Devops: A Software Revolution in the Making?

August 2011
by Dominica DeGrandis

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

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

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

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

THE PROBLEM: LITTLE-PICTURE THINKING

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

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

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

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

THE SOLUTION: SYSTEMS THINKING

Imagine a company very much like Steve's — we'll call it "WeNeedHelp.com" — and then further imagine that its heads of Software Engineering and IT Operations agree to try a different approach, pursuing incremental evolutionary change (driven by customer demand) across their departments. Analyzing customer demand across functional boundaries would enable teams to step back and objectively look at their productivity from the "outside in." They would fully describe where their work originates, the types of work, the arrival rate, and the expectations from upstream and downstream customers.

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

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

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

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

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

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

THE BENEFITS OF SHIFTING TO A DEVOPS APPROACH

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

Increasing Revenue

Increasing revenue requires releasing our product or service to market faster, which requires ever-faster, ever-more-reliable software build and deployment methods. This begins with a thorough understanding of

the interdependencies among build artifacts, database schemas, environment configurations, and the infrastructure they sit on.

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

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

Driving Down Costs

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

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

Keeping Good People

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

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

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

LEADERSHIP: THE ESSENTIAL INGREDIENT OF DEVOPS

This all sounds good, yet the devops revolution can't happen without good leadership — and good leadership often seems to be in woefully short supply. Some managers appear incompetent in part due to

the structure of the corporation, with CEO fiduciary responsibility focused solely on short-term shareholder profit. Some management is slanted toward new projects (instead of much-needed maintenance work) because new projects can be written off as capital expenditures. Some management breeds contention between teams with annual merit ratings favoring certain teams over others. With developers scoring points for making changes happen and ops guys scoring points for keeping things stable, is it any wonder they find working together for the good of the company difficult? Good teamwork may help the company, but it provides few tangible points on an individual's annual merit review.

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

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

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

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

Management by Fear Breeds Distrust; Good Leadership Drives Out Fear

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

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

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

Making the shift to a devops systems thinking approach requires trust. Trust is essential for influencing change, and gaining trust takes time. Sociologists have learned that trust is event-driven and that small, frequent gestures or events enhance trust more than larger, grand gestures made only occasionally. Trust

is also asymmetrical in that a single act of bad faith can destroy it, and repairing the damage, if possible at all, will take many small acts completed with competence and delivered as promised.

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

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

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

IMPROVING THE QUALITY OF DECISION MAKING

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

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

Statistical process control (SPC) charts are a mechanism for the Study part of the Deming cycle and Seddon CPD (see sidebar).

SPC Charts

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

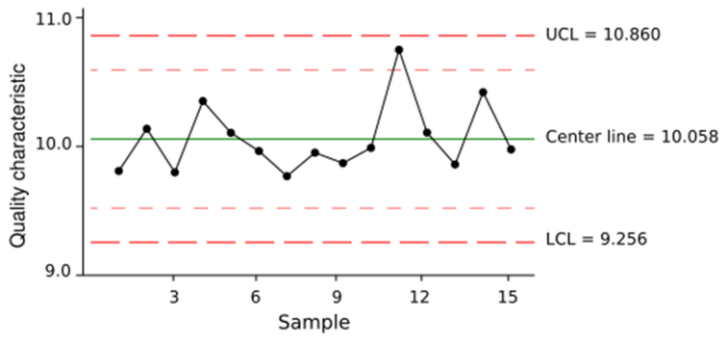

Figure A — Control chart showing common cause variation.

Removing Variability Results in Greater Predictability

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

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

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

PRECISION AND QUALITY DRIVE DOWN COST

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

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

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

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

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

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

SUMMING UP

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

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

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

ENDNOTES

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

[2]Perez, Pascal-Louis. "Continuous Deployment in an SEC-Regulated Environment – SLL Conf." Wealthfront Engineering, 25 May 2011 (http://eng.wealthfront.com/).

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

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

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

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

[7]Beck, Melinda. "The Sleepless Elite." *Wall Street Journal*, 5 April 2011 (http://online.wsj.com/article/SB10001424052748703712504576242701752957910.html).

[8]Deming. See 6.

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