then, is so much of the devops discussion about those lower-priority items and not the people (see Figure 1)?

What this risks is the emergence of pseudo-devops, much like pseudo-agile, where people mimic its lower-order practices without adopting its philosophy — and then wonder why they are not seeing the success the methodology promises. Devops success is not given to you by tools; it is generated by the people forming your product team and how they approach their work.

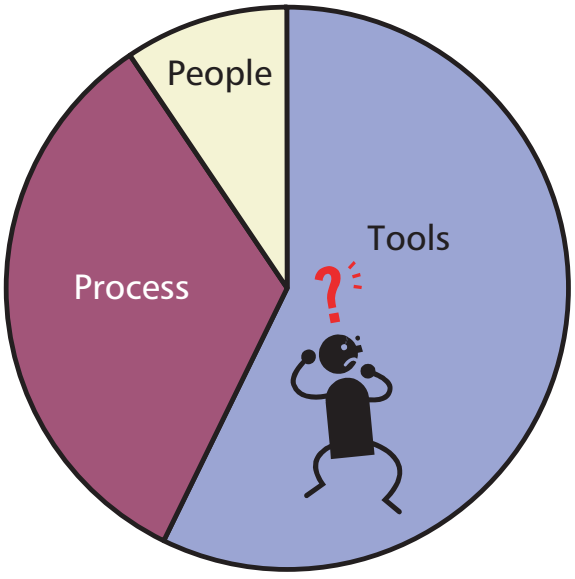


Figure 1 — Devops: all about the tools?

CULTURE: BRIDGING THE DIVIDE

John Willis coined the acronym CAMS (Culture, Automation, Measurement, Sharing) to describe what devops is about based on his research among successful practitioners.[4] And there are many cultural issues to face when implementing devops. Even in an optimal situation, organizational change produces confusion and stress and requires active management to turn out positively.

And usually you're not dealing with an optimal situation. The history of "dev" and "ops" — and, really, the history of ops itself (traditionally, IT infrastructure departments)[5] — is one of silos. Dev is siloed from ops, and ops is further siloed into technical specialties such as networking, Unix, Windows, database administration, and other separate subteams. James Turnbull calls this challenge "Bridging the Great Dev/Ops Divide,"[6] and it is a result of the otherwise admirable tendency toward specialization that empowered the Industrial Age. However, this specialization has created different cultures in these different teams,[7] and there is a great deal of (not entirely unjustified) polarizing stereotypes that the different groups have of each other, whether it's the "cowboy coder" or the "bastard operator from hell." What you need is to somehow win the hearts and minds of both groups.

I deliberately chose the phrase "hearts and minds" because of its double-edged connotation — in Vietnam, as we know, this famously stated policy was not a success. It is easy for management to blithely tell people they should get along together and collaborate, but when those people have different languages, goals, and responsibilities, it is likely to fall apart at the grass-roots level. How can you facilitate collaboration among the individuals carrying out the everyday work of creating and managing your services? Most of the advice I've heard on creating an atmosphere of collaboration in a devops environment has consisted of somewhat simplistic insights like "Have lunch together!" and "Beer makes everything better!" As a leader, you can't make

a culture change happen, but you can certainly facilitate it. Let's talk about some actionable paths to profound and lasting devops culture change.

CONSTRUCTION: BUILDING A TEAM

Getting the right people onto your team can be difficult. Depending on whether you are recruiting for a new team or crafting a team from current staff, there are things you can do to set yourself up for success.

If you are recruiting people from scratch, check out Joel Spolsky's *Smart & Gets Things Done*,[8] probably my favorite book on technical hiring. In it, he explains that passion, persistence, and fit are as important as technical chops in terms of real-world effectiveness, and that goes double for a collaborative devops environment. John Allspaw from Etsy says that collaboration is a job requirement on the operations team there and a trait for which they specifically interview.[9] If you're forming a new internal team, treat it like external hiring — interview and vet the people going onto the team. If they don't fit that team's culture, there will be trouble.

The last time I managed a team, I made sure that part of the interview process was a group interview with all members of the team. It's the best, quickest way to find out if there is good fit or not! We rated candidates on a 1-10 scale on six factors:

1. Technical skills
2. Communication
3. Initiative
4. Culture fit
5. Leadership
6. Problem solving/troubleshooting

At the bottom of the interview form, each interviewer was asked whether the team should "Decline" or "Accept" the candidate — no waffling. The decision was still up to the manager (in this case, me), but as a team we looked askance at any candidate that someone had declined. We didn't have collaboration and passion as specific line items — maybe we should have — but we used those characteristics to inform our judgment when rating on the six chosen factors. Even when we were pretty desperate to fill a spot, we passed on a fair number of candidates with seemingly good technical skills because we didn't feel that they were the kind of people who would work well with the team. We never regretted it — although we certainly regretted some of the times when we gave in and hired anyway.

You also want to get your team composition right. When we formed our current team, we did it from internal staff and decided we wanted to break the current practice of silos. We specifically formed an integrated team with developers and operations staff all in one, and we made sure that part of the development staff was allocated to automation to serve those operational needs. As we scaled, we urged other partner development teams to have their own integral ops expertise as well.

The more stable your systems environment, the higher the ratio of development staff to operations that is acceptable. But in a highly dynamic cloud environment, for example, you may end up going as low as a 3:1 dev to ops mix if the systems are a major moving part of the service. Furthermore, you may want to avoid relegating systems automation to the realm of "scripting," as if it's some afterthought that doesn't require sophistication or entail complexity; make sure you have dedicated, proper programming expertise working on it.

Passion, persistence, and fit are as important as technical chops in terms of real-world effectiveness, and that goes double for a collaborative devops environment.

On our new devops team, which is working on a cloud platform product, we ended up having three programmers, one systems automation programmer, and two systems engineers — a very ops-heavy mix. The old IT ratio of 20 (or more) developers to 1 operations staff member is one of the reasons operations teams become bottlenecks in software releases. Once you move past a certain ratio, there is no collaboration, just "transactions." If that is working out well for you, great, but in my experience the friction and loss of agility from not having enough time to truly engage with product teams leads to failure. On our teams that develop SaaS products, we expect an ops person to be on the product team — *really* on it, not just paying attention to it with 1/20th of his or her time.

Traditionally, organizations have operated on assumed ratios between operations staff and servers, such as "one sys admin per 20 servers" or similar arbitrary ratios between operations staff and users.[10] Ideally, with devops you are automating such that one ops person can handle many servers, and with modern multitenant services, you get thousands of users for every one person you need answering support tickets. That is

the wrong framing, however. You are not delivering *servers* to your customer; you are delivering a service, a product. You need to be thinking about those product teams and staffing them up to meet the need. Take a look at the activities in the ITIL Service Design stage,[11] which you should be performing during your development: Service-Level Management, Availability Management, and Capacity Management. Information security management. All these are areas that depend heavily on operations expertise. Consider your ops staff to be just a different kind of developer asset — one whose specialty is delivering infrastructure instead of Java code — and put as many on your project team as you need to deliver a quality service.

Much of the hostility between functional silos is based on legitimately competing goals.

How many is that exactly? Well, just as with "How many developers do I need?" this figure is completely dictated by the requirements of the product at hand. Make sure someone with operational expertise is included in the product formation stage, where you'll talk about level of effort and resourcing, so that you can understand the need. A very common mistake is thinking that ops doesn't need to be involved until much later in the development cycle. Then, when your assumption that some minimal work by existing staff is all that is needed comes crashing down around you, your project will be plagued by product delays.

COMMUNICATION: FINDING A COMMON LANGUAGE

Once you have a project team with collaboration-minded developers and operations staff, you have to overcome the fact that these separated tribes have developed their own languages over time. A good example of this syndrome, related to me by a colleague from an Austin, Texas–based startup, involved a Web application performance testing exercise leading up to a major release. After an abortive attempt to start doing performance testing, the company realized that the developers, QA personnel, and operations staff (and their tools) all expressed load in different ways — concurrent connections, user sessions, hits per second, and so on. They had to go through a careful mapping exercise to get to a common taxonomy around just that one topic in order to translate back and forth from their toolsets and be effective.

Besides general differences in technical terms, agile terminology and practice are much less common among operations professionals than among developers — and they're not completely pervasive among developers, either. The members of your team will need to spend time coming to an agreement on terminology and learning terminology that is unfamiliar to them. We spent a lot of time in our team during the "forming, storming, and norming" stages[12] of the project discussing definitions and recording them on a common wiki for future reference.

Common tooling can help provide a common language. I've seen many IT departments in which developers file bugs into a bug-tracking system that has certain priority definitions. Operations staff file issues into a separate database, with its own severity levels. Then other specialists such as security analysts come in with yet another list of items to remediate, rated in priority according to their own discipline's or tool's preference, and try to hand it off to everyone else. It should come as no surprise that these different stacks of tasks do not get prioritized and acted on effectively. If you have everyone feed these items into one queue and use the same conceptual structure to assign, prioritize, and track them, then you get a common language across all your team members.

Similarly, if you concentrate on transparency and self-service tools, you also smooth out communication issues. Transparency is empowering both externally to your customers and internally among your teams. We followed best practices in implementing a "public health dashboard"[13] to communicate the system state to end users and, using the same thinking, implemented Splunk to bring real-time production logs directly to the developers. The developers were extremely excited about being given such direct access to information — and isn't transfer of information the goal of communication? One developer put it best when he said that on our devops team, instead of providing a process interface between development and operations, "You guys give us an API instead." This kind of communication is efficient and makes team members happy and effective.

COORDINATION: HAVING THE SAME GOALS

Much of the hostility between functional silos is based on legitimately competing goals. If the developers are incentivized to deliver new functionality rapidly, but responsibility for uptime, performance, and security is laid at the feet of the operations team, you have created your own recipe for conflict. Figure out how to reward

the entire team for overall service delivery — great functionality that is provided in a manner in which a user or customer can use it.

In our IT department's Web team, we had the traditional division of responsibilities between development and operations. This resulted in production problems persisting for months or even years without being fixed. Operations was resentful about the continual work this created for them, but developers were judged on how many new features they developed — therefore, a stalemate came about. Operations felt forced to reduce the rate and scope of changes to the system in the name of uptime, thereby decreasing the rate of delivery of new features.

This low quality of service delivery did not go unnoticed by the business owner, who initiated a program with both developer and operations leadership to define what he expected of IT. The five pillars he defined — availability, performance, agility, total cost of ownership, and ratio of new development to maintenance — were goals expected of the entire organization. This went a long way toward both appropriately prioritizing fixes in the developer groups and investing the operations team in proactive product delivery and not simply "playing defense" all the time.

Not only did the new approach improve uptime and agility at the same time, but it improved morale overall. Most people want to do a good job and create a meaningful work product. If you have groups with mutually contradictory goals, then either they war with each other to achieve what they think is right, or they give up and do not put effort or emotion into their work. But if their goals are aligned, you free up those teams to spend their effort on the task at hand and not expend it in friction with the other teams.

Finally, don't fall into the trap of having separate "functional" and "nonfunctional" requirements that are assigned to different teams. Require your product owners to define all the parameters of an acceptable solution, including service quality parameters, and reward all teams — or hold them all responsible — for that overall picture.

CELEBRATION: FAIL FAST BUT SUCCEED LOUDLY

You're going to be asking people to do things they have never done before. Don't punish failure, or else people will retreat into their previous niches. You will want to provide plenty of encouragement of efforts down the path of collaboration and celebrate successes widely to reward people for their achievements.

One IT department I worked in had a small organizational development team that handled training and similar tasks. Its members were persistent advocates of recognition and continually evangelized it to the organization's managers and technical leads. This was seen as a frivolous use of resources by some, but you could clearly differentiate the teams whose managers had listened from those that had not. The teams that bought into the culture of recognition had noticeably higher *esprit de corps* and retention.

Don't punish failure, or else people will retreat into their previous niches.

Celebrating successes in a clumsy manner, though, can undo all your good intentions. I remember one organization in which the regular celebratory post–go live emails were widely considered a joke. The higher-ups would effusively praise all the business analysts, mention the developers in passing, and never mention the operations staff at all (even when the project at hand was a systems upgrade project managed by the operations team!). The results were more corrosive than not recognizing success at all. Even though those concerned always laughed it off, it created trust barriers between the teams.

On our devops team, like most teams, we have to force ourselves to take the time to celebrate — to have product shipping parties, get-togethers with other teams, and the like. We learned that we needed to actually allocate time to do things like that; in the crunch of a sprint, it's difficult to take time to celebrate, and team members end up skipping out to work anyway. But the more that happens, the more you build up long-term issues that are the social equivalent of technical debt — "cultural debt." Think of celebration, recognition, and similar activities as necessary work to reduce your level of cultural debt.

CONSISTENCY: MAINTAINING ENTHUSIASM

People forget if you don't reiterate the basics. I'll be honest — this one sneaks up on me from time to time. On our current team, after we went through all the initial definition, worked out our process, ran some iterations, and delivered a couple of products, I thought that our new devops way of working had soaked its way in and was the new default. Then we had a couple high-level product decisions that scrubbed one of our

iterations halfway through and threw us out of our rhythm. As we quickly regrouped, I was surprised by some team members who seemed to have forgotten elements of what we were doing and why we were doing it — even things I thought were fundamental parts of the approach we'd all explicitly discussed and agreed on in the past. But not everyone is a high-level strategic thinker; the more tactically focused people need to hear the team's goals and techniques and rationale reinforced over and over again. I have a personal grudge against repetition. When I have to repeat myself, it bothers me on some fundamental level. But I've had to teach myself to embrace repetition — not just in the short term, but over time — in order to maintain alignment.

Just because you have selected a good team and they seem to be "getting" your new devops setup, you can't just coast. You have to continually evangelize internally as well as externally and maintain enthusiasm instead of inertia. Sometimes it seems like a waste of time to have quarterly reviews of your process or architecture: "It's the same as it was last time, people; let's get to work!" However, even though I personally hate to do it, it is one of the keys to success with culture change.

CONCLUSION: IT'S ABOUT THE PEOPLE

You can have all the cutting-edge automation tools you want — continuous deployment, Chef or Puppet, the whole works — and you can have everyone working in a lovely Scrum, but none of these things are solving your business problems. The promise of devops is not that it will reduce some costs or eliminate some internal hurdles, but that you can leverage important skills and knowledge to provide a product or service to your customers. In the end, this isn't a transactional activity; it is a profoundly personal activity, and all the "messy details" of life — relationships, communication, respect — are fundamental to it.

Devops is a culture, and maintaining a culture requires work. Many a startup has grabbed up a couple smart people, added free sodas, and crowed about their "culture." It lasts until they grow that next little bit, or face a major challenge, or think about doing an IPO, and then it turns into "every other company," because no one is consciously curating it.

Contrast that with the approach of National Instruments (NI),[14] which has a 100-year business plan.[15] When people hear that, they get startled. How can you have a 100-year plan? Because that plan goes beyond products or technology or even business — it's about the company culture. The folks at NI know that they want to maintain that culture in the long term, and they know that doesn't happen by accident.

As a leader hoping to gain the benefits of devops, you likewise need to take a long view of culture. It's not something that happens overnight or something that continues without effort. It requires you to give more than lip service to winning your people's "hearts and minds." But that investment will enable them to work together effectively to bring you success.

It's hard, because when there are setbacks, people want to do what's comfortable and re-silo. It's difficult to be confronted with a different language, with people who have different concerns than you do:

- "Let's limit the spec brainstorm to just developers so it doesn't get out of control."
- "Let's not send our ops status report to the developers; it'll just confuse them."
- "We don't need to involve everyone in the concept stage of the product. I'm sure we know all our capabilities without that."

All these statements are comfortable and reasonable — and all are totally wrong. If you fragment communication, you then fragment effort, then organization, and this vicious cycle ends up in enmity. When you unify communication, and effort, and organization, you build solidarity. When more people are working in solidarity on your product, not only do you gain efficiency from alignment of effort and elimination of friction, but you are also able to capture and catalyze more ideas — and that's the currency of innovation.

ENDNOTES

[1]Honor, Alex. "People over Process over Tools." dev2ops, 23 February 2010 (http://dev2ops.org/blog/2010/2/23/people-over-process-over-tools.html).

[2]Manifesto for Agile Software Development (http://agilemanifesto.org).

[3]Wilson, Scott. "Devops." Agile Operations, 19 November 2009 (http://agileoperations.net/index.php?/archives/24-Devops.html).

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

[5]Mueller, Ernest. "Before DevOps, Don't You Need OpsOps?" The Agile Admin, 12 March 2010 (http://theagileadmin.com/tag/opsops).

[6]Turnbull, James. "Bridging the Great Dev/Ops Divide." TechNewsWorld, 30 June 2011 (www.technewsworld.com/story/72777.html).

[7]Arundel, John. "Spliffs and Submarines: The Two Cultures and the State of Devops." Agile Web Development & Operations, 23 November 2010 (www.agileweboperations.com/spliffs-and-submarines-the-two-cultures-and-the-state-of-devops).

[8]Spolsky, Joel. *Smart & Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent*. Apress, 2007.

[9]Allspaw, John. "Some WebOps Interview Questions." *KitchenSoap*, 26 May 2010 (www.kitchensoap.com/2010/05/26/some-webops-interview-questions).

[10]Verber, Mark. "How Many Administrators Are Enough?" *Unix Review*, April 1991 (www.verber.com/mark/sysadm/how-many-admins.html).

[11]*ITIL Service Design*. ITIL V3. Office of Government Commerce (OGC), 2011.

[12]"Tuckman's Stages of Group Development." Wikipedia (http://en.wikipedia.org/wiki/Tuckman%27s_stages_of_group_development).

[13]Rachitsky, Lenny. "7 Keys to a Successful Public Health Dashboard." *Transparent Uptime*, 1 December 2008 (www.transparentuptime.com/2008/11/rules-for-successful-public-health.html).

[14]In the spirit of full disclosure, I should note that I work for NI as its Cloud Systems Architect.

[15]"NI Culture and Stakeholder Engagement." National Instruments, 2010 (www.ni.com/citizenship/stakeholder.htm).

Ernest Mueller is the Cloud Systems Architect at National Instruments in Austin, Texas. He has worked as a developer, systems engineer, and IT manager in various industries and environments. His current focus is on evangelizing and implementing cloud computing and devops to deliver software as a service in the scientific and engineering space. Mr. Mueller can be reached at ernestm@mindspring.com.

Where Is IT Operations Within Devops?

by Bill Keyworth

It would seem that the devops discussion is mostly driven by development's incentives, and appropriately so, given developers' focus on building functionality for the business user. So it's no surprise that development is the originator of the whole devops lifecycle, but are there any dangers lurking in a one-sided focus on devops issues?

A hefty majority of devops articles come from writers of the development persuasion who are motivated by the legitimate frustrations of the application deployment process. The movement to agile development has been a key contributor in the increase of handicaps encountered as a result of more frequent transitions from development *to* operations IT groups. Online and verbal discussions identify the primary challenge as getting IT operations to be more creative and flexible in their approach to changes coming *from* the application development discipline.[1]

Given its critical involvement in deploying new and enhanced applications, why hasn't IT operations been more active within the devops movement? Given the opportunity for IT operations to be more effectively heard by its primary IT companion organization, why hasn't IT ops been more responsive to devops strategic initiatives? Given that changes to the operational model are almost guaranteed, why isn't IT ops more proactive in anticipating such changes? Given the need for more effective alignment with the business user, whose automation needs are frequently accommodated by inhouse development, why hasn't IT ops seized the chance to leverage this obvious path to improving IT's core value proposition to end-user communities? I've heard many theories, but few have resonated.

Devops is a maturing discipline and is obviously offering an increasingly visible value proposition for the enterprise. This growing devops "maturity" is driving more effective partnerships and better integration opportunities. That's the good news. The bad news is that the devops partnerships and integrations are not coming fast enough to appease the significant IT market demand. One proof point is the rapid escalation in demand for SaaS application deployment, which is a symptom of the ongoing struggle between development and operations groups in satisfying end users' business demands. Competitive options now exist (and are expanding) for businesses to choose alternatives to IT-delivered and -supported applications, which further fuels the customer's questioning of IT's contribution to achieving corporate business objectives.

"SERVICES" MINDSET OF IT OPERATIONS ESSENTIAL TO DEVOPS

Fundamental to understanding the mindset of IT operations is its emphasis on delivering and maintaining "services" that are of value to the business community.

> Solutions to business problems and support for business models, strategies, and operations are increasingly in the form of services. The popularity of shared services and outsourcing has contributed to the increase in the number of organizations who are service providers, including internal organizational units. This in turn has strengthened the practice of service management and at the same time imposed greater challenges upon it.[2]

An essential requirement for devops success is to think of, prepare, implement, and support the new application as a mandatory component of a service that is being offered to the business customer. That service is positioned as having an intrinsic value that is broader than the application by itself; as a result, the business executive can easily identify it as critical to achieving business goals. IT operations brings a rich experience of managing those services to devops initiatives. Ops needs to be more aggressive about sharing that unique perspective at the same time that IT development needs to be more attentive to those IT service management (ITSM) best practices that are documented, promoted, and used on behalf of devops by its core practitioners.

IT operations has struggled for decades to deliver more and more technology services with fewer and fewer resources — and is actually doing a fair job.[3] Fundamental to that achievement has been a focus on ITSM best practices specifically designed for improving the operational alignment between IT and its business customers. These guidelines and processes have been

captured and documented by leading ITSM professionals for the Information Technology Infrastructure Library (ITIL), funded by the UK's Office of Government Commerce (OGC).

As a starting point for understanding the mindset, roles, and values of IT operations within the devops discussion, ITIL V3's focus on delivering business value through more effective IT services is invaluable. ITIL positions these best practices as "based on expert advice and input from ITIL users [and] … both current and practical, combining the latest thinking with sound, common sense guidance."[4] The success of that "common sense" approach has contributed significantly to the rapid acceptance and global implementation of ITIL V2 and V3 by ITSM organizations over the last 10 years.

BRIDGING THE DEVOPS GAP

So what might the operations perspective on devops initiatives be? I would offer that ITIL's V3 Application Management (AM) function[5] is probably one of the more useful resources available for describing that operational viewpoint. It is written as a set of best practices or guidelines for supporting ...

> the organization's business processes by helping to identify functional and manageability requirements for application software, and then to assist in the design and deployment of those applications and the ongoing support and improvement of those applications.[6]

As such, ITIL AM is designed to help bridge the devops gap and articulate some of the common language or terminology now lacking between the development and operations organizations.

As a process improvement approach, the Capability Maturity Model Integration (CMMI) has become a highly successful, best practice model for software engineering. CMMI is a set of process guidelines driven primarily by development's need to deliver applications of value to the business user and secondarily by how the end customer uses the application as a service to achieve business purposes. This is a subtle yet critical distinction. The CMMI model does not adequately address the service mentality and service priorities of the IT operations organization, which has fully embraced ITIL for that role instead.

In the August 2011 issue of *Cutter IT Journal*, Bill Phifer of HP Enterprise Services created a compelling case for embracing a host of ITSM processes, with the ITIL Service Lifecycle becoming the needed counterbalance to the development perspective that is incorporated and respected within CMMI.[7] The strength of ITIL is its unrelenting emphasis on "serving" the needs of the business community and adapting any IT deliverable to the "whole" of the system — as perceived by the business customer, not IT operations. For devops, this requires a correlation of management processes and tools that can address the demands coming from end users, business functions, and technologies as well as applications. ITIL in its intended form[8] does not create a focus on IT management processes for the sake of IT personnel. An ITIL-oriented project fails miserably when such an IT-only focus happens.

The CMMI model does not adequately address the service mentality and service priorities of the IT operations organization, which has fully embraced ITIL for that role instead.

ITIL V3 APPLICATION MANAGEMENT FUNCTION

Coming at devops from a "Service Operation" perspective, ITIL V3 defines the ITIL AM function as a shared responsibility of IT operations and IT development in satisfying the service requirements of the end user. ITIL AM can therefore become a complementary yet essential best practice approach for devops due to its operational focus.

Figure 1 depicts the lifecycle of shared processes required of both Application Management and Application Development in order to effectively deliver the service that the business requires.[9] The devops principle that becomes obvious in this diagram is that some phases in the ITIL AM lifecycle are clearly driven by the development group while others are best driven by the operations group. This raises the question, "Who is actually driving each of these phases?" Too often development is driving operations-oriented phases because operations is not yet on board. Successful devops initiatives find a way to ensure that each phase is led by whatever individual, team, or group possesses the necessary experience, skills, and vision to deliver that lifecycle stage as a business service component. This person or group may have a background in development or operations … or, optimally, experience in both.

As indicated previously, responsibility for the ITIL V3 AM function is shared between development and operations (devops). Given the secondary, yet complementary, role of IT operations in the development and deployment of applications, a successful devops team is going to see to it that IT ops makes its contribution in each of the AM lifecycle phases. This contribution is

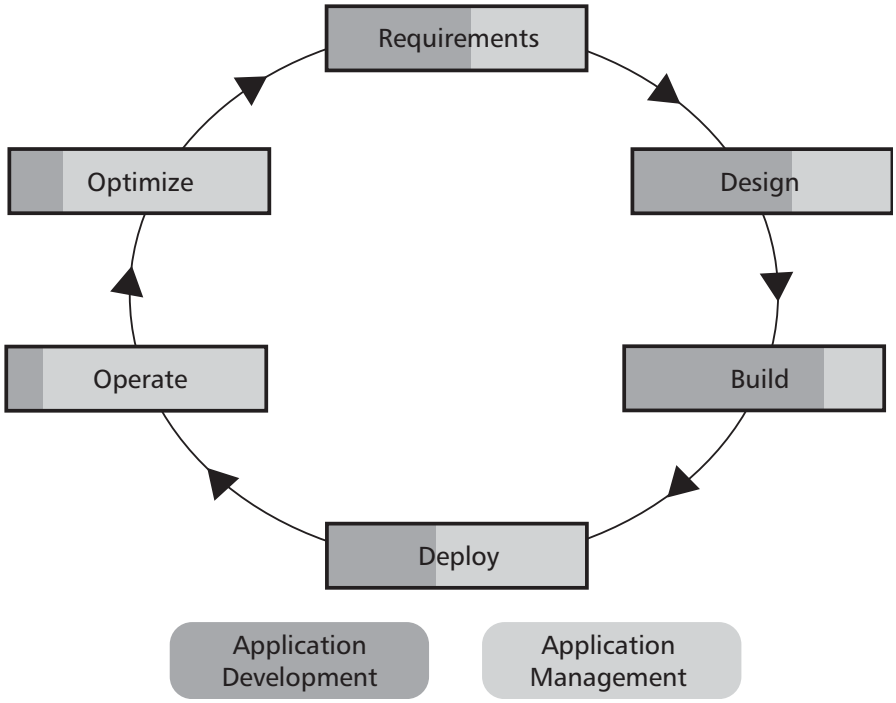


Figure 1 — Roles within the Application Management Lifecycle.

frequently the underappreciated half of the devops equation because, as we've seen, the drive for application deployment originates from development. The phases are as follows:

- **Requirements phase.** Clarity regarding manageability requirements is as critical to the service perspective of the end user as the functionality, architectural, and usability requirements, yet too often they are delayed until the second version and handled through an enhancement process. This has been such a consistent pattern in my ITSM career that I sound like a broken record in my emphasis on this up-front (not subsequent) inclusion. Delay in satisfying manageability requirements jeopardizes the acceptance and early use of the application by the business user, who can't get the needed training, support, security, monitoring, and self-service.

  Interface requirements that ensure the new application satisfies dependencies for existing IT management and security tools are paramount. Who can define those dependencies except those who manage and use these ITSM applications? How often do teams postpone the definition of service-level requirements until a later phase of the AM lifecycle, only to find out too late that they cannot adequately specify the application's performance metrics (other than "It should initially work"), the quality of the application's output (reporting, dashboards, business intelligence, etc.), or any other qualitative or quantitative aspect that the business customer or user needs to measure?

- **Design phase.** In addition to leveraging the requirements into the design of the application and environment, where is the design accommodated and reviewed for changes to the operational model that the application has to adhere to? For applications that are purchased rather than developed, the responsible business unit is typically able to offer feedback regarding functionality, but there is still a need for IT operations to provide feedback to the software vendor about the manageability and performance of the software. How early in the AM lifecycle is IT operations empowered to have this discussion with the software vendor or internal development group? Finally, will operations have an opportunity to provide input to the design of intended customization capabilities for manageability and reporting needs?

- **Build phase.** Because this is the phase where application components are coded, integrated, and tested, and the application and operational model are made ready for deployment, it is also the phase in which IT operations makes the least contribution. Yet it is during this phase that there should be a focus on whether the management specifications are satisfied. Testing is integral to this phase, so testing the management of the application within the operational

model should be accommodated to the satisfaction and acceptance of IT operations.

- **Deploy phase.** With the operational model embedded in the existing IT infrastructure, any new application must be deployed within that IT environment and on top of that operational model. Because operations is responsible for this production environment, this is too often the first time non-devops "deployment" initiatives will reach out to IT ops in order to ensure the application actually works as promised after it has been downloaded and installed. IT operations and development are supposedly working together for a predefined period of time (Early Life Support) for testing, validation, and monitoring of the service that contains the new or revised application. However, if functions contained within phases 1-3 (Requirements, Design, and Build) of ITIL AM have been handled outside of IT ops, then frustration and accusations become the norm and devops simply cannot realize its promise.
- **Operate phase.** This phase is obviously the realm of operations; it's where the name of this IT organization originates. According to ITIL V3, "The IT services organization 'operates' the application as part of delivering a service required by the business. The performance of the application in relation to the overall service is measured continually against the Service Levels and key business drivers."[10] Applications are only one of the many elements that are necessary to initiate, provide, and maintain a business service as defined and understood by the end users, and IT operations is responsible for operating all these service assets with their constant changes, unpredictable events, illogical problems, and difficult demands (requests) by the end users. Is it any wonder that ops staff overreact when confronted with an application for which they are unprepared?
- **Optimize phase.** Improving service levels and lowering the costs of an application are two IT operations strategies that should always be optimized. Neither will ever be "good enough." As devops goes through this ITIL AM lifecycle for each new application, it must also go through the lifecycle for each application enhancement or modification. Operations must maintain previous versions of the application while simultaneously engaging in the devops lifecycle for the next version. Agile development ensures that an iterative devops approach is continually ongoing. In addition to optimizing the application, IT ops should also be constantly monitoring the ITIL AM function for continuous improvement possibilities.

ITSM-ORIENTED ACTIVITIES FOR DEVOPS

There are a host of activities that are part of IT operations' unique contribution to devops — too many to attempt a complete list. Following is a small sampling of ops activities included within ITIL AM:

- Instrumentation of applications for generation of meaningful events
- Scenarios to predict impact of potential changes to utilization, functionality, manageability
- Application sizing, volume metrics, load testing for availability and capacity planning
- Recruiting and training programs to ensure skilled resources for AM
- Operational models to ensure optimal use of system resources
- Error analysis, bug tracking, patch management, antivirus for inhouse applications
- Vendor management to manage contracts and deliverables of purchased applications
- Project involvement for operating system upgrades, server consolidation, physical moves
- Definition, review, and maintenance of application documentation (user/admin manuals, standard operating procedures)
- Process metrics for response time, incident resolution, changes backed out, security issues

Improving service levels and lowering the costs of an application are two IT operations strategies that should always be optimized. Neither will ever be "good enough."

ITIL V3 SERVICE MANAGEMENT LIFECYCLE

One of the important contributions of ITIL V3 is the way the AM function is accommodated within the larger Service Management Lifecycle (see Figure 2).[11] Applications are services in the eyes of the business user, yet their design, development, deployment, and management are distinctly different from almost all other IT services. However, just because application-driven services are markedly different from other IT services doesn't mean application developers can ignore the time-tested principles of Service Management. For example, ITIL V3 documents how Application

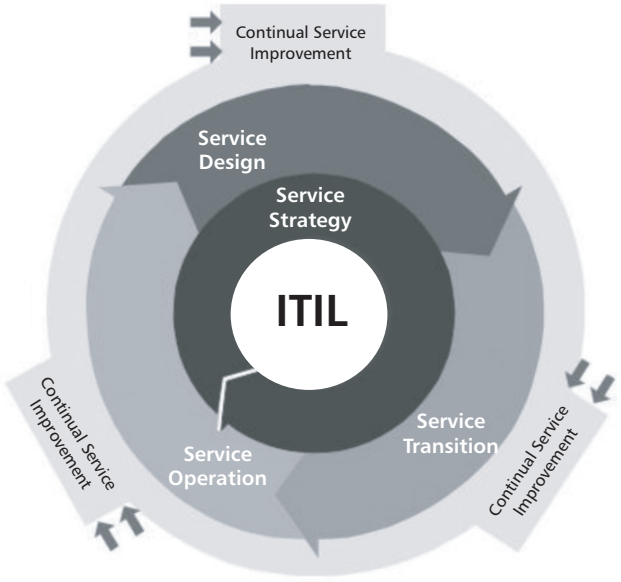


Figure 2 — ITIL Service Management Lifecycle.

Management is integrated with each of the five phases of the Service Management Lifecycle:[12]

1. **Service Strategy** — defines criteria for developing applications inhouse, outsourcing development, purchasing or customizing applications; defines how applications fit within the service portfolio of all IT services; includes ROI analysis for both applications and services they support
2. **Service Design** — establishes requirements for manageability in addition to functionality; works to ensure such manageability objectives are satisfied during the Requirements and Build phases of ITIL AM and therefore fit within the design of the targeted service offering
3. **Service Transition** — includes testing, validating, and deploying applications operationally within the targeted service offering as intended for the business end user; enables operational change processes to be examined, accommodated, and enforced
4. **Service Operation** — delivers support, maintenance, training, and metrics required by the business end user in the achievement of the service's business goal; institutes service desk processes for event management, incident management, problem resolution, and request fulfillment for application support
5. **Continual Service Improvement** — measures the quality and relevance of the applications within the operation of the business service and provides recommendations on potential improvement; ensures feedback from the user to operations and development

Each of these ITSM activities should be addressed within the scope of devops. Documentation, training, and expertise on how to fulfill the AM function are available in the ITIL V3 resources.

IT'S CULTURAL FOCUS ON CHANGES

Probably the number one guiding principle of IT operations is that unplanned changes are the root of most failures. That mindset drives every process, tool, and activity within the ITIL Service Management Lifecycle. Consequently, it is in the DNA of operations personnel to resist anything that might introduce unpredictable changes within the IT infrastructure. IT ops is rewarded (not penalized) for consistency and for preventing the unexpected or unauthorized from happening. Conversely, development staff are rewarded for creativity in solving business problems and for project completions in a timely manner. Metrics that foster these desirable behaviors are going to ensure the two groups are in natural conflict — hence the need for and promise of devops in resolving this disconnect.

Knowing that the IT operations culture has an aversion to change, the devops group would obviously address that mindset early and continuously in the ITIL AM lifecycle. ITIL change management identifies seven questions[13] that IT ops staff must answer with regard to all changes proposed to the IT infrastructure and operational model:

1. Who raised the change?
2. What is the reason for the change?
3. What is the return required from the change?
4. What are the risks involved in the change?
5. What resources are required to deliver the change?
6. Who is responsible for the building, testing, and implementing the change?
7. What is the relationship between this change and other changes?

Without agreement on the answers to these seven questions, any assessment of impact, cost, or risk of the proposed changes cannot be completed. Furthermore, there will be no understanding of how responsibility for the changes to the desired business service is to be shared between development and operations. More important, the business benefit that is motivating development of the application will not be completely achieved; the application could even have a negative effect on the existing services provided by all of IT to the business

end users. The core culture of IT operations demands that devops address the impact of changes to the existing and intended services offered to the business community.

CONCLUSION

Long-term success in devops is completely dependent upon the development of common metrics that are recognized and accommodated by both IT operations and development — not one side only. A change in perspective is needed where IT operations comes to the devops table and is motivated by something other than making development's work easier. ITIL V3 Application Management assists devops efforts in detailing how the application itself fits into the larger delivery of service management for the business service purchased by the IT customer, providing guidelines and process definition for that operational integration — the thing that is most often missing from early devops initiatives.

Operations personnel should approach the devops opportunity with their own best practices defined and implemented for Application Management, recognizing that development's objectives will be different from their own, but equally critical in fulfilling the business needs of the end user. Development should approach the devops opportunity with the intention of better understanding the unique needs and demands that are driving the operations function.

ENDNOTES

[1]Read, Chris. "DevOps: State of the Nation." Agile Web Development & Operations, 30 November 2010 (www.agileweboperations.com/devops-state-of-the-nation-by-chris-read).

[2]*Introduction to the ITIL Service Lifecycle*. ITIL V3. Office of Government Commerce (OGC), 2010.

[3]McAfee, Andrew, and Erik Brynjolfsson. "What Makes a Company Good at IT?" *The Wall Street Journal*, 25 April 2011.

[4]ITIL (www.itil-officialsite.com).

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

[6]*ITIL Service Operation.* See 5.

[7]Phifer, Bill. "Next-Generation Process Integration: CMMI and ITIL Do Devops." *Cutter IT Journal*, Vol. 24, No. 8, 2011.

[8]Fry, Malcolm. *ITIL Lite: A Road Map to Full or Partial ITIL Implementation.* The Stationery Office, 2010.

[9]*ITIL Service Operation.* See 5.

[10]*ITIL Service Operation.* See 5.

[11]*Introduction to the ITIL Service Lifecycle*. See 2.

[12]*ITIL Service Operation.* See 5.

[13]*Introduction to the ITIL Service Lifecycle*. See 2.

Bill Keyworth is a Senior Consultant with Cutter Consortium's Business Technology Strategies *practice. With more than 25 years' experience successfully defining technology and market trends within the IT service management industry, Mr. Keyworth has established a reputation as a credible and consistent voice in maximizing business value from IT operations.*

Leveraging telecom and network management experience gained at AT&T and Digital Equipment Corporation, he was among the first to identify and justify market requirements and benefits for a network management (monitoring) framework, thrusting him into an early leadership role within the IT management industry. As former VP and Research Director at Gartner Group, Mr. Keyworth initiated the successful Network and Systems Management (NSM) service and helped to create a US focus on ITIL processes years before industry adoption. Through research and consulting roles at Gartner and Ptak, Noel & Associates, he has led various strategies on IT infrastructure management that are critical for managing the dynamic elements of Internet and enterprise applications. In executive roles at Digital, Peregrine, TCI Solutions, and Nexiant, coupled with marketing consulting roles at six high-tech startups in recent years, Mr. Keyworth has demonstrated expertise in refocusing the marketing engines of an organization and turning product-driven companies into market-driven industry leaders. He now translates that market perspective into success criteria for IT management initiatives and remains Editor-in-Chief of BSMReview.com, a thought leadership site for IT service management's role in successful business initiatives. He holds a BS from Brigham Young University's Marriott School of Management and an MBA from Babson College. Mr. Keyworth can be reached at bkeyworth@cutter.com.

Disciplined Agile Delivery and Collaborative DevOps

by Scott W. Ambler

Agile software development strategies took the world by storm in the early 2000s. Since that time, we've seen organizations around the world experiment with, and then adopt, agile techniques in a range of situations, from the development of simple websites to the improvement of existing legacy applications and even the creation of life-critical embedded software. We've learned along the way, adding and modifying ideas from several agile methodologies as well as concepts from tried-and-true traditional methods (particularly when we're applying agility at scale), and thereby improved our development efforts. In parallel, we've come to recognize that the majority of IT spending is devoted to the operations, support, and maintenance of systems, and only a very small portion to actual development of new functionality. This realization has motivated organizations to adopt strategies from ITIL, and to a lesser extent COBIT, to streamline their operations and support efforts.

Although we've seen improvements on both sides of the IT house, it has become obvious that there are still inefficiencies resulting from differences in the way that development and operations organizations work. For example, poor understanding of operations issues can result in significant rework to make applications integrate with the current infrastructure, deployment dates being pushed back because of little or no release coordination, and operational costs increasing due to a growing number of technology platforms being supported. This misalignment of development and operations is often referred to as the "devops gap," (see Figure 1). The diagram also shows how we can move toward an environment where a devops strategy addresses the integrated activities that streamline the collaboration between your development and operations groups, activities that occur through all phases of the development/delivery lifecycle. In short, devops is based on the lean idea that instead of optimizing your development and operations efforts separately (which may actually make it *harder* for the two groups to work together effectively), it's far better to optimize the whole of IT.

In this article, I present an overview of Collaborative Development and Operations ("Collaborative DevOps" for short) and Disciplined Agile Delivery (DAD). More important, I will describe how DAD explicitly "bakes" devops strategies right into the process framework.

COLLABORATIVE DEVOPS

Collaborative DevOps[1] is based on the observation that effective collaboration between development and operations occurs as a combination of three fundamental dimensions:

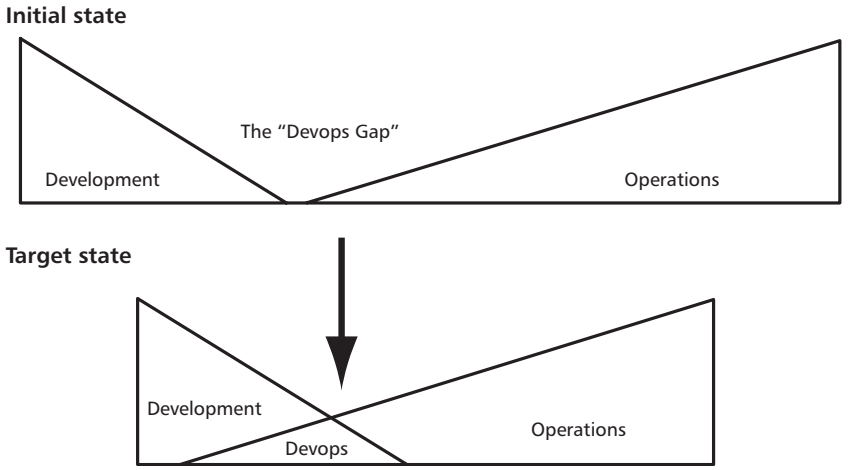


Figure 1 — Bridging the gap between development and operations.

1. **Process integration**. There are activities in your development and operations processes that control transitions between teams, and these activities must reflect the realities of the teams involved. Examples include both the deployment of a solution into production and the resolution of production problems caused by defects in a solution (which therefore must be fixed and redeployed). In both cases, control is passed between the development and operations teams; in the latter case, at least once in each direction.
2. **Tool integration**. This occurs when one tool leverages an interface provided by another. Interfaces may be tool-specific, such as a particular API or one based on an open standard specification such as Open Services for Lifecycle Collaboration (OSLC).[2] For example, to enable the release of a solution into production, a deployment tool could access production configuration information maintained in an asset management tool. To enable reporting of a problem with an application, a support management system could register the defect report into a Jazz repository,[3] and then developers responsible for fixing the problem could pick it up via their own development tools.
3. **Data integration**. This allows information to be shared or linked in a federated manner. For instance, on the development side, a released solution may include quality results, configuration information, and proof of regulatory compliance. In operations, usage statistics are tracked and configuration/installation details are maintained for the solution. Clearly each group requires unique information, but there is also information that both groups share. Federation of this information enables consistency and accuracy between your development and operations teams, which in turn reduces overall costs and even potential outages.

To be successful at Collaborative DevOps, you must address all three of these dimensions, although the focus of this article is the first dimension, process integration. There is a further focus on the DAD process framework, even though Collaborative DevOps is applicable to other methodologies as well, including traditional ones.

DEVOPS PROCESS INTEGRATION IN THE DAD PROCESS FRAMEWORK

The DAD process framework is a people-first, learning-oriented, hybrid agile approach to IT solution delivery.[4] DAD is a hybrid in that it adopts and tailors proven strategies from methods such as Scrum, Extreme Programming (XP), Agile Modeling (AM), the Unified Process (UP), Kanban, and Agile Data (AD). The upshot is that DAD is a second-generation agile process framework that has a risk-value lifecycle, is goal-driven, is scalable, and is enterprise-aware. As you will see throughout this article, DAD bakes operations-friendly activities and strategies right into the process itself, making them an explicit aspect of the overall method. This is important because it ensures that operations concerns are addressed in a streamlined and coherent manner, not tacked onto your project as an afterthought.

The DAD lifecycle, depicted in Figure 2, addresses the project lifecycle from project initiation through construction to the point of releasing the solution into

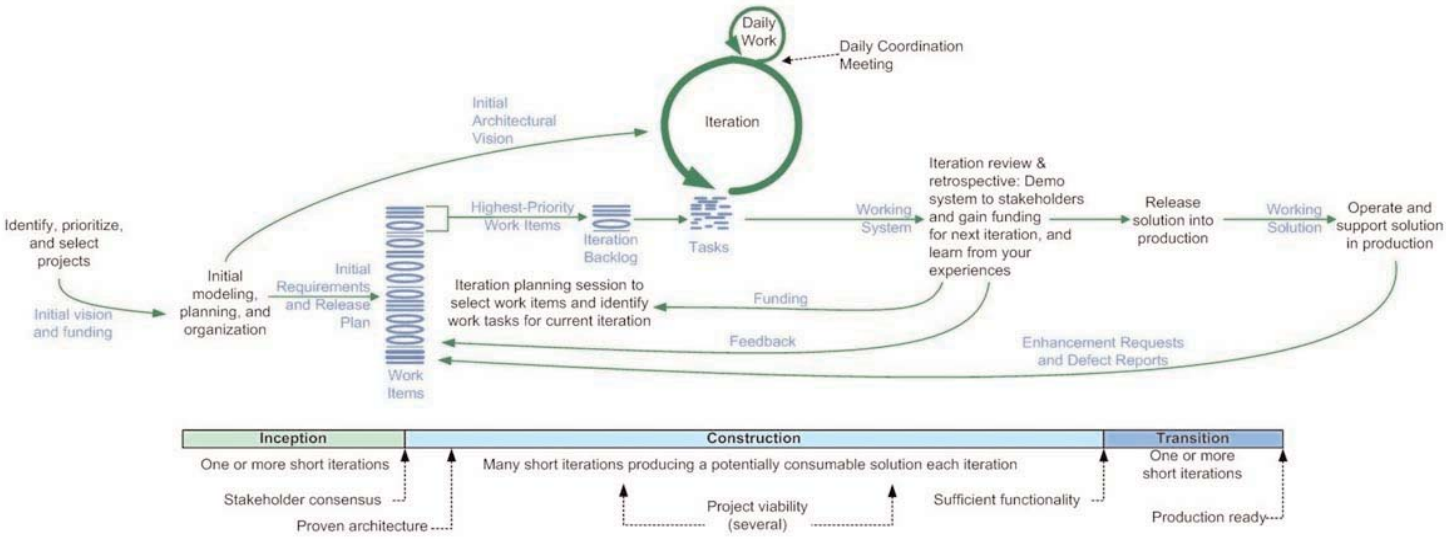


Figure 2 — The Disciplined Agile Delivery (DAD) lifecycle.

production. (It also shows some preinitiation portfolio management activities, as well as postdelivery production activities.) This differs from first-generation agile methods, which typically focus on the construction aspects of the lifecycle, leaving the details about how to perform the rest of it up to you.

DAD teams work within the organization's ecosystem and explicitly try to take advantage of the opportunities presented to them. To borrow an environmental cliché, they "Think globally and act locally."

The DAD process framework includes several key philosophies that enable Collaborative DevOps:

- **Enterprise awareness**. An important consideration for an effective devops strategy is to ensure that your development teams produce solutions that work well with your operational environment. DAD teams recognize that they work within the organization's ecosystem and explicitly try to take advantage of the opportunities presented to them. To borrow an environmental cliché, disciplined agilists "Think globally and act locally." This includes working closely with enterprise technical architects and reuse engineers to leverage and enhance the existing and "to be" technical infrastructure; with enterprise business architects and portfolio managers to fit into the overall business ecosystem; with data administrators to access and improve existing data sources; and with IT operations and support people to understand and follow enterprise IT conventions.
- **A robust stakeholder definition**. A stakeholder is someone who is materially impacted by the outcome of a solution, and for devops to be successful, your development teams must accept that operations and support staff are key stakeholders whom they should work with throughout the project. Of course, operations and support staff aren't the only stakeholders. A stakeholder could be a direct user, an indirect user, a manager of users, a senior manager, an operations staff member, the "gold owner" who funds the project, a support (help desk) staff member, an auditor, your program/portfolio manager, a developer working on other systems that integrate or interact with the one under development, or a maintenance professional potentially affected by the development and/or deployment of a software project.
- **A solution focus**. DAD team members recognize that, as IT professionals, they do far more than just develop software to address the needs of their stakeholders. Software is clearly important, but DAD teams will often provide new or upgraded hardware, furnish supporting documentation (including operations and support procedures), change the business/operational processes that stakeholders follow, and even help change the organizational structure in which their stakeholders work.
- **Explicit governance**. Although agile approaches are based on trust, smart governance strategies are based on a "trust but verify and then guide" mindset. Effective governance strategies motivate and enable development teams to leverage and enhance the existing infrastructure, follow existing organizational conventions, and work closely with enterprise teams — all of which help to streamline operations and support of the delivered solutions. To enable explicit governance, DAD promotes enterprise awareness (as discussed previously); includes explicit, lightweight milestones (as you can see in Figure 2); and supports open and honest monitoring of the team. Monitoring can be enabled when DAD teams use instrumented and integrated tooling, which generates metrics in real time that can then be displayed on project dashboards.[5] Senior operations and support personnel can access the project dashboards to determine the quality of the solution being produced, the status of supporting documentation, and the likely delivery date, among other key tidbits of information that enable them to plan effectively.

Earlier I claimed that devops affects the entire solution delivery lifecycle from end to end. Let's explore each DAD phase one at a time and see how this is true.

COLLABORATIVE DEVOPS FROM END TO END

During Project Inception

Before jumping into building or buying a solution, it is worthwhile to spend some time identifying your objectives for the project. Traditional methods invest a large amount of effort and time in planning projects up front. Agile teams recognize that there is significant risk in detailed up-front planning and specification, but they also acknowledge that there is significant risk in doing no such activity at all. As a result, they have a short initiation effort, which is called "Sprint 0" in Scrum and "Inception" (a term adopted from the UP) in DAD. This project initiation effort takes 3.9 weeks on average,

although it can range from several days to several months depending on your situation.[6] From a devops perspective, there are three critical activities that occur in parallel during the Inception phase:

1. **Initial requirements envisioning**. DAD teams invest time at the beginning of the project to identify the high-level scope for the effort in a lightweight, collaborative manner. To do this, the team's product owner, often supported by other team members, will facilitate agile requirements modeling sessions with key stakeholders to explore their needs. These stakeholders should include operations engineers, support staff, and enterprise architects (among others), as they are important sources of requirements, in particular nonfunctional ones. Common operations requirements may include the need to back up and restore data sources, to instrument the solution so that operations staff can monitor it in real time, or to architect the solution in a modular manner to enable easier deployment. Requirements envisioning is an agile modeling effort that serves to initially populate your work item stack (or "product backlog" in Scrum) and provide key scoping information for initial planning and architecture envisioning efforts.
2. **Initial architecture envisioning**. Disciplined agile teams will also identify a viable architectural strategy that reflects the requirements of their stakeholders and your organization's overall architectural strategy (hence the need to work closely with your enterprise architects and operations staff). This is a key point in time to make sure the team is building (or buying) a solution that will work well with the existing operational infrastructure and to begin negotiating any infrastructural changes (such as deploying new technologies) early in the project. It is also important to ensure that the architecture takes operations-oriented requirements into account early on, as these types of requirements tend to address foundational/overarching architectural issues. Architecture envisioning is a lightweight, agile modeling effort that provides key information for your initial planning efforts, potential constraints on your scope, and technical direction for your construction efforts.
3. **Initial release planning**. Every single project team that I have ever been involved with was asked early in the project to indicate, at least roughly, what it was going to deliver, how much it would cost, how long it would take, and how it was going to do it. In other words, the team was expected to produce a viable plan up front. This plan should be a bit fuzzy (or ranged, if you will) at first and tighten up based on actual information as the project progresses. As part of release planning, the DAD team works closely with the operations group to identify potential release windows to aim for, any release blackout periods to avoid, and the need for operations-oriented milestone reviews later in the lifecycle (if appropriate).

The three practices described above are among several that help a DAD team formulate a solid, albeit lightweight, foundation from which to proceed with construction.

DAD teams know that they must consider operations issues throughout the lifecycle and validate that they're doing so through concrete tests.

During Construction

The DAD Construction phase is the period of time during which the required functionality is built (or configured, in the case of purchased solutions). The timeline is split up into a number of time-boxed iterations. At the end of each iteration, the team will have produced and regression tested a demonstrable increment of a potentially consumable solution. The phase ends when there is sufficient functionality to justify the cost of transition (sometimes referred to as a "minimally marketable release") and the key stakeholders have agreed that the functionality is acceptable to them.

There are several agile strategies DAD teams commonly follow during the Construction phase that support Collaborative DevOps:

- **Active stakeholder participation**. DAD teams work closely with their stakeholders all the way through the lifecycle to ensure that their evolving needs are understood. This may involve negotiation of changes to the operational infrastructure, such as the addition of new technologies to support the coming release or the retirement of existing technologies that the team may have depended on in the past.
- **Continuous integration**. This is a common agile technical practice where the solution is built/compiled and regression tested; in advanced situations, the solution is run through static and dynamic analysis to look for potential defects. Integration is performed many times a day within developer

environments, at least daily within the project integration environment, and regularly within your preproduction test environment. Some of the regression tests will validate that the team has addressed operations' concerns, such as installation, performance, security, and integration issues (to name a few). DAD teams know that they must consider operations issues throughout the lifecycle and validate that they're doing so through concrete tests.

- **Independent testing**. In simple situations, agile teams will take a whole-team approach to testing, where the testing is performed by the development team itself, often by embedding testers into the team. For enterprise-class development or solution delivery at scale, particularly in regulatory environments or when the domain or technology is very complex, disciplined agile teams will find they need to support their whole-team testing efforts with an independent test team running in parallel to the development team.[7] This team performs independent validation of the solution (important for some regulatory regimes), but more important, it focuses on testing issues that development teams often struggle with. These testing issues frequently include validation of nonfunctional requirements — such as security, performance, and availability — and those having to do with production system integration. All these issues are of clear importance for your operations department, which is one of several reasons why the DAD process framework suggests this strategy. In midsized to large organizations, the economics and logistics of production-level system integration testing motivate a separate test team that supports many development teams. Development teams will provide their working build to the independent test team on a regular basis (at least once an iteration), and the test team will then test to the risk and report potential defects back to the development team.
- **Continuous deployment**. With this practice, you automate the promotion of your working solution between environments. For example, after your work successfully integrates in your development environment, it can be automatically promoted to your project integration environment. When the system integrates successfully in that environment, it can be automatically promoted into your testing environment, and so on. Different forms of testing will occur at each level; for example, you may perform installation testing in your independent test environment but not your development environment. Deployment into production is generally not automatic, as this is an important decision to be made by your operations/release manager(s).[8] By automating as much of the deployment effort as possible, and by running it often, the development team increases the chance of a successful deployment and thereby reduces the risk to the operations environment.
- **Continuous documentation**. With this practice, supporting documentation, including operations and support documentation, is evolved throughout the lifecycle in concert with the development of new functionality. Many DAD teams will include this documentation in their definition of what it means for their solution to be potentially consumable (an expansion of Scrum's "potentially shippable" concept).

It's important to note that the DAD process framework is goal-driven, not prescriptive. So what DAD actually recommends is that you address issues such as testing, documentation, and understanding stakeholder requirements in some way; it then suggests practices such as those listed above. It does not insist that you follow those practices — other options are available to you. It is up to you to tailor DAD to meet your unique needs; DAD merely suggests good options and discusses the tradeoffs when there are several viable alternatives available.

During Transition

The DAD Transition phase focuses on delivering the system into production (or into the marketplace in the case of a consumer product). There is more to Transition than merely copying some files onto a server, and as a result, the time and effort spent transitioning varies from project to project. Shrink-wrapped software entails the manufacturing and distribution of software and documentation. Internal systems are generally simpler to deploy than externally available systems. High-visibility systems may require extensive beta testing by small groups before being released to the larger population. The release of a brand-new system may entail hardware purchase and setup, while updating an existing system may require data conversions and extensive coordination with the user community. Every project is different. This phase ends when the stakeholders are ready and the system is fully deployed.

There are several potential Transition practices DAD teams may follow that support Collaborative DevOps:

- **Production release planning**. This is the subset of your release planning efforts that focuses on

the activities required to deploy into production. Technical activities include backup of the existing data, baselining of the solution to be deployed, restoration of the existing solution in case of problems, and so on. Technical activities can often be automated as part of your continuous deployment efforts. Nontechnical activities include training and education of people affected by the release (including operations staff), communication activities surrounding the release (e.g., advertising), transitioning of team members to the next project, and so on. You should plan for and execute all these activities as appropriate.

- **Production readiness reviews**. There should be at least one review, performed by the person(s) responsible for your operations environment, before the solution is deployed into production. For important systems, such as those that are life- or mission-critical, there is likely to be a series of such reviews throughout the project, becoming more formal the closer you get to the actual deployment.
- **End-of-lifecycle testing**. Minimally you will need to run your full automated regression test suite against your baselined code once construction ends. In complex situations, there may be manual acceptance reviews or testing to be performed, one last independent testing effort, and then any appropriate fixing and retesting required to ensure that the solution is truly ready for production.
- **Finalize documentation**. Although many DAD teams adopt AM's continuous documentation practice described above, in reality the documentation can lag behind development efforts, something that is particularly true toward the end of construction. Your operations and support staff know this and will work with the development team to "clean up" its supporting documentation during the Transition phase to ensure that it is of sufficient quality. On many non-DAD project teams, operations and support documentation is written as an afterthought, if it's written at all. This increases their overall risk.

For many DAD teams, the planning and review effort will start partway through the Construction phase. Furthermore, Construction practices such as continuous deployment and active stakeholder participation are clearly appropriate during the Transition phase.

PARTING THOUGHTS

In my experience, the organizations that succeed at bridging the devops gap do so by baking devops activities right into both their development and operations processes. The most effective way to address operations concerns is to do so directly in your delivery process. I also believe that while better integration of development and operations is a great start, and one that will take organizations many years to truly achieve, it isn't the final end game. What organizations really need is effective integration of IT and the rest of the business — perhaps "BusDevOps," were I to coin a new term. More on this topic in a future article.

ENDNOTES

[1]Collaborative Development and Operations. IBM, 2011 (www.ibm.com/software/rational/devops).

[2]Open Services for Lifecycle Collaboration (OSLC) (http://open-services.net).

[3]Jazz Community Site (www.jazz.net).

[4]Ambler, Scott W., and Mark Lines. "Disciplined Agile Delivery: An Introduction." IBM, 2011 (http://public.dhe.ibm.com/common/ssi/ecm/en/raw14261usen/RAW14261USEN.PDF).

[5]You can see live examples of such dashboards for several IBM agile development teams at www.jazz.net.

[6]Ambler, Scott W. "Agile Project Initiation Survey Results: July/August 2009." Ambysoft, 2009 (www.ambysoft.com/surveys/projectInitiation2009.html).

[7]Ambler, Scott W. "Agile Testing and Quality Strategies: Discipline Over Rhetoric." Ambysoft, 2007 (www.ambysoft.com/essays/agileTesting.html).

[8]Humble, Jez, and David Farley. *Continuous Deployment: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.

Scott W. Ambler is IBM Rational's Chief Methodologist for IT, working with IBM customers around the world to help them improve their software processes. He is a contributor to Cutter Consortium's Agile practice. Mr. Ambler is the founder of the Agile Modeling (AM), Agile Data (AD), Disciplined Agile Delivery (DAD), and Enterprise Unified Process (EUP) methodologies and creator of the Agile Scaling Model (ASM). He is the author or coauthor of 20 books, including Refactoring Databases, Agile Modeling, Agile Database Techniques, The Object Primer 3rd Edition, The Enterprise Unified Process, *and the forthcoming* Disciplined Agile Delivery. *He is a Senior Contributing Editor with* Dr. Dobb's Journal. *Mr. Ambler can be reached at website: www.ibm.com/software/rational/leadership/thought/scottambler.html; or* Agility@Scale *blog: www.ibm.com/developerworks/blogs/page/ambler.*

Metrics-Driven Devops

by Alexis Lê-Quôc

Devops as a cure for the dysfunctional gap between development and operations is here to stay. Complex applications built as an orchestration of highly distributed services, some internal, some outsourced, demand that development and operations find a common language in which to collaborate.

Rather than discussing the toolkit *du jour*, I focus in this article on the necessity to anchor the devops conversation in shared and actionable metrics. To that end, I will examine the recent transformations in the way we build and run applications, discuss the resulting need for better metrics, and introduce a simple framework for evaluating a metrics-driven devops practice in the enterprise.

THE RISE OF APIs AND THE DAWN OF DEVOPS

APIs to the Cloud

It is difficult to overstate the fundamental shift that cloud computing is bringing to our industry. So much[1] has been written about it already that I don't need to go over it in detail:

- The concentration of basic infrastructure building blocks in the hands of a small number of large providers has made outsourcing server capacity financially desirable despite the risks, thanks first to sheer economies of scale driving costs down for these providers, but also to a utility model that substitutes operating expenses for capital expenses for their customers.
- These building blocks have been *de facto* standardized by virtue of being only offered at sustainable cost[2] by a small number of providers. For example, computing is offered either as bare virtual machines (VMs) or application servers, while storage is offered as file systems, raw block devices, or full databases.
- This standardization crystallized itself around a small number of APIs — effectively embodying in code the contractual relationship with the provider with unprecedented precision, practicality, and immediacy.

This last point in particular is having a profound impact on the traditional organization of IT departments, classically split into distinct development and operations groups. Of these two, development is the most inclined to use APIs. Developers have the incentive to experiment with them to create new functionality. Developers also have the supporting tools and the testing frameworks in place already. Whether an API is used to assemble a Web page, charge a credit card, or power on a hundred servers deep in the bowels of a hosting provider is largely irrelevant, because what it does is reduced to a few lines of code.

We don't know yet whether IT departments will massively migrate to public cloud providers such as Amazon or will instead deploy smaller inhouse private versions of cloud. In both cases, APIs are here to stay. With them, developers know they can control the full lifecycle of IT infrastructure, so there's no going back.

From Ops to Devops

The very next step for developers is the management of these API-controlled resources, traditionally under the purview of the operations group. This is the essence of the devops movement: development teams start taking on operational roles[3] while operations teams manage infrastructure with code.[4] The boundary between both roles, once an organizational Rubicon, is fading away under business pressure to deliver applications faster.

This shift to APIs shouldn't come as a surprise, as it is merely the repetition of an automation episode similar to the ones every industry inevitably goes through. In our case, substantial parts of application delivery are streamlined by taking the human element out of the cycle.

COMPLEXITY, AGILITY, AND THE NEED FOR METRICS

While it is tempting to see the transition I've just outlined as a beautiful and enabling simplification, we must be careful not to underestimate the additional complexity that (together with accelerated, agile development cycles) it will bring into our environments.

Simple APIs, Complex Services

Just as standardizing on common engine platforms enabled the car industry to mass-produce considerably more sophisticated propulsion designs, our development teams can now mix and match a large number of arbitrarily complex services, all hiding behind deceptively simple APIs.

The cardinal quality of APIs is to hide complex operations behind clear semantics and simple syntax. In fact, with a few lines of code and no lead time, today we can:

- Provision a phone number and text it out to a group to start an impromptu conference call
- Start 100 customer-facing servers with redundant storage, deploy a custom software stack on them, and redeploy half of them every hour
- Provision a clustered relational database
- Provision a distributed Java application

None of these services are easy to provide at scale, but the beauty of APIs is that it becomes someone else's job to keep them running and up to date. API providers take care of what Amazon CTO Werner Vogels calls "undifferentiated heavy lifting": noncore utilities that rich applications depend on. Much like running an electrical grid, it is no small feat, but the end user is blissfully insulated from the intricacies of building and managing complex distributed systems.

As a consequence, the typical application architecture has evolved from mostly monolithic code running across a handful of application servers and databases to highly distributed code running partly inhouse and partly on such and such a provider's resources via a number of API calls.[5]

The often-overlooked flip side of this evolution is that it makes the overall behavior of an application considerably harder to predict. There is more complexity overall: more layers, more inter–data center network round trips, more dependencies. The system, composed of the application, its infrastructure, and its third-party API providers, has become substantially more multivariate.[6]

As a result, we can't rely on our conceptual understanding of these components. Rather than trying to imagine every potential cause of failure of the services we use, we need to quantitatively describe and track the behavior of their APIs. We don't need to know that a faulty memory controller could bring our storage service down. What we do need to understand and track, however, is the service's performance profile, its failure rate, and the distribution of its response latencies.

In other words, the rise of APIs has reaffirmed the need for comprehensive metrics to understand and model complex services we only see as black boxes.

Agile Development Needs Agile Operations

I certainly don't need to present the agile movement to this audience.[7] Agile has gone mainstream with development teams, yet agile development is on a collision course with the traditional mindset of operations:

- **Delivery schedules are different.** We use fast-paced code updates as opposed to slow-paced, carefully planned maintenance windows.
- **The appetite for change is different.** Development sees new code as its *raison d'être*, while operators see changes as the root of all evil.
- **Business mandates are seemingly opposed.** "The more we get done during this iteration the better" versus "The more changes we make, the more outages we get."

Hence the notion of agile operations, operations that can attain two goals:

1. "Keep the lights on"; that is, make sure that applications are available and performing well enough
2. Allow a more or less constant stream of updates that support business needs

When we want to slow updates down in the name of safety, what we really mean is that we are flying blind.

Agile Operations Needs Metrics

What obstacles are standing in the way of agile operations? Beyond cultural issues and an entrenched team mentality — both widely addressed by the devops community — one key obstacle is the lack of objective, shared, actionable metrics.

When we want to slow updates down in the name of safety, what we really mean is that we are flying blind. We are relying on people, not hard data, to guesstimate whether a given update will work, and we take corrective actions when things break. Actionable metrics allow constant updates to be deployed safely, reverted, or corrected when necessary. Actionable metrics essentially take the fear out of change.

Process and Tool Friction Stand in the Way

The lack of metrics is compounded by the friction incurred by most organizations when they decide to reliably track a metric they deem actionable. Friction is anything that turns "I would like to measure X" into "I won't measure X because it's too much effort." Friction is having to ask for permission to measure. Friction is a tool that treats all metrics as equally significant, thus drowning signals in noise.

Friction will turn into postponements and will invariably lead to a crucial metric being left untracked. Conversely, the metrics that *are* tracked will be the ones that require little work to track, regardless of their significance. The net result is monitoring of the extremes:

- Very high-level metrics (e.g., revenue, transactions) are tracked because of executive pressure.
- Very low-level metrics (e.g., CPU utilization, database cache hit ratio) are tracked because they come out of the box with most monitoring tools.
- Little else in between will be tracked because friction means hard work.

Table 1 sums up this simple metric taxonomy with relevant characteristics.

This is not to say that low-level or high-level metrics should be ignored, but rather that they provide an incomplete (and possibly out-of-date) picture of the user experience or quality of the application. High-level metrics can be used to assess changes *a posteriori* ("Does this new feature increase revenue?"), especially when the time frame is significantly longer than that of an agile iteration. Low-level metrics tend to highlight very obvious issues ("A drop in physical memory means one of the memory chips has failed") or very indirect ones ("The frequency of garbage collection has increased by 30%. Should the virtual machine be tuned again? Or the application code reviewed for memory leaks? Or has the amount of data that the application must process suddenly increased?").

The gap in between high-level and low-level metrics is where development teams have the most to contribute. They know the application intimately, so they know which specific metrics are actionable. They also know which ones to pay special attention to in the context of frequent releases, the natural byproduct of agile development. If we can eliminate that friction, developers as well as operations staff will contribute meaningful metrics.

Where does friction come from? From existing processes and tools; for example:

- From processes that require privileged access to track metrics, which in essence says: "I, the operator, don't trust you, the developer, to look at the application data, let alone interpret it."
- From tools that are not adequate for individuals across teams to maintain large numbers of custom metrics: "Tracking it is too much work. There is too much to look at."

IMPLEMENTING METRICS-DRIVEN DEVOPS[8]

Let's review what we have established so far:

- The rise of cloud computing ushered in the era of the API for both development and operations.
- Applications built on these APIs are more distributed and exhibit more complex and diverse behaviors.
- This complexity, along with the transition to agile development methodologies, creates a stronger-than-ever need for actionable metrics.

Table 1 — Application-Level Metrics Are the Sweet Spot for Devops

	Low Level	Application Level	High Level
Examples	CPU utilization, raw disk utilization	Application analytics, widgets computed by unit of time	Revenue
Audience	Devops, operations	Devops, development	Executives
Ease of collection	High	Variable	High
Specificity to the line of business	Low	High	High
Mean time to detect (compared to duration of agile iteration)	Short	Short	Long
Mean time to repair after detection	Long (in nonobvious cases)	Short	Long

- The truly actionable metrics are the ones that track application behavior, not very-high level metrics or very low-level system metrics.
- Yet actionable metrics are the least likely to exist because of the friction that traditional tools impose.

The next logical step is to examine what we need to do to eliminate friction.

Are My Metrics Actionable?

A metric is actionable if it provides, in and of itself, enough information to answer a diagnostic question on a feature or system. This is particularly the case if:

- **The metric has a clear definition;** there is clear documentation of what it measures, what its unit is, and how it is collected, aggregated, or sampled. It is surprisingly difficult to get clear documentation of even the most basic out-of-the-box metrics provided by standard monitoring tools.
- **The metric has a defined acceptable domain,** decided and documented *ahead of time.* As long as it lies in that domain, no action need be taken.
- **The diagnostic is unambiguous when the metric is outside of its acceptable domain.** The decision may depend on where the metric lies with respect to the acceptable domain, but it does not leave room for interpretation.

In addition, I find that the most actionable metrics are often composed of, or derived from, one or several direct measurements, which means that I often need to track composite or derived metrics such as ratios, rates of changes, or percentile distributions from raw metrics. For example:

- The traditional load average[9] in itself is not an actionable metric. A "load average" of five processes is only meaningful when compared to the number of CPUs in a time frame that is commensurate with the average duration of a task and a steady request arrival rate. We need to know that there are two CPUs, that we consider a 15-minute load average, and that each task is supposed to take five seconds. Only then can we decide to add two machines to absorb that "load."
- Often used in multi-tiered Web applications, the database query rate isn't a completely actionable metric either, as it can easily lead practitioners to mistake an increase in Web traffic with an anomaly in the code. We can, however, normalize the DB query volume by the amount of inbound requests and use the resulting ratio as a fully actionable metric.

Of course, the characterization of a metric as actionable isn't easy to make. There will always be contradictory or fuzzy metrics to interpret; there will always be imperfect metrics to choose from. Nevertheless, we should still strive for creating and tracking individually actionable metrics.

Actionable metrics are the least likely to exist because of the friction that traditional tools impose.

Who Is Responsible for Creating Actionable Metrics?

In short, everyone on the engineering team, whether part of development or operations.

To be more precise, the author of a change (in code or infrastructure) is responsible for creating actionable metrics that support his or her change. In terms of reducing friction, this means that no one asks for permission to collect new metrics, and the responsibility to do so is aligned with that of making code or infrastucture changes in the first place.

From a tools perspective, this implies:

- Engineering-wide access to metrics definitions and tracking.
- An approval-free workflow to create metrics.
- Beyond production environments, instrumentation of development and test environments, which is where most actionable metrics will be conceived.
- Graceful scaling to large numbers of metrics tracked, to avoid unnecessary friction. It is not unheard of to track tens of thousands of metrics in real time and be able to add more on a daily basis. A number of open source packages (e.g., Ganglia, Graphite, collectd) will provide a good starting point.

When Is the Right Time to Create Actionable Metrics?

In the context of agile development and operations, the right time to create actionable metrics is before a story is considered "done." It used to be that "done" meant unit-tested, integrated, and ready for deployment. In the devops context, I am adding "actionable metrics are created and understood by relevant teams" to the definition of "done" for each story. This, of course, also applies to stories that are bugs or operational tickets.

To postpone instrumentation until deployment time negatively impacts team collaboration in the long term. It forces the operationally minded members of the team to play catch-up and be purely reactive. Instead, all relevant team members should have full visibility into tracked metrics, graphs, or dashboards in all environments (development, testing, production).

Metrics are collected by tools, not by people.

Who Looks at Actionable Metrics?

Just as metrics are created by everyone on the engineering team, both developers and operators look at metrics. This is where devops particularly shines, with its blending of previously split roles and realignment of operational responsibilities.

This does not mean that everyone is spending their time watching graphs all day long, but that all team members can manipulate metrics without hindrance, that tools play the first-alerter role, and that upon alert, a large pool of informed and operationally aware team members can contribute to the response — including the developers who implemented the faulty feature in the first place.

From a tools perspective, this implies at a minimum:

- Access to metrics in all environments across relevant teams in development and operations
- "Discoverability" and "searchability" of metrics, which are most easily achieved by using one shared tool for all metrics definition and reporting

Other more sophisticated tools may also help solve the (hard) problem of filtering metrics to determine which warrant attention at a given time. Algorithmic techniques have been developed to that end (e.g., EventRank[10]), and this is an area of active development for modern tools.

How Are Actionable Metrics Reviewed? When?

Like any other part of an agile process, metrics are to be regularly reviewed and updated. A metric that is not deemed actionable anymore has to be fixed or discarded.

Metrics reviews are a team exercise that should be part of regularly scheduled retrospectives. They also absolutely have to happen upon resolution of operational incidents.

Who Collects Actionable Metrics?

Metrics are collected by tools, not by people.

Metrics that are collected manually don't count! Setting aside the wasteful exercise their tracking represents, experience proves that they will not be collected constantly and consistently and that they won't be transparently available to all relevant teams.

METRICS TO TELL STORIES

Strong standards for metrics upheld by all members of the technical team, all the way to the executive level, serve the same purpose that verification standards serve in investigative journalism or scientific inquiry. Standards allow us to isolate interesting data from the surrounding noise in order to build stories — stories that connect seemingly disconnected events and make them understandable so that we can learn and act.

In the end, metrics-driven devops is not about obsessing about metrics. It is about dispassionately guiding the decisions and actions of developers and operators in ways that benefit the business.

ENDNOTES

[1]For the seminal popularization of the theme, see: Carr, Nicholas. *The Big Switch: Rewiring the World, from Edison to Google*. Reprint edition. W.W. Norton & Company, 2009. *Cutter IT Journal* has run a number of articles on the topic, many of which can be found in the report *The Truth About Cloud Computing: Adoption Strategies, Security, and Reliability*. Cutter Consortium, 2009. If you are looking for the more visionary and entertaining end of the spectrum, see: Gibson, William. *Neuromancer*. Ace Books, 2004.

[2]Sending one SMS using Twilio from a smartphone costs $0.01. Sending one SMS directly from the same smartphone costs between $0.03 and $0.10.

[3]Allspaw, John, and Jesse Robbins. *Web Operations: Keeping the Data on Time*. O'Reilly Media, 2010.

[4]Allspaw and Robbins. See 3.

[5]In extreme cases (such as the Amazon homepage), a single page might trigger tens or hundreds of Web service calls.

[6]For an example of seemingly isolated issues that triggered system-wide errors affecting API customers, consider, for instance, Amazon's S3 outage in 2008 (http://status.aws.amazon.com/s3-20080720.html).

[7]Gat, Israel. "Agile Roots, Agile Operations, & Agile Clouds." *Agile Executive* (podcast), 19 June 2009 (http://theagileexecutive.com/2009/06/19/agileexec003).

[8]For an example of metrics-driven engineering, see: Brittain, Mike. "Metrics-Driven Engineering at Etsy." Slideshare (www.slideshare.net/mikebrittain/metrics-driven-engineering-at-etsy).

[9]In Unix, the traditional load average metric is an exponential moving average. For an in-depth explanation of load, see: Gunther, Neil J. "Unix Load Average Part 1: How It Works." TeamQuest, 2010 (www.teamquest.com/pdfs/whitepaper/ldavg1.pdf).

[10]Begnum, Kyrre, and Mark Burgess. "Improving Anomaly Detection Event Analysis Using the EventRank Algorithm." Oslo University College, 2007 (www.iu.hio.no/~kyrre/Research_files/begnum_aims-1.pdf).

Alexis Lê-Quôc cofounded Datadog to bring to market cloud-based tools that will help fellow developers and Web operators track, in real time, events, changes, and metrics that can affect their applications. He currently splits his time between caring for Datadog's data stack and thinking about ways to improve the product.

Prior to launching Datadog, Mr. Lê-Quôc was building infrastructure software and leading a team of IT operations staff as a Director of Operations for Wireless Generation, supporting several million teachers in the US. In practice, that meant doing everything from racking servers to obsessing over SQL queries and writing embedded code deployed into teachers' hands nationwide. In an earlier life, he spent time optimizing the performance of Web applications for Orange's 25 million mobile subscribers in France. Mr. Lê-Quôc can be reached at alq@datadoghq.com.

Reducing Software Release Pain by Releasing More Often

by Kief Morris

ONE PAINFUL RELEASE TOO MANY

Release 3.2 of our team's software product was the latest in a series of painful releases that had missed promised delivery dates. The pressure to get the thing out the door meant that our customers ran afoul of several bugs, forcing us to hurriedly squeeze out two bug fix releases in quick succession. The budget overruns and unhappy customers that resulted from this fiasco spurred our operations director and product director to call together everyone involved in the release process in order to work out a way to avoid repeats.

Several people felt we needed to schedule more time for testing and other release activities, but this would mean abandoning our ambition to release twice a year. We were already embarrassingly slow in responding to feature requests from our customers, so we couldn't accept an even longer release cycle.

Our operations director then put forward a counterintuitive suggestion — rather than releasing less often, why not try releasing *more* often?

Initially this seemed ridiculous. If our team wasn't capable of getting a release out in six months without running overschedule and overbudget, the idea of an even shorter release cycle seemed like a fantasy. But the operations director outlined his thinking, which was based on the concepts of *Continuous Delivery*.[1] Over the rest of the year, we made some radical changes to the way we worked and saw some very practical benefits to delivering one or more software release(s) every month.

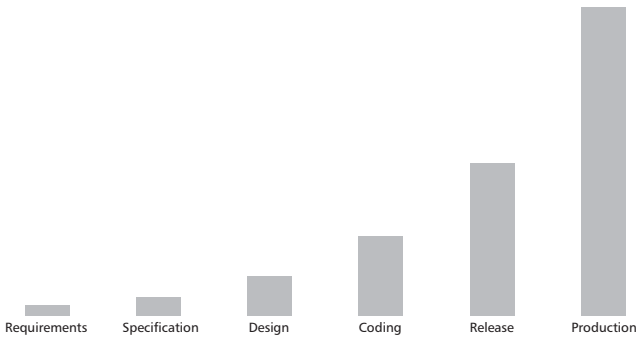


Figure 1 — Cost of change across the release cycle.

COST OF CHANGE OVER THE DEVELOPMENT LIFECYCLE

When the QA manager joined our team the previous year, he reminded us of the principle that the cost to fix a defect increases with each phase of the development lifecycle.[2] It's easier and cheaper to change a specification than to change code that's already been written. It's more expensive still to change software that's already in production, since it requires the considerable overhead of scheduling and managing a new release (see Figure 1). Keeping this principle in mind led us to involve the QAs from the start of planning for each release, instead of waiting until development was nearly complete, as we had been doing.

The Knowledge Gap

Another principle to consider, however, is that until software is in production and being used by customers, we don't know exactly what the requirements are, what the correct design is, or how best to implement them. Each phase of the software development lifecycle makes assumptions about what will work in later phases and, ultimately, what will work for customers. No matter how much time we spend in front of whiteboards, talking to focus groups, or having users play with prototypes, though, we never know for sure whether those assumptions are correct until people are using the software for real. This knowledge gap decreases with each phase of the release, in contrast to the increasing cost to fix (see Figure 2). As our understanding of the correct requirements and the best way to implement them improves, the cost of correcting errors and incorrect assumptions also increases.

Therefore, it would be nice if we could get small software changes into production use as quickly as possible so we can assess the results. As we learn more about what works and what doesn't, we can correct our

assumptions, and each new change can be specified, designed, and implemented with a more accurate understanding of customer needs (see Figure 3).

Reducing the Cost of Change

The problem with this approach is that it implies we will often be making changes after release, which, as we've seen, can be expensive. However, drilling into the reasons why changes are more expensive later in the release cycle suggests that shorter release cycles can be used to avoid this syndrome.

One reason for the increasing cost of change is that each phase builds on the previous phase, so any change requires going back and revising the work of previous phases. However, as our team began releasing smaller chunks of work, effectively one feature at a time, we discovered that the amount of requirements and design work that needed to be revised was much smaller than it had been for our large releases. Not only was there less planning work to redo, it was also simpler to understand the impact of any change than when we had dozens of pages of requirements and specifications to revise.

Reducing the Software Release Penalty

The penalty for fixing errors that make it into production is especially high because it means scheduling a new release, which has many overhead costs no matter how small the actual software change. But once we had releases coming out every few weeks, we found ways to automate and streamline the release process, which drove down those overhead costs. Furthermore, there was less pressure to introduce an unplanned "emergency" release to fix an issue if the next scheduled release was only two weeks away rather than six months away.

Accumulated Cost of Change

While these were promising discoveries, we realized that, for us, the biggest factor in the higher cost of fixes late in a release was still the amount of coding we'd done for the release. During the QA phase of our painful 3.2 release, we found that we'd incorrectly designed a key component at the start of the project. Since we'd had eight developers working for nearly four months building on top of the component as it was originally designed, we had nearly two and half person-years worth of code changes to untangle from the faulty design.

If we had put the software into production earlier, with the most minimal implementation of the overall feature set we were working on, we would have discovered the error with the component then. Consequently, none of the later code would have needed to be refactored.

The lesson we took away from this was that the cost of making a change during a software release is proportional to the size of the release. So with a short software release, the cost of a change may still be higher toward the end, but since much less code has changed, the cost will be much less than with a long release (see Figure 4).

Designing Software to Be Changed

As to the point that changing code is more expensive than changing a specification document, this is less true with modern software engineering practices and tools than it was 20 years ago. A software application is not like a building or bridge, which is completed and then remains relatively unchanged (other than repairs) during its lifetime. Software is continually improved, refactored, and rearchitected. Development of a software application doesn't end until nobody is using it any longer.

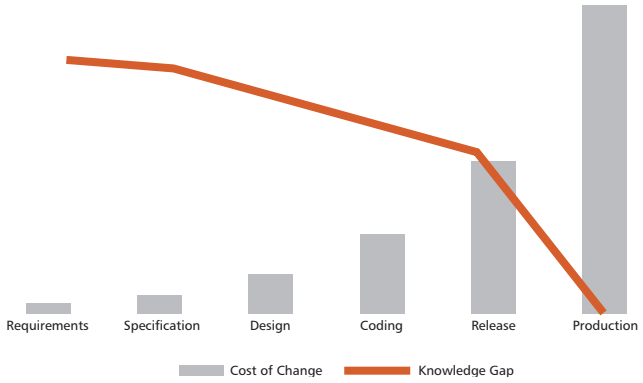


Figure 2 — Knowledge gap across the release cycle.

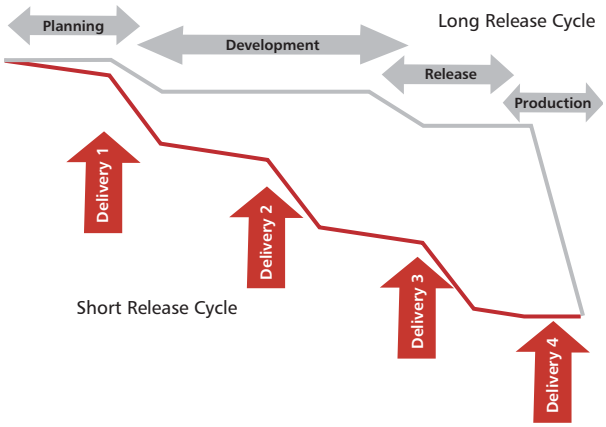


Figure 3 — Knowledge gap across long and short release cycles.

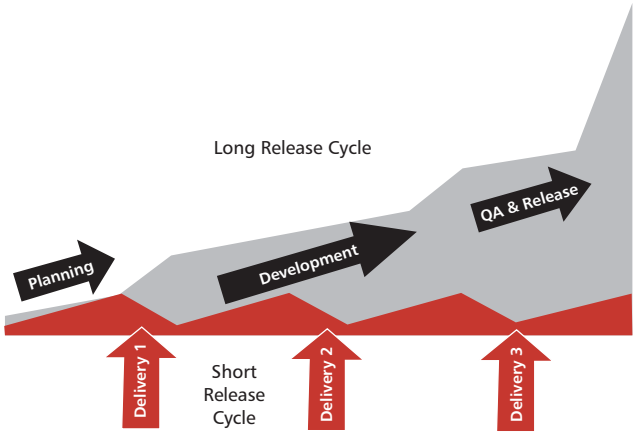


Figure 4 — Accumulated cost of change across long and short development cycles.

Today's developers understand this, so developing quality software means developing software that can be routinely and easily refactored throughout its lifetime. Practices such as continuous integration (CI) and test-driven development (TDD), as well as refactoring support in modern developer tools, make changing software much safer than it was in the past.

PRODUCTION CHANGES CAUSE PRODUCTION RISK

Our systems manager had a different objection to the idea of more frequent releases:

> It's fine for the development team to want their software released more often, but every release is a huge impact on my team. It's not just that we need to allocate resources for setting up and managing the testing infrastructure, although that's a big concern. But our pain carries on after you guys walk away and start your next project. Support calls go up as users get their heads around the changes, and it takes a while for a release to "bed in." The last release crashed several times a week until we tracked down that memory leak and got the bug fix release.

The authors of *The Visible Ops Handbook*[3] say that 80% of production system outages are due to a change, as opposed to hardware failures or other causes. In addition, 80% of the time needed to recover from an outage is spent figuring out what changed. A software release is essentially a large package of changes, each of which represents a risk to the stability of the production systems. At first glance, more frequent releases mean more frequent change, and thus constant risk of disruption.

Decreasing the Surface Area of Risk

However, like other organizations such as Flickr,[4] our team has found that smaller software releases introduce less risk to operations. Smaller releases contain a smaller change set, which presents a smaller surface area of risk. That is, a smaller change set means there are fewer things to go wrong *and* it's easier to find the cause of problems that do occur, both for systems administrators and developers.

An added benefit in terms of finding and fixing bugs is that developers will have the relevant areas of the code fresher in their minds. It's much easier to get your head back into code you were working on last week than code you wrote four months ago.

HABITS OF TEAMS WITH LONG RELEASE CYCLES

Before we changed to shorter releases, our team showed a number of antipatterns common to teams with long release cycles.

Releases Are Disruptive

Every software release was an intensive, disruptive, and costly process. After each release, we would update our extensive release and deployment documentation to avoid repeating mistakes we had made. Unfortunately, by the time the next release came around, some of our team had inevitably moved on to other projects or roles, other people were new to our team, and even those of us who were still there had forgotten some of the painful lessons.

Every Release Is Different

On top of that, each release had a large amount of change in it, and our hosting infrastructure hadn't stayed still either. The release would therefore inevitably involve some activities that hadn't been done before, such as migrating to a new application server, integrating with a different authentication system, or changing the monitoring to use a new set of health check "hooks" in the application.

People Are Involved Too Late

All too often, many of the people involved in releasing and supporting the software weren't brought into the picture until shortly before the release phase began. This didn't give them much time to fully understand the impact the release would have on them or to work out the best way to implement the changes needed. Worse, they often had insights that we could have incorporated into the system design and implementation if we had known about them early in the development cycle.

Table 1 — Comparison of Release Cycle Habits

Teams with Long Release Cycles	Teams with Short Release Cycles
Team membership changes between releases.	The same team members get many releases under their belts.
Production infrastructure has changed between releases.	Production infrastructure changes less between releases.
Infrastructure changes are managed using different processes (and often people) than those used for managing software releases.	The same people manage both infrastructure and software changes using coordinated processes.
Release team (e.g., QA, infrastructure, support) gets involved after development.	Everyone involved in a release is engaged before development begins.
There is a special "emergency release" process.	The normal release process is quick and robust enough that it is used for all releases, even bug fixes.
Many activities happen only during the release phase (e.g., UAT, performance testing, audits).	All release-proofing activities are done continuously throughout development.

Emergency Releases

Another common pattern of teams with long release cycles is a special process for unplanned releases. Normal releases are subjected to rigorous testing, user acceptance testing (UAT), performance testing, security audits, and the like. But when a critical issue is found that affects production users, the team is able to shove a patch, bug fix, or emergency release through to production, bypassing the normal process.

There may be a formal emergency release process that involves approval by a change advisory board, but this simply legitimizes the practice of skipping full release protocols and taking whatever shortcuts are needed to get the job done. When things go wrong in an unplanned release, another one is hurried through to fix it, and so on until things stabilize.

HABITS OF TEAMS WITH SHORT RELEASE CYCLES

When our team moved to shorter releases, we started by looking at our six-week release cycle and trying to find things that could be cut to streamline it. All we got from this was a release process that we didn't believe in and that was still too long. So we scrapped that approach and instead started from our emergency release process, thinking about what would need to happen to make it sustainable. By working this through, and looking at the way high-performing software organizations with very rapid release cycles work, we came up with a number of key habits (see Table 1).

Don't Wait for the Release Phase

A fundamental practice is to do as many of the release cycle activities as possible continuously throughout the development phase. The obvious example of this is testing. Although we tried to test new code and bug fixes as they were developed, in practice we were always behind in testing. This meant the amount of untested changes grew and grew, leaving us at the release phase with code that we didn't have much confidence in.

After discussing the issue with management and stakeholders across the business, we all agreed to invest more attention in continuously testing the software. We adopted a "stop the line" approach, whereby if we ended up with too many untested stories at any point in time, development work would stop until the testing backlog had been cleared. Other team members, including stakeholders from other parts of the company, would chip in to get the testing work done. Over time, this strategy made it clear that our team's ratio of QA to development staff was too low, so budgeting was agreed to fix the issue.

Automated Testing Buys More than Shorter Test Phases

Our full manual regression testing was too exhausting to repeat every few weeks, so we invested in improving our automated testing infrastructure. We moved from nightly builds to running tests on every commit and started tracking our test coverage.

TDD and automated testing are common practices that we had done to a certain extent, but not rigorously or thoroughly enough to give us confidence to cut down on the manual testing we did during the release phase. The developers struggled to write tests for their code as they went along. They wanted to believe tests could be written sometime later in big batches, but we never managed to find enough slack in our schedule to write comprehensive tests for code that was already written.

We visited several teams that are very disciplined about TDD and found that, for them, creating a comprehensive automated test suite was only a side benefit of TDD. These teams get clear productivity benefits from having tests that catch their mistakes as they develop, as opposed to writing tests long after their bugs have settled into the code.

A key benefit of frequent releases that we hadn't appreciated is simply that practice makes perfect.

Performance testing was a significant part of our release phase — very time-consuming to set up and conduct — so we also started working to automate it and to carry it out at least weekly throughout development. Not only does this mean we don't need to set time aside during release for performance testing, it also alerts us quickly if we make a change that slows the application down, so we can fix it right away.

CONTINUOUS COLLABORATION

The most effective teams we met had a high level of collaboration between development and operations. One smaller company had a single team with the developers and system administrators, including people who were proficient in both roles. Another had them in separate teams but working in the same room. The devops movement provided ideas and guidance that we used to streamline our release process.

Our team agreed on a rotation scheme, in which system administrators would rotate in to work closely with the development team, and developers would rotate out to work with the operations team. The developers who worked support rotations would handle maintenance and bug fixes, but they also spent time helping to make the application easier to deploy and support.

We also implemented a Kanban board[5] to track work items that crossed both development and operations. This got us talking more across the teams to get things done, and it made clear where things were falling between the cracks or just getting stuck.

Developing Software for Better Operations

Having system admins working with the development team turned out to have benefits for both groups. One outcome was that, over time, developers on support rotation integrated our application software with the tools our infrastructure team uses to manage servers and networks. Application logs were fed into log analysis tools, which allowed us to correlate events across our software, networking equipment, databases, and operating systems.

We also discovered that the core open source framework our application was built on had a comprehensive set of monitoring hooks, so we integrated these into the production monitoring system. This gave the developers much better insight into the inner workings of our application.

Jointly Designed Release and Deployment Process

We also built out a much smoother automated deployment system for our application, and we used this to deploy to all our environments, from production back through to the systems that ran automated tests on every commit. The systems team had begun using the Chef automated configuration tool for production systems, so working with them, we used it to configure our test systems as well.

The end result was that we were testing changes as our developers made them on systems configured and deployed nearly identically to our production servers. Since our deployment for release testing and final release was automated, this became much less of a pain point for the release phase.

Additionally, the system admin team was aware of the impact software changes would have for them as soon as those changes were developed. They would make any needed changes to their infrastructure ahead of time, and the software and infrastructure changes would be tested together during automated test runs many times every day until they were released to production.

CONCLUSION

Our team still has plenty to do before we are releasing as frequently and painlessly as we would like. But we are now releasing software at least every month and find that it does get easier as we go. A key benefit of frequent releases that we hadn't appreciated is simply that practice makes perfect. We were poor at managing releases in the past because we didn't do it very often, but now that we're doing it regularly, it's becoming routine. Each time we find a new way to improve the process for the next release.

The key changes we made — and that you can make — to implement shorter release cycles are:

- **Do one thing at a time.** Focus on thoroughly implementing and releasing a single feature at a time.
- **Minimize the release process.** Move as many activities as possible into the development phase (continuous testing, review, etc.) so the software is always "release ready."
- **Practice continuous collaboration.** Engage QA and operations teams throughout development, not just at the end.
- **Integrate change management.** Manage infrastructure the same way across development and production, ensuring changes are tested consistently.

ENDNOTES

[1]Humble, Jez, and David Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.

[2]Pressman, Roger S. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 1992.

[3]Behr, Kevin, Gene Kim, and George Spafford. *The Visible Ops Handbook: Implementing ITIL in 4 Practical and Auditable Steps*. IT Process Institute, 2005.

[4]Allspaw, John, and Paul Hammond. "10 Deploys per Day: Dev & Ops Cooperation at Flickr." Velocity, 2009 (www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr).

[5]Anderson, David J. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.

RECOMMENDED READING

In addition to the sources mentioned in the endnotes, I recommend Mary Poppendieck's essay "How Cadence Predicts Process," available at www.leanessays.com/2011/07/how-cadence-determines-process.html. The book *Web Operations: Keeping the Data on Time* (O'Reilly Media, 2010), edited by John Allspaw and Jesse Robbins, is an excellent resource for learning about current trends in highly effective infrastructure operations. Chapter 4, "Continuous Deployment" by Eric Ries, is particularly relevant to this article. Although our team didn't read it until after the events described in this article, its ideas influenced us via others who had.

As a Continuous Delivery Consultant for ThoughtWorks, Kief Morris helps software delivery organizations become more responsive to business needs. His focus is on working with organizations to improve collaboration across the end-to-end delivery cycle and to implement infrastructure that rapidly and reliably delivers changes into production. He has over 15 years' experience in software development and infrastructure. Mr. Morris can be reached at kief.morris@thoughtworks.com.

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