DevOps Perspectives 5

Insights and Guidance from DevOps Leaders and Practitioners

ca technologies

Political Will Meets Administrative Won't: Using **DevOps** to Change Government IT Culture

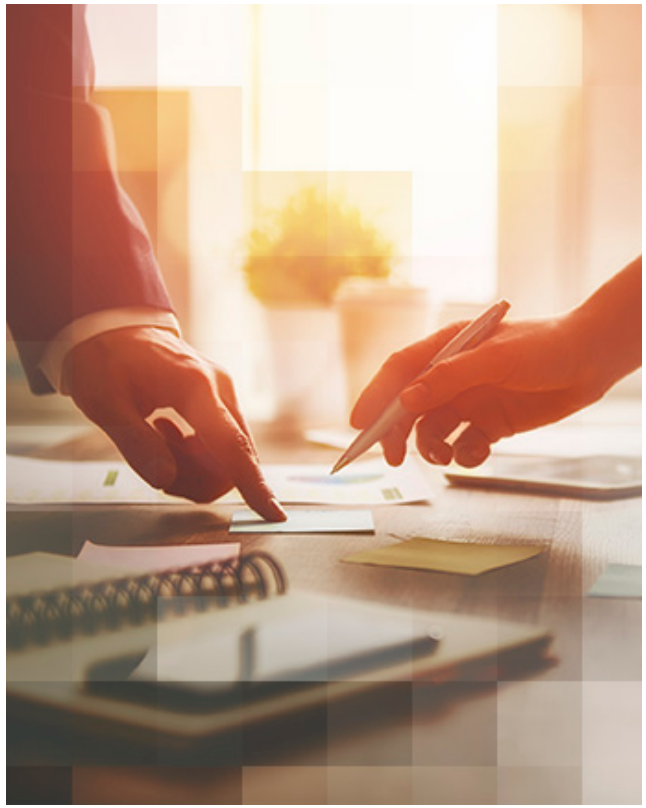
It's perhaps indicative of the growing footprint of the DevOps movement that the articles in this edition cover such a wide gamut of topics from mechanical sympathy to the networking team as a DevOps organisational entry point to some excellent case study examples of best practices, such as that of Live Nation's Ticketmaster operation.

Of particular note are two public sector use case exemplars—one from a UK local authority perspective where agile techniques are being used at Stockport Council to revolutionise development and introduce cost savings in an age of austerity, and one from the U.S. where the US Citizenship and Immigration Service is chasing the Chaos Monkey in its own attempts to inculcate a DevOps culture within government agency bureaucracy.

Both are great examples of the cultural and change management challenges that DevOps evangelists face. They're challenges found in the private sector, and, in the risk-averse environment of the public sector—where the political will meets the administrative won't—are all the more magnified and engrained.

As USCIS CIO Mark Schwartz puts it, "Bureaucracy is based on making a plan and getting everything down on paper, which really isn't a very effective way to oversee IT programmes."

But with organisations such as the UK's Gov.UK committed to the advancement of agile techniques and development methods in government IT, there's growing weight behind breaking down the existing practices and moving to a better approach. Demonstrable results such as those at Stockport, where the relationship between IT and government services has matured and transformed over the past year, shows that these are not just tech improvements but business process improvements as well.



Contents

DevOps—No Monkey Business in Growing a DevOps Culture in Government

Mark Schwartz

Back in 2014, Mark Schwartz, CIO of US Citizenship and Immigration Service (USCIS), made a bold statement about a monkey running amok. "If you read that Chaos Monkey has shut down Homeland Security, take it as a success," he quipped.

This particular simian is a script that sets out to cause deliberate havoc and try to screw up things in production, and which USCIS uses as part of its development work internally to ensure the robustness of systems before they go live.

Two years on, Schwartz is still a fan of the Chaos Monkey approach. "It's an interesting direction that I believe in more and more," he says. "You have to test systems in production. You can't just assume things will work when there's a problem. You need to really test and exercise."

That said, the idea of crashing Homeland Security systems in any way is one that sent shivers down the

"You have to test systems in production. You can't just assume things will work when there's a problem. You need to really test and exercise."

collective spine of the USCIS bureaucracy back in 2010, when Schwartz made the move from the private to the public sector. What he found when arriving in his new office was a situation common to government organisations and agencies around the world—a way of doing combined with a not-invented-here mindset that was holding back change and efficiency.

Schwartz recalls situations such as being told that it would take eight months to a year just to make a few changes to a web page, something that he could do in a couple of minutes, because of the need to follow the strictures of the software development lifecycle (SDLC) as laid out in Management Directive 102 (MD102).

MD102 cast a long shadow over Schwartz's work. It was cited as a reason for not doing something differently on multiple occasions, leading to him examine it closely. Rather surprisingly, perhaps, he describes it as "a beautiful document," into which a lot of work had been put to meet the needs of the then-embryonic, post-9/11 Department of Homeland Security, under whose auspices USCIS sits.

When MD102 is put into the context of the "total chaos" of bringing together 22 components of government overseen by 102 Congressional Committees, then, argues Schwartz, a different side to it can be seen. "It's been written by human beings in a government organisation trying to do their best at a difficult time," he argues. "This is not a faceless bureaucracy, its real people."

Flash forward to 2016 and that argument is still front and centre for Schwartz. He's clear in his reasoning about the suitability of the DevOps and continuous delivery model for government and can list the benefits:

- Easier change management in procurement, with everything being scripted and regression test suites allowing new contractors to come in and not have to wade through piles of documentation left by their predecessors
- Better metrics, because it's necessary to prove that every decision has been made objectively and catalogued
- Easier to meet compliance requirements, such as for the Federal Information Security Management Act (FISMA), by working during the development process, rather than having gatekeepers come in at the end and find that a system isn't compliant
- Enhanced security through continuous monitoring of vulnerabilities and feedback from developers during production
- Better risk management capabilities by dealing with issues as they emerge during development, not discovered after the system is complete
- Reduced expense and waste, an across-the-aisle issue that can unite policymakers

But organisations can only realise these benefits if there's a cultural shift toward doing things differently. In government terms, that involves finding new approaches without digging up the foundations that have been put in place to support what Schwartz calls a necessarily low-trust environment. Government will always be risk-averse. Worldwide, policymakers and change agents coming into the public sector have run into the situation of "political will meets administrative won't." It's a common theme.

Over the past few years, progress has been made, says Schwartz. "Within our agency, we've done a lot of cultural change. We have agile coaches around the place, but their role today is more about technical methodologies. We're trying to institute a shift toward a hands-on approach where everybody is involved in a quality check sort of role. I want everybody to be involved in creating products or checking results. So we have our agile coaches pair with developers and get involved in the creation process. It's the same with Quality Assurance and Test Oversight."

"We're also encouraging our middle management layers to remember their technology roles. We have a new workforce coming into IT now which is made up of very talented technologists. They tend to respect management more if they, too, are hands-on. We have a lot of people who used to love coding—and still do—but they don't do it as much, so they feel disconnected. Equally, you get these brilliant young technologists coming in and you can have management feel that their coding skills aren't as strong as they used to be, but we encourage them to do the brilliant stuff, because they can probably still do it."

It's also about transitioning to a new approach to working, adds Schwartz. "The transition that I would like us to make is toward a learning-and-feedback approach, rather than controlling things though upfront plans. Bureaucracy is based on making a plan and getting everything down on paper, which really isn't a very effective way to oversee IT programmes. We need fewer heavyweight, upfront plans and more lightweight check-ins on progress."

"I want to move the focus onto measuring outcomes. I don't want to start with a big list of requirements, but with outcome metrics that we want to optimise. So where there's a need for manual intervention in a process, the outcome we want should be to speed up the ability to do that intervention. So, we would say that there is a product owner for that who will just prioritise the things that will impact that outcome. What we need to do is to show the oversight bodies that this actually gives them more control than they have with the huge requirements documents."

He adds, "Using DevOps and continuous delivery, we can give the overseers what they need, but there's just a different way of thinking about it. The planned-delivery approach never really worked. The reality would never actually match the plan that was on paper. With DevOps, we can control our projects better and provide the best possible outcomes. We can accomplish more."

But in government it all takes time. Circling back to MD102 and 6 years after it first came into Schwartz's life, it remains a work in progress. There is now agreement to pilot new approaches as MD102 substitutes with five programmes in Homeland Security at work on these. "These are intended to show overseers that this will give them more control, not less," concludes Schwartz. "Based on feedback, we will then rewrite MD102."

"I want to move the focus onto measuring outcomes. I don't want to start with a big list of requirements, but with outcome metrics that we want to optimise."

Golden Ticket to a **DevOps** Cultural Revolution

Jody Mulkey

A 230-percent increase in the number of developers but only a 12-percent increase in the number of operations personnel—that's a situation that's going to bring pressure to bear on any organisation. So it was at Live Nation's Ticketmaster business in 2014, where incoming CTO Jody Mulkey found himself faced with just such a dynamic.

His response to this over the past two years has been to inculcate a new people-centric culture of empathy, empowerment and metrics to create a new development model for a firm that is in fact one of the oldest SaaS companies in the world.

Mulkey's argument is simple: DevOps isn't about technology; it's a mindset that needs to be about working together as human beings. "If you put great people together, then they will do great things," he posits. "We are not in the operations business; we are in the software development business."

He adds, "The only way we make money is by meeting market needs and the only way we do that is by changing our software. We serve the customer through software products."

Inevitably, as happens so often when introducing DevOps into any organisation, there was an inherent resistance to change within Ticketmaster, with some people whispering that Mulkey was "crazy, he's going to let developers touch the software."

Two years on, Mulkey can admit that change is hard and that this has been a challenging process for longer-serving members of the Ticketmaster team, some of whom have over 10 years of service under their belts. That duration of tenure itself brings its own challenges when trying to introduce new working methods and models.

"What's interesting of course is that Ticketmaster is a market leader in its field and we have quite a few folks that have been here a long time," says Mulkey. "So, when I say to some people. 'We need to go

"DevOps isn't about technology; it's a mindset and that mindset needs to be about working together as human beings. If you put great people together, then they will do great things."

faster,' they look at me and say, 'Why? We're already number one.'"

"You can try to make the change through logic, but logic doesn't create change—emotion does. As a global market leader, there's always a handful or more of smaller companies all trying to take a bit of our business away. I need to show the team who those companies are and how fast they're growing so that I can stir them up a bit and get them to understand that this great place is number one today, but we always need to act like we're number two. We need the fighting spirit, and the emotion drives that."

Mulkey's approach is built on three prongs:

Empathy—This applies both internally, "You can't make material change without engaging hearts and minds" and externally, so that Ticketmaster employees understand the importance of enabling a great customer experience. To that end, staff go out and work with clients in the field. Mulkey jokes that if a developer is present at a One Direction concert when 5,000 people want to get in and the ticket scanner breaks, it gives them a new sense of urgency.

Empowerment—This goes back to the great people wanting to work with great people idea. "Game attracts game," says Mulkey. "We have 100 development teams globally and they want to run their own projects. Today, 100 percent of the 73 teams we have in the U.S. push their own code all the way to production."

> "And we're continuously hiring, and hiring for the mindset rather than the skill set. It is getting easier. Great people want to work with great people and we have a lot of great people."

Metrics—Business metrics are what really matter, argues Mulkey. This wasn't something that was built into the Ticketmaster culture, with none of the product managers talking about money. Now, there are revenue graphs on all the operational dashboards, while the focus has shifted from output to outcomes. "We had been a 'ship the feature' company rather than a 'solve the problem' one," recalls Mulkey.

A longer-running resistance came from the ops side of DevOps, something Mulkey attributes to a changing locus of leadership across the years. Back in the dot-com era of the early 2000s, that locus sat with operations, because even though the software was relatively simple, running it at scale across thousands of servers was a challenge.

That locus then shifted, first to software developers in the Web 2.0 era, and now to design professionals. "Ops had all the power in the company and that has shifted," says Mulkey. "But we are not an operations company, we are a software development, SaaS company."

That said, things level out over time, he adds. Inevitably, in any cultural evolution, there will be those who just won't make the grade in the new regime. "If people signed up to do something and that changes, even if the new thing they're being asked to do is better, they may no longer want to be here," Mulkey reasons. "Any organization has turnover and we believe a healthy amount of self-selected attrition can make our team stronger and our products better."

Over time, new people will also come in to an organisation who are on the same wavelength as the new culture. It's a never-ending journey, says Mulkey, one that's exponential in nature. "There's a non-linear path to where we want to go," he says. "We're on the way, maybe 50 percent of the way there. We're using a big move to public cloud as a manifestation of the change."

"And we're continuously hiring, and hiring for the mindset rather than the skill set. It is getting easier. Great people want to work with great people and we have a lot of great people."

In fact, he concludes, only two people he's interviewed haven't ended up working for Ticketmaster—and one of those had a plum offer from Uber and is now a multi-millionaire. "He's not regretting his choice," laughs Mulkey.

"You can't make material change without engaging hearts and minds."

Sympathy Pangs

Martin Thompson

It was renowned motor racing driver Jackie Stewart who declared, "You don't have to be an engineer to be a racing driver, but you do have to have Mechanical Sympathy."

As a three-time Formula 1 champion, there's clearly something to his thesis, which boils down to an argument that understanding the technology which goes into building the race car makes you a better driver.

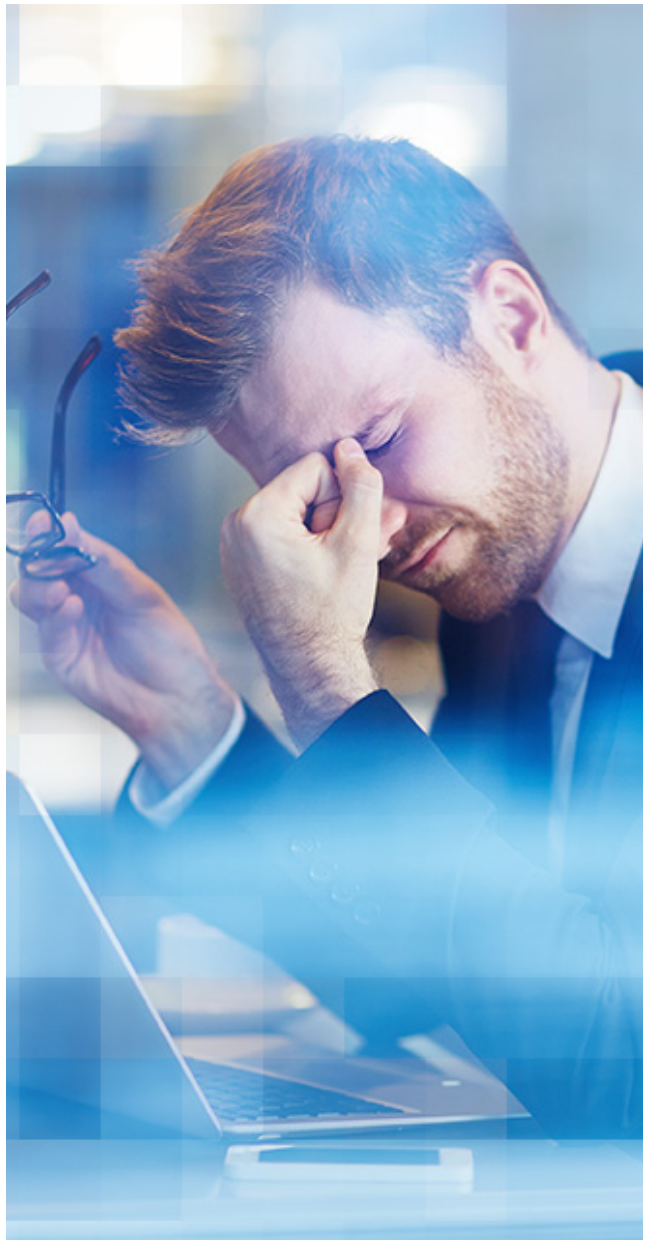
Martin Thompson, proprietor of the blog "Mechanical Sympathy" and founder of the LMAX Disruptor open source project, took that idea and applied it to the software development world, positing that a better understanding of hardware is essential to the creation of the software that is going to run on it.

A lack of awareness of this synergy leads to scenarios where the software of today doesn't feel as though it's running any faster than DOS-based applications from 20+ years ago. "This is a 20–30-year-old problem, there is a collective amnesia in our industry that means that we don't learn from the past," he declares. "We should be coming into jobs and working with people who've done this for a long time and know how to do things well, and learning from them. In any other discipline, that would be the case. But the problem is that IT is so new."

"That's particularly true of the software development industry. How many of us could say that our parents were in the software industry? Who can say they are a second- or third-generation programmer? Not many. Really, we're living in an era of software alchemy, where we are like 16th-century physicists, making things up as we go along. Fundamentally, there is science underpinning this. We have to have hypotheses for how things work."

For his part, Thompson's developer history dates back to the ZX80s, Spectrums, BBC Micros of the 1980s, when it was necessary to deal directly with memory when coding. This produced an acute awareness of how much memory you had to play with and how it worked. Developers had to understand this or they couldn't create performing systems.

The same is true today, he suggests. "It's about making sure that things are correct," he says. "When you're storing data in a database, you can be running on various file systems, so then knowing you are not getting into a mess is good. Have you chosen the right system for storage? Do you know how it works and which choices to make to get the best result?"



There's a cultural mind shift that needs to take place, argues Thompson. "We have had a view of, 'let's throw hardware at a problem' and that has worked in some cases, but not in others." he says. "If you look at the world from the '50s and '60s through to the 2000s, processors kept getting faster, memory got faster, disks evolved. Then CPUs stopped getting faster, disks didn't get faster until SSDs came along. We've flipped over."

"As time progresses however, CPUs are not going to get significantly faster, so the question becomes how we split up work in such a way that it is going to be truly scalable. If you want to scale things, then there has to be a cost model. If you're gong to add more resource, then you should be getting economies of scale. Most software today is not written with that in mind." Some basic understandings are simply overlooked today, adds Thompson. "We don't measure things until we hit problems," he states. "We choose to wait until the last possible responsible moment, but that responsible moment often comes too late. Individuals come across unknown unknowns, but these are often things that are well known and understood across the industry. For example, TCP is a very well-understood protocol whose issues are well known to some parts of the community. But how you work with it is only well known to a limited part of the DevOps community. We tend to follow the latest, coolest trend. We want instant gratification, but some stuff requires a bit of work."

Another case in point is the enthusiasm for working with microservices. "Everyone is crazy about microservices, but who's talking about how they are going to communicate with one another?" asks Thompson. "That's all about protocols of interaction. How do you design for cohesion? I'm not seeing anywhere near enough focus or discipline around this. You ask some people about cohesion and they can come up with some kind of definition, but then you look at their code, it's not there."

So how will this change? And what is the mindset that needs to come into play? "Cost is always an interesting driver of change," suggests Thompson. "Are you using IT efficiently? Are you getting the best out of it? Are you being cost-effective? To operate in any environment without being aware of the cost is not professional. If you measure ROI across various disciplines, we in IT rank as one of the worst and most inefficient across any other domain."

But the likelihood is that this is a long journey and one that's going to be a bumpy ride with lots of back and forth on the way. "We've had a software crisis since the 1960s which the agile movement helped to address, but things have gone too far the other way now," Thompson suggests. "Today, big upfront design is seen as a bad thing, so there is virtually none done. But not doing any upfront design is just as bad as doing too much. We will move forward and we will swing from extreme to extreme and we will learn from that.

"Different parts of the DevOps community will move at different rates," he concludes. "I do see elements of it where there are individuals who are talented and who are doing the right thing. It's whether or not this becomes a wider culture. We will have to do it. It's just a case of when we do it."

"We don't measure things until we hit problems. We choose to wait until the last possible responsible moment, but that responsible moment often comes too late."

Capitalization, Agile and Why You Need to Care

Dan Greening

In many companies, Agile software development is misunderstood and misreported, causing taxation increases, higher volatility in profit and loss (P&L) statements and manual tracking of programmer hours. I claim Scrum teams create production cost data that are more verifiable, better documented and more closely aligned with known customer value than most waterfall implementations. Better reporting can mean significant tax savings and greater investor interest. Agile companies should change their financial reporting practices to exploit Scrum's advantages. It might not be easy, but it can be done.

> "Misunderstandings in how to track and report agile project costs have cost many companies millions of dollars in improper taxation."

Software development is an investment in the long-term future. We spend money upfront on engineer salaries and then (hopefully) profit later from cost savings or revenue. If we invest wisely—converting cash (one type of asset) into software (another type of asset)—the company's value should go up. Tax authorities and investors rely on financial reports to understand the value of a company. How we report development expenses matters.

First, let's define capitalization and expensing. Capitalization means spreading investment costs (sometimes called capital investments or capital expenses) over a long-term asset's life of returning value. Capitalization is used in tax filings and financial reports (such as P&L reports). Capital investments become part of the declared assets of the company. Expensing means taking the hit of a cost immediately as an operational expense that returns short-term or no value. A company that expenses all of its software development has a hard time arguing that its software is part of its long-term value.

"Companies can gain tax advantages by capitalizing software development: by deferring costs they typically offset more taxable revenue and gain more interest income."

It's easy to make damaging mistakes when classifying software costs. Some companies incorrectly treat all software investment as an operational expense, which could provide an opportunity for impropriety. Classifying software investments as operational expenses usually just results in the company overpaying taxes and understating its value, which in turn would depress its stock price and reduce its borrowing power.

Agilists Should Understand Capitalization

Agilists should learn proper capitalization and teach their colleagues. Misunderstandings in how to track and report agile project costs have cost many companies millions of dollars in improper taxation. Poor capitalization rules create choppy income statements for agile companies, making them look poorly managed. So-called conservative waterfall processes can rarely track which design efforts or management tasks led to which features, while agile methods can. Yet accountants typically do not understand how to properly track and report labor in agile projects.

On the positive side, Scrum Masters and agile department heads who understand capitalization can generate millions in tax savings because good agile practices can enable more verifiable capitalization, and because spreading investment costs over time often reduces the overall tax and helps find earlier funding to hire additional engineers.

Scrum Masters, often more than anyone else in the company, can correctly classify work as long-term investment or short-term expense, and usually have all the data needed to defend their classifications with financial staff and external auditors.

Scrum Masters promote processes that more reliably align actual team behavior with documented goals. Scrum techniques have an adaptive statistical basis, backed by experimentation, which is absent in classic project management techniques. In my experience, auditors can trust agile-based reporting more than waterfall-based actuals.

If, as a Scrum Master, you want to tackle this opportunity, labor classification will then likely become your responsibility, along with other Scrum Masters in the company. You will likely become, by necessity, an expert on the topics of software capitalization, depreciation and impairment. Welcome to the world of finance.

Proper Classification Creates a Bright Future

Tax authorities and investors use operational expense and capital expense concepts to make better decisions. They usually want companies to invest in the long term, so they let companies spread investment costs over time to offset revenues roughly in parallel as the investment earns money.

Software work can provide short-term value (all ROI in under a year) or long-term value (ROI over a multiyear period). Here's a short-term example: a contract software company might create a website for a customer, get paid for it and retain no further rights. In this case, we say development cost is an operational expense (opex).

Public companies usually must report yearly and quarterly profit to shareholders and tax authorities. Computing the profit seems easy:

profit = revenue – expense

Here's a long-term example: a toy retailer builds a website to sell its toys. Years after it built the website, the long-completed work keeps generating revenue. In this case, we say development cost is capital expense, a long-term investment. Computing the total profit, ex post facto, is easy:

total_profit =
revenue(year_1) + … + revenue(year_n) – investment

Every year, shareholders and tax authorities expect a financial report; their first concern might be to ask, "What were our profits last year? If we have a long-term software project that gains no revenue in its first year, and if we have to treat it as an operational expense, we might need to post a loss. Fearful shareholders might sell shares of our company. Maybe we don't have to pay taxes this year, great. But next year, we might have no development expense and a lot of revenues from our toy retail site, in some jurisdictions taxed in full." If we had to treat development efforts this way, it would discourage us from investing long-term.

Wisely, tax authorities and accounting groups let us spread these capital expenses over time using a system called depreciation. Most depreciation schedules spread a capital expense evenly over the expected lifetime of the software; if the toy retail site we develop will likely stay in use over a 5-year period, we can expense 20 percent of the development cost the first year after deployment and 20 percent each year after through the fifth year. (Contact an accountant for more information on depreciation schedules, which can vary depending on the expected lifetime of an asset.)

An investment might not be usable right away. Since we don't immediately gain revenue from it, we can usually defer depreciation until it goes into use. The accounting shorthand for this time before deployment is the capitalization period. (Capitalization benefits continue after depreciation starts, by the way.) If we remove features in our website software or stop using it entirely (possibly because we replaced it) either before or after deployment, we "impair" our old investment and then have to immediately expense all remaining costs.

Companies can gain tax advantages by capitalizing software development—deferring costs they typically offset more taxable revenue and gain more interest income. Departments also gain some advantages in hiring. When a department can defer software investment costs, it often can spend that deferred cost on employee salaries (hiring more people, providing raises, etc.)

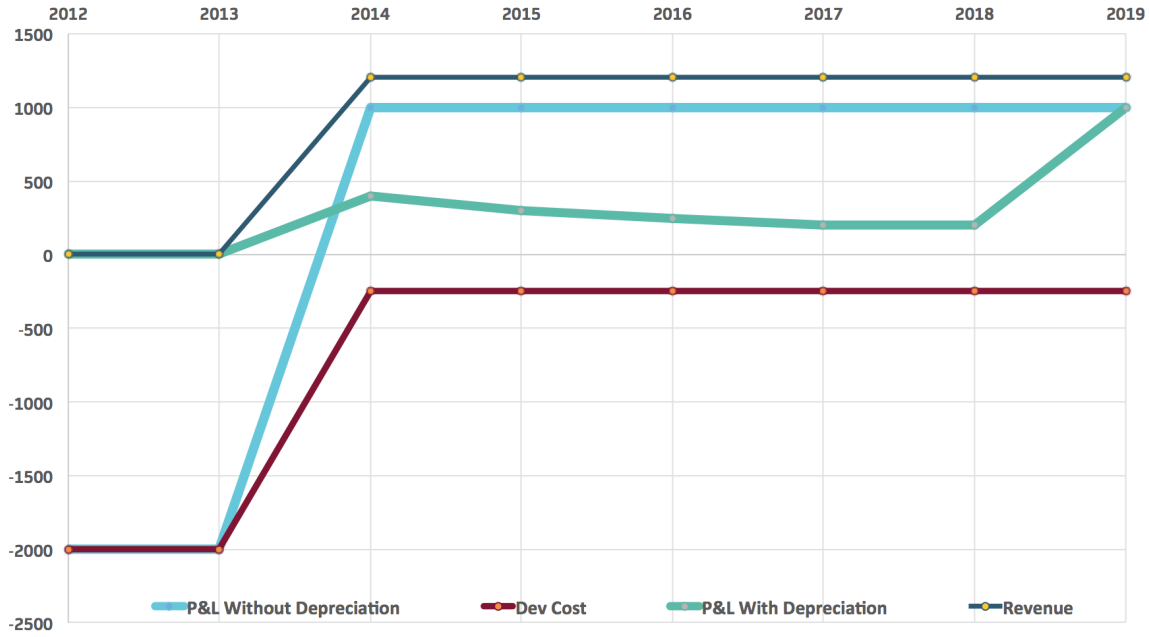
Profit and Loss, With and Without Depreciation

The graph illustrates how P&L can be affected by depreciation. The numbers shown are in thousands of dollars. As is typical for software projects, the major costs (dev cost) occur at the beginning of the project: $2 million in 2012 and $2 million in 2013. In 2014 and beyond, the costs will be $200,000 per year which is the cost of adding features to the software. The project doesn't start earning revenue or cost savings until it is deployed in 2014, and at that point it may earn $1.2 million yearly. If project costs are not depreciated but expensed immediately, the blue line tells the P&L story: Huge losses in the first two years, then enormous profits in subsequent years.

When costs are depreciated, the green line tells a different P&L story. No costs are taken from P&L until the software is put into use, and when it is, we compute our profit by depreciating the cost over a five-year window.

Why is this important? Governments typically tax P&L on positive amounts, and make it difficult to use losses to reduce future year taxable P&L (unless you can justify your choice with depreciation).

Finance people often over-expense by treating all software expenses as operational expenses, claiming this is somehow conservative behavior. It isn't. If you are investing in the long term, placing software investments in a short-term expense class will make your company look volatile—that's irresponsible to your shareholders. It can generate higher tax liability, which is both irresponsible to your company and out of line with the goals of your host country, which would want you to invest long-term.



A company with high profits can offset a product development department's production losses. This would avoid the tax problem, but it doesn't avoid the poor-planning problem. In my experience, executives pay attention to departmental profits and losses and drive headcount from that. Who among us hasn't seen boom-and-bust cycles of hiring and firing in large software concerns? In part, this headcount volatility is caused by failure to properly recognize software as an investment.

Finally, if agile software projects are expensed and waterfall projects are not, it would essentially doom any long-term enterprise adoption of agile practices. If waterfall projects can hire more employees but agile projects can't, guess what methodology managers will promote?

Recommended Accounting Practices Ignore Agile

Accounting practices are not completely dictated by tax and securities law. Instead, the U.S. Financial Accounting Standards Board (FASB) interprets these laws to produce generally accepted accounting practices (GAAP). FASB guidelines for internal use software are in [ASC 350-40], and for externally sold software are in [ASC 985-20]. Their treatments are roughly equivalent for this discussion. The International Accounting Standards Board (IASB) produces the International Financial Reporting Standards (IFRS). FASB and IASB provide guidance on how to interpret law. Their recommendations, which were written before agile practices were popular, show how to classify work using waterfall examples

Waterfall capitalization timeline

Misinformed people believe FASB and IASB guidelines force agile projects into a waterfall world of engineer time tracking, with RUP-like phases of analysis, prototyping, development, packaging and maintenance. Instead, the guidelines state that market analysis prior to development is expensed, prototyping prior to a decision to invest is expensed, development for long-term value is capitalized, packaging for shipment is capitalized and maintenance (fixing bugs) is expensed. The figure above shows capitalized items in green.

Auditors recognize that FASB and IASB guidance cannot be routinely applied to new situations. What tax authorities and auditors look for is conformance to law and its spirit, consistent application and full transparency. We can give them all that; but because agile practice is new, we must understand the law and its motivations, document our capitalization policies and practices, track project work consistently and be perfectly transparent. This aligns well with agile principles.

However, if you ignore the law and its motivations, inconsistently track work or fail to document processes clearly, you risk the wrath of tax authorities and investors. Adverse audit findings and the resulting submission of corrected financial reports can cause tax authorities and investors to lose trust in the company, which would subject it to higher scrutiny and a lower stock price.

Finance departments are justifiably conservative in their approach. If your finance department doesn't like how you do things, they could:

- Force engineers to track hours (degrading their creativity and productivity with mind-numbing work-tracking)
- Undercapitalize software development (leaving huge sums on the table)
- Reclassify past expenses (raising investor questions about the stability of the company)

It's easy to make multimillion dollar mistakes in this area. Because the vast majority of companies make capitalization mistakes that increase tax receipts, the authorities don't complain. And because agile software practices are arcane to investors, they don't complain either. But they should.

If you involve at least one person that has a moderately good understanding of three fields—finance, engineering and process—you can dramatically improve your bottom line. Since the returns are so high, it may be worthwhile to hire a consultant to help get it right.

How to: Financial Reporting in the Agile Frontier

Until FASB and IASB guidelines are revised to explicitly discuss agile examples, responsible agilists must work directly with their own corporate finance departments and auditors to craft acceptable capitalization processes.

First, establish a clear and consistent bright line demarking when your company could start capitalizing work. ASC 350-40 states that cost capitalization can begin when all work in the preliminary project phase is complete, when management commits to funding it and when it is probable that the project will be completed, and used. Capitalization begins when you move from what to how you will design and develop the software asset.

In most cases, capitalization should begin when the whole production team assembles for its first sprint. Your company should complete an initial market exploration and architectural design before it invests in a full team of designers, engineers and testers. However, if a research team runs a feasibility spike sprint to determine which architecture to use or whether the market warrants further sprints before it can create something that could provide long-term value, you are likely in a preliminary project phase, and your costs should be treated as operational expenses.

Once your company has committed funding to a project likely to be completed and used, you can start capitalizing the work. All work critical to designing, creating, testing and deploying the asset should be capitalized, including engineers, testers, user experience designers, product management, project management and Scrum Masters.

Second, establish whether the entire or only part of the project should be capitalized. In many cases after the preliminary project phase, the entire project cost can be capitalized. This happens when a significant percentage of the work (we felt 95 percent was sufficient) should be classified as capital expense. However, some common activities must be expensed.

If any of the following apply to you, you may have a mixed-mode project:

- Your team is fixing regression bugs in a released product while developing new features.
- Your team is creating a product for international release and localizing the product for multiple languages.

- Your team (not just its software) manually converts data from one form to another.
- Your team helps train people to use the software.
- Your team participates in operations activities beyond deployment, such as monitoring, reporting, backup, machine configuration.
- Your team performs routine Sarbanes-Oxley (SOX) or security reviews [15 USC 7211].
- Your team refactors code unlikely to be relevant to new functionality (you probably shouldn't do this anyway).
- Your team modifies software to support individual customers.

Whether these items or others should be expensed or capitalized will depend on your finance department and technical accounting advisors.

How to: classify mixed-mode projects

If you have mixed-mode projects, establish a way to apportion labor to operational or capital expenses. If you have strong Scrum practices within your organization, you can likely defensibly use proportional allocation of estimation points (also called story points) for each team. If each team has a different point scale, it can be accommodated. For a quarter, sum the points completed by the team then divide it by the total cost of the team (including product owner, Scrum Master, team members and the appropriate percentage of part-time contributor salaries). You will now have the cost per point.

ID	Description	Estimate	Cap?
1	Add internal language capability	8	Y
2	Fix regression bug in English-language version	5	N
3	Localize for Spain, France, Germany	13	N
4	Customize software for Acme Corp client	3	N
5	Restructure site with better graphics, information flow	13	Y
6	Fix bug that "export" never worked on Mac OS X	8	Y
7	Implement import function	13	Y

In this example, the team completed 63 estimation points in its 4-week sprint, and could capitalize those 42 points. If the total team cost (the total salaries for the team) for those 4 weeks was $112,000, then the capital expense was $112,000 × (42 ÷ 63) = $75,000.

If product owners write product backlog items in a ritualized story form, it can be easier to determine whether it is a capital or operational expense. My preferred product backlog item story form, a variation of a form promoted by Chris Matts and Dan North [North 2006], helps clarify most classification work. It looks like this:

As a <stakeholder>,
I can <perform an action>,
so our company <receives business value>

Acceptance tests:
<acceptance test 1>
<acceptance test 2>

In this format, you substitute specific values for <stakeholder>, <perform an action>, <receives business value> and <acceptance test …>. The stakeholders are never the team, but they can be anyone else consuming the product: a user, a customer, a systems operator, a business analyst or an administrator. If you aren't serving someone outside the team, it isn't really a user story. <Perform an action> should be something that the stakeholder can do that they couldn't do before the product backlog item was completed. <Receives business value> is a phrase usually articulated with the developing company in mind: Why are we building this? Will we get more users? Will users pay more for the product? We will gain a competitive edge or match a competitor's features? Will we save operations costs? On rare occasions, the <stakeholder> can be a future developer; this accounts for focused efforts to reduce technical debt, however; make sure the acceptance tests confirm that future developers benefit.

Finally, we have acceptance tests. I counsel teams that acceptance tests should be written so stakeholders (usually non developers) can verify that the work was done, ideally in the sprint review. An acceptance test written for a team member to verify is not really an acceptance test.

Here's an example:

As a systems operator, I can monitor the current load on the system, so the company can add machines if the load approaches the point where new users will be denied access.

Acceptance tests:

From the administration screen, a systems operator can easily find the load.

If the load is in the green area, at least 50-precent more users can be added to the system without concern. If the load is in the yellow area, at least 20-percent more users can be added to the system without concern. When the load is in the red area, additional machines can be added to bring the load back to the yellow or green region.

This story should be capitalized because it adds functionality not previously available, even though it serves a stakeholder inside the company. This subtlety is sometimes missed, but is clear from thoughtful reading of [ASC 350-40], which contains the concept in its title "Internal Use Software." Because most cloud computing and website development projects run in the developer company's machines, they are characterized as internal use software.

This format not only serves well for financial classification, but also has benefits in helping the team understand the context of its work.

What about tracking hours?

Whenever capitalization comes up, someone usually suggests that one just track programmer hours. This is a mistake, not only because it disrupts agile behavior, but because measurement is likely inaccurate and not as verifiable.

Hour-by-hour financial monitoring slows down software development. Software development is creative work and interruptions to track hours disturb the creative process. If we enforced hour-by-hour tracking with engineers, we would pull developers from their Zen state of thinking about the stakeholder, stakeholder actions, the acceptance tests and the code into a self-conscious state of thinking about what they did in the last hour.

So, to avoid the disruption, companies almost always simply ask engineers to fill out time cards at the end of the week, at best. By this time, the work they've done is lost in the fog of the past. In my experience, their weekly reporting is quite inaccurate.

Auditors support accounting practices that provide honest transparency while maintaining high productivity. Those auditors I've encountered acknowledge that hour-by-hour tracking is problematic. When I have suggested that proportionally allocating actual cost by story points would provide honest transparency, they have, at first, cautiously agreed.

In doing so, I make an assertion that estimated effort is highly correlated with actual work time. This assertion is defensible. The Scrum framework is designed to help teams drive toward high correlation between estimation points and actual time. Scrum provides better forecasting accuracy than waterfall, and teams that embrace Scrum principles examine their estimation points and outcomes, trying to ensure that their sprint forecasts are roughly met by the sprint result.

Auditors become enthusiastic supporters of this approach when they see the effect. When we track product backlog items, estimation points and completion dates (sprint end dates), we know exactly which team did the work, usually have a day-by-day task burndown and a proportional allocation. The product backlog items we report are well documented and understandable (thanks to the story form). When auditors visit team members, the team members say the same thing as the executives. This is an auditors dream: that managers and executives report aggregate data verifiably backed by statements from individual contributors.

What Happens in a Transition to Agile Capitalization?

If you are about to embark on a transition from waterfall to Scrum, this is a great opportunity to consider changes in financial reporting. Agile approaches to software development are radically different from waterfall and justify a significant change in financial reporting methodology.

First, you can create a nearly bulletproof system for tracking engineering costs that eliminates the need to track actual hours. Auditors and financial staff will at first be wary with this new approach, then delighted when they realize that everyone—from developers to Scrum Masters to product owners to managers to finance staff—discuss the work your company does consistently and thoughtfully.

You may find that transitioning to a more accurate and responsible capitalization approach dramatically increases the amount of work capitalized. Your finance department should expect a high rate of capitalization because the work of software development is usually an investment in a long-term future. However, a dramatic change can be seen as a red flag to them and their auditors.

You should address these concerns head-on. Explain that agile software practices make this detailed approach feasible. It can be difficult for waterfall teams to responsibly track which design efforts or project management tasks led to which features, while agile methods will expose this information naturally through sprint backlogs. For example, in the past, your company might have lumped post-investment-decision design work into the preliminary project phase; this would no longer be appropriate.

Furthermore, because agile practices create releasable software every month, they can tie infrastructure development work with individual features, and you can capitalize those efforts. Some waterfall companies have felt that infrastructure work was so indirectly connected with user features that it had to be expensed.

Regardless of your situation, be completely frank with finance and auditors. If you expect your capitalization rate to increase dramatically, share that information with them. Discuss why this will happen. And finally, explicitly connect these changes to your company's transition to agile. Your commentary may actually appear in a company shareholder report, which you should welcome, proud agilest that you are.

Summary: Agile Capitalization as Opportunity

If you have read this far, you are likely an enterprise agilest, comfortable with the idea that agile thinking should affect not just engineering, but also finance and other departments. Welcome.

Now that you know more about agile capitalization, your company has an opportunity to report its activities more responsibly to shareholders and tax authorities. This can require a lot of negotiation, planning and process changes to do so. However, your engineering group may be able to hire more engineers, your company may be able to reduce its tax burden significantly and your company's financial reports may stabilize. The value of these improvements may be in the millions.

For agility and the greater good, I remain your humble servant.

Agile Development Drives Service Change at Stockport Council

Emma Collingridge

Faced with pressure to do more for less, Stockport Council in the UK has set out a plan to deliver public services in a more agile manner called Investing in Stockport. It's in part a response to the need to deliver citizen-facing services more efficiently, something common to all local authorities in an ongoing age of austerity.

But Stockport has the added complication of being part of the Greater Manchester Devolution, which sees councils in the region given powers over transport, housing, planning and policing, as well as health and social care budget control. There's a need for a new approach to service delivery, which in turn dictates a new approach to systems development—which is where agile development comes into play.

"There is a challenge out there for all of us in local government," says Emma Collingridge, Digital by Design Programme Manager at Stockport Council. "The budget pressures are more complex and more multi-faceted, around demand, around living wage, around ageing population. What we know is that we are in an era of unprecedented change."

"In Stockport, as well as the ten local authorities with the gift of devolution, we are in a situation where not only do we have less money and more demand but we also have the opportunities and challenges around Greater Manchester devolution, such as some of the changes around health and social care integration, the way children's services and some of the place-based services are being delivered."

One challenge that had to be faced up to was a need to put in place systems that are agile enough to cope with future needs. "This can't be about making one-time changes. This is not about putting lipstick on a pig," says Collingridge. "It's about not just trying to cope with now and manage the current crisis; it's about trying to find a way to cope with now, but in such a way that really sets us up for the future."

That's not something that a waterfall approach to development was going to be able to deliver. That's the sort of approach that Collingridge describes as: "a big traditional implementation, with products from big vendors and implementing them to almost meet our needs."

"When we went to the market, we started to feel increasingly unsure about that approach and increasingly unsure about how this would deliver to meet our requirements," recalls Collingridge. "It might do the bit about 'fixing the now' but would it deliver the ongoing solution we want?

"What we felt as we spoke to more people was that there was another option, a road that would allow us to do more fundamental organisational redesign and organisational change by the back door; by using technology and some of the ways of working, particularly in agile ways of working and agile IT. We could get so much more bang for our buck than just implementing a massive new CRM."

The result of coming to this conclusion was to decide not to buy anything, but to go back to the drawing board. "We went into a period of discovery to try to find out what kind of a technical architecture and infrastructure would deliver that kind of deep organisational change that we wanted at the pace that we needed," explains Collingridge.

> "There is a challenge out there for all of us in local government. The budget pressures are more complex and more multifaceted, around demand, around living wage, around ageing population."

"What we found was that a lot of the solutions that we wanted to go for weren't necessarily the cheapest solutions, but they were going to be the right thing for us for a long time. They embedded some of those principles around agility, around being able to constantly change. We know that if we design a system for now, it can be the wrong system by the time it comes out and [certainly] by three to six months down the line."

Some principles were laid down as a result of this period of thinking. "What we realised was important was that we should build modular solutions around common standards," says Collingridge. "We should build capabilities to enable continuous change. We shouldn't commit to the long term and we should build in the agility and organisational change that we needed."

The shift across to delivering systems according to these new principles is now beginning. "We're doing delivery in a completely agile process," says Collingridge. "We're changing pretty much our entire IT infrastructure. This isn't about putting a team on top, this is about changing the way that we do IT and more importantly about the way that we serve people and the way we understand their needs."

"We have a very large, complex programme with lots and lots of people doing lots of agile development," she adds. "What we are looking to do is deliver viable product around the platform over the summer. The initial platform release will deliver a number of benefits, but more importantly, it will be a one-step change release that will spark a different relationship with the public and a different relationship with services. After that, it will be about a continuous process, about continuous development."

This has ramifications beyond DevOps. "The way that we do change at the moment is very much hooked up to a financial cycle, not hooked up to when we can deliver benefits as soon as possible," explains Collingridge. "We really want to take on the kind of Government Digital Service (GDS) model of being able to have the confidence and ability with senior

stakeholders and services to release capabilities and improvements as and when, so we can go live with those improvements all the time. That's not just tech improvements, but all business process redesign improvements as well."

All of this has already made a difference within Stockport Council. "There has been such a change, even in the past six months, about the relationship between services and IT and the energy and productivity within teams," says Collingridge, pointing to "the kind of evangelical comments" that IT is getting from the business stakeholders.

"Agile IT and agile development and the way they're changing fits perfectly," concludes Collingridge.

"Children's Social Care has said that agile IT, and the way that we're moving toward it, fits like a glove with the way that they're doing change and the way we're helping them to do change. They don't really have a big waterfall plan, so waterfall IT doesn't fit with it."

> "This isn't about putting a team on top, this is about changing the way that we do IT and more importantly about the way that we serve people and the way we understand their needs."

Networking as the **DevOps** Entry Point

David Gee

The natural place for the now DevOps movement to originate was always going to be the server and application space. It's the space where sysadmins realized major benefit early on, from simple to now full stack and gated deployments of mutable infrastructure like bare metal servers and virtual machines.

In recent times, the maturity slope has hit immutable infrastructure and thus destructible runtime environments. Several updates a day can be pushed to an application, and private infrastructure like PaaS can be extended out to public offerings to provide on-demand elasticity.

Networking remains the last thought-about-thing, but in terms of DevOps, the networking space, while a challenging entry point, is the most pervasive.

Networking is the backbone fabric for internal business communication, business-to-business, partner and business-to-consumer transactions. It's the very fabric that applications hook into. Without the network, there would be no internet, no cellular communication and no access to business function serving applications, and thus, no agility as we understand the term today.

The networking movement for years has developed various self-protection mechanisms, some more healthy than others. There are accredited engineers who for years studied the standards-based protocols that originate from the Internet Engineering Task Force (IETF) and Institute of Electrical and Electronic Engineers (IEEE).

Some network hardware and software vendors have built entire career paths for IT professionals based on education, certification and skill promotion in addition to shipping product. These people are the network warriors who spend hours looking at terminal screens, configuring network elements with well-honed domain specific instructions. The trouble is, that's great for "build it and walk away," but not for the agile and dynamic world of DevOps or the agile responses required in modern IT architecture.

By overlooking the network team, organizations can be missing an incredible source of design and operation knowledge that an application or development team does not possess. Instead of simple access tiers and overlay-based networking to bypass the networking team, join forces and bring the agility to the end-to-end network. The coupling between the two parties has unbelievable, business-changing potential.

DevOps to the Network

When we say DevOps, we immediately think about automatically deploying code to an environment, gated processes and rapidly available telemetry for sharing. DevOps for networks is a little different, but not as different as you think.

Networks are predominantly based on a push model in terms of configuration. Being able to test configuration changes is somewhat of a complicated problem due to the simple fact that every network is akin to a snowflake. But that said, the basic principles remain true of each Ethernet and IP-based network. Frames and packets traverse a network infrastructure. The network has resiliency and redundancy built into its very core and is not designed to fail fast.

Testing configuration is complex, but in the name of failing fast, generating and pushing configuration based on a deployable set of code with information pulled from a normalized and verified source of truth like a network team-owned database, is absolutely possible. Distributed testing will verify that the configuration is in place and operating as expected. Sure, tiered fallback has to be considered in case of functional test failure and there is much more to consider like availability budgets. This article brushes the very surface of this concept.

Imagine a world where a container is spooled up, pretending to be an application that tests the new network segment and chain of functions that make up the service. Another container pretending to be a client could be spooled up on an external public cloud and a functional test is automatically executed from the network with results going back to the gatekeeper function. At this point, the network canvass has been tested and thus the actual application can be pushed in full confidence; the infrastructure is ready to receive it. That big green tick now not only suggests the code has been pushed, but a user experience has been emulated from the very network it's been deployed to.

Bring in the desire to tread the DevOps path and the network team becomes massively important in providing this crucial agility factor into the wider company effort. Does it mean network engineers become developers? Some. Not all aspects of a network will be automated and security also has to fall in line with the movement. Automation is often best employed between boundaries within an organization, and with a healthy, curious and blame-free culture, these complexities can be exercised for the full benefit of the business.

There's a way to go yet. Using networking as an entry point is still fairly new, and region by region, the interest, desire and skill level changes. Internet service providers can see immediate benefits of tight integration. Service providers have never been under so much pressure to deliver competitive, over-the-top services.

Europe, in my experience, is the most conservative of each region, with APAC leading the way in bravery, I suspect from an absolute need to do more and close the delivery gaps of services. The U.S. so far appears to be somewhere in the middle, taking a business-as-usual approach.

"Networking remains the last thought-about-thing, but in terms of DevOps, the networking space, while a challenging entry point, is the most pervasive."

For those who have embraced the idea of DevOps, the opportunity to learn something new and solve the problems of today has relit the flame that has slowly been doused over the years. The number of network engineers learning to code is phenomenal, and even if they never write an application, a whole level of understanding has been opened up to build on or create new relationships with people in their organization they might have never thought about before.

Useful additional reading:

http://ipengineer.net/2015/07/netdevops-delivering-network-levers/
http://ipengineer.net/2014/05/from-cli-to-python-beginner/

Also, this podcast was done in 2014 on the network engineering journey to that of a more programmable world:

http://blog.ipspace.net//2014/07/network-programmability-with-david-gee.html#mor

> "Bring in the desire to tread the DevOps path and the network team becomes massively important in providing this crucial agility factor into the wider company effort."

Devs, Ops and the Future for the Software Development Lifecycle

James Woolfenden and Matthew Skelton

With the traditional software development Lifecycle (SDLC), everything's fine if there's complete synchronisation between developers and the operations team.

How often does that happen? The answer to that question brings painful recognition to all too many organisations, where the phrase, "Well, it worked fine in development," is met with irritable frowns from the operations team.

So what impact can a DevOps approach have on this unfortunate status quo? Well, it's not an entirely simple silver bullet waiting to be fired. It demands discipline, organisational process change in some changes, and most of all, a willingness to challenge some of the accepted norms of the SDLC.

While those might be intimidating requirements for some organisations, there's a need to face up to changing realities. With applications becoming far more ephemeral, most notably seen in the rise of

"We need to be cautious about measuring DevOps collaboration itself, partly because collaboration is a means to an end and not always the right approach."

mobile apps, it's critical that development teams have more immediate feedback to drive improvements and ensure a high-quality customer experience.

That also means that the operations team needs to be able to support those goals and rethink its own role. "I can't see any future for a traditional Ops department; for me this is how it already is," says James Woolfenden, DevOps consultant at EqualExperts. "There are two streams: those already embracing the new methods and those becoming bypassed. Ops is a vestige of old IT practices."

Woolfenden says he has seen the shift happening in his experience. "I didn't have Ops in my previous project; I certainly don't have anything like that in the current one," he says, "There isn't any separate role for them. If there is, then they should be part of the team and in no way separate. We don't have separate test teams either. I am part of a small, self-contained cross-functional team. I can only see this approach developing further. I can only see me encountering Ops departments when I'm helping in the transition of legacy IT teams."

But while the goal to improve the customer experience is admirable, how can it be measured in practice? It's a question picked up by Matthew Skelton, principal consultant at Skelton Thatcher, who argues that the starting point needs to be the user experience (UX).

"The starting point for Ops team metrics and monitoring efforts should be UX; this is quite a departure for many teams, and so some help from UX specialists on the Dev side works well," he says. "Ops teams can help improve customer experience hugely by focusing their deep experience of metrics and monitoring on UX and business-level KPIs. Also, we need to ensure that developers are able to use operational data effectively and explore, with Ops team members, how to make use of metrics/monitoring/logging data to improve the software products on a daily basis."

But Skelton strikes a warning note about trying to over-metricise, particularly when it comes to measuring the level of DevOps collaboration. "We need to be cautious about measuring DevOps collaboration itself, partly because collaboration is a means to an end and not always the right approach," he suggests.

"Our metrics should have far more applicability to how our platforms provide consistent and reliable user experiences than to internal team performance. Customer satisfaction may be measured in a market-specific way (basket checkout speed, repeat order frequency, etc.) and correlated to user experience changes via A/B and multivariate testing. Feature usage reporting is also generally acceptable now for most products, as long as the data is anonymised."

Woolfenden argues that metrics should be built into the applications themselves. "Our application is fully instrumented," he says. "We can tell how it's being used in real time. Features can be partially or fully rolled out and switched via toggles in production, our responsibility. Each week we demo new, major features in production to the main stakeholders."

> "We only have to look at the recent (and increasingly frequent) data breaches to understand that proactive security and resilience activities are more needed than ever before."

But in this new world order, what's left for the traditional Ops team to do? As elements like configuration, provisioning and test-driven development are

automated, what's left beyond the inevitable keeping the lights on? If that question isn't answered, cultural resistance to change is likely to kick in from the Ops side.

Woolfenden takes a firm stand on this. "I'd split up the Ops team and make them part of a cross-functional team. Getting rid of silos is key to what DevOps is supposed to be about," he states bluntly. "Maybe some of those old skills can be assimilated. There has been a revolution in the approach taken with Stateless/Immutable backed up by having automated infrastructure as code for the entire stack."

Skelton on the other hand does see a new role for switched-on Ops teams. "The Ops mindset is necessarily different from the Dev mindset," he argues. "Service restoration, incident response, pre-emptive capacity or resilience improvements are all crucial things that Ops teams know how to do well, and that Devs often don't know or about."

It's also necessary to kill off some myths, he adds. "The idea that automation and cloud mean that Ops have no role to play is causing a significant amount of pain for many organisations," he says. "We only have to look at the recent (and increasingly frequent) data breaches to understand that proactive security and resilience activities are more needed than ever before."

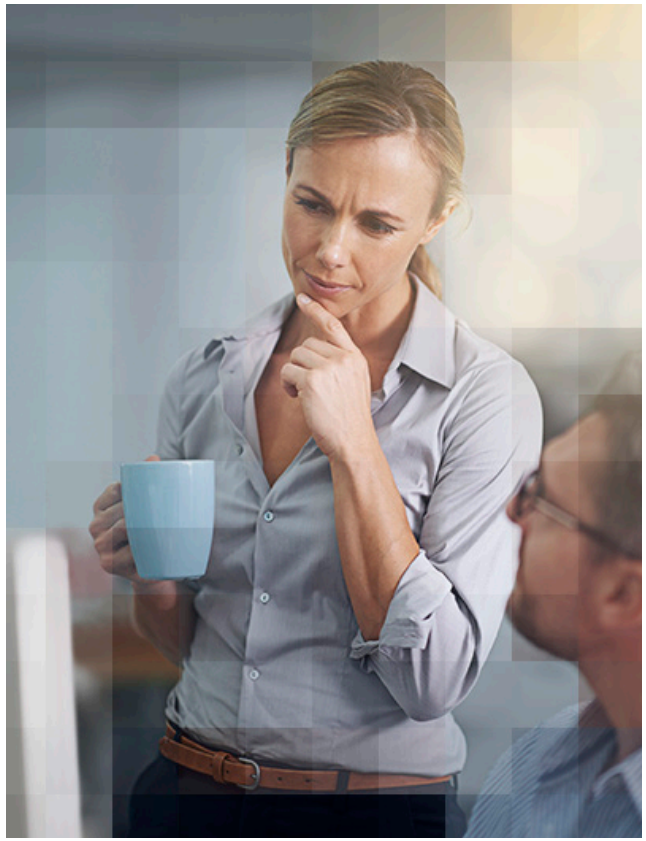
"Yes, old-school Ops people who prefer to rack-and-stack servers or configure SANs will have fewer opportunities, but it is woefully naive of tech teams to believe that there is no need for the proactive, diagnosing Ops mindset or experience, whether your software is in the cloud, on-premises or deployed as IoT sensors."

In the end, there's a new world order, but it's going to require some mind shifts to achieve the real benefits and make a positive impact on the SDLC of old.

DevOps and Deviance—When Bad IT Practices Become Accepted as Normal

Peter Waterhouse

How many times have you witnessed a suboptimal IT practice that everyone else thinks is ok? Then, over time, have you accepted the behavior as being just fine and dandy and started practicing it too? Of course you have; it's normal human behavior.



Regardless of whether you lead a startup or work in an established business, we all have a tendency to accept dodgy and suspect behaviors. Even if outsiders see them as wrong, our IT teams are so accustomed to using them (without any adverse consequences) that they're quickly established as normal and accepted.

Studies into what's commonly referred to as the normalization of deviance have been conducted in areas from health care to aerospace, with evidence showing that many serious errors and disasters occur because established standards have been bypassed and bad practices normalized.

While examining this phenomena is critical in the context of safety, it's equally applicable in how we develop, secure and operate software applications. With the boundaries blurred between the digital and physical world, any adverse behavior leading to security and reliability issues could have dire consequences for customers. And when software becomes infused into long-lasting products (from light bulbs to limousines) it's not so easy to discretely exit markets.

As businesses look to software innovation for growth, the critical differentiators become faster time-to-market and high quality applications. Unfortunately, both can be compromised if pre-existing change aversion or newer speed-at-all-cost mandates lead to a normalized bad practices. More critically, if a head-in-the-sand IT culture persists, systemic business failures may eventuate—think massive security breaches or major cloud application outages.

The DevOps movement, with its focus on collaboration across development and other IT functions, is now

regarded as the best way to establish the culture and environment needed to support fast and reliable software delivery. This, together with guidance from other fields, can help IT identify and eliminate poor practices. In the field of health care for example, studies illustrate seven factors that lead to a normalization of deviance, all of which are extremely relevant in IT:

- **The rules are stupid, dumb and inefficient.** In health care, accidents occur when practitioners disable equipment warning systems because alarms are seen as distracting. This happens in IT all the time, like in IT operations where staff will filter out noise and alerts on the many monitoring consoles because they regard them as irrelevant. It also surfaces when testing is skipped because of lengthy manual processing and setup delays.
- **Knowledge is imperfect and uneven.** Employees might not know a rule exists, or they might be taught a practice not realizing that it's suboptimal. In IT, this persists because many new employees feel uncomfortable asking for help, or when the application of new technologies distorts logical thinking.
- **The work itself, along with new technology, can disrupt work behaviors.** To support goals of more continuous software delivery, organizations are introducing many new technologies and methods—like microservices and containers. New work practices and learning demands may lead staff to poorly implement technology or use it to perform functions it was never designed for; for example, containerizing monolithic legacy applications just because it's possible.
- **We're breaking rules for the good of the business.** Staff may bypass rules and good practice when they're incentivized on faster delivery times or delivering new functional software enhancements. For example, repeatedly procuring additional (but unnecessary) hardware to rush through an update, rather than addressing the root-cause of performance problems.
- **The rules don't apply to us … trust us.** Autonomous agile teams are extremely valuable, but empowering them to select their own one-off tools, haphazardly use open source code or bypass compliance policies can compromise program objectives or lead to security breaches. Unfortunately in today's fast-paced digital business, talented professionals often feel completely justified in playing the trust card.

> "Regardless of whether you lead a startup or work in an established business, we all have a tendency to accept dodgy and suspect behaviors."

- **Employees are afraid to speak up.** Violations become normal when employees stay silent. How many times have poor software code, costly projects (and bad managers) been tolerated because junior staff are afraid to speak up? Even in IT organizations with a strong, blameless culture, people can and will stay quiet for fear of appearing mean.
- **Leaders withhold or dilute findings on application problems.** Whether you work in health care or IT, no one wants to look bad to managers. Rather than present ugly and unpleasant realities, many will distort the truth, presenting diluted or misleading information up the command chain. In IT, this behavior is easily normalized, especially if teams get away with reporting technical vanity metrics over more actionable and outcome-centric performance indicators.

No sudden cultural reawakening across the IT organization or liberal sprinkling of collaboration fairy dust will eliminate ingrained bad practices, but DevOps and Lean thinking can help identify warning signals. This starts with leaders clearly visualizing the flow of value delivered by software applications, pinpointing all the bottlenecks and constraints impeding delivery. Analogous to pathway stepping stones, these are all the value interrupts, which when lifted, reveal all the process and technology issues that cause good people to do the wrong things. Immediate candidates are software release and testing functions, but analysis shouldn't be limited to the development side of the software factory. Every stone, be that enterprise architecture, stakeholder engagement, information security, vendor management, operations

or customer support can hide ugly behaviors that over time can and will become normalized.

Of course, identification is just the start. Next comes the hard part, with leaders using evidence to impress how behaviors impact current performance and business outcomes. This might involve using new tools, but this again courts disaster when advanced technologies become a vehicle to automate bad processes.

As with anything involving people, the organizational and psychological barriers encouraging staff to break rules or for their colleagues to remain silent is where most attention should be focused.

> "No sudden cultural reawakening across the IT organization or liberal sprinkling of collaboration fairy dust will eliminate ingrained bad practices, but DevOps and Lean thinking can help identify warning signals."

Mark Schwartz

CIO, Office of Information Technology Management Directorate

Mark Schwartz is the CIO of USCIS. One of his key goals is to increase the organization's responsiveness to mission needs by reducing time from concept to deployment for new capabilities. To support this goal, Schwartz has introduced such practices as agile and Lean development, continuous delivery and DevOps. He also leads efforts across DHS to introduce agile IT approaches.

In 2015, Schwartz received the AFFIRM award for Leadership in Technology Innovation and an Amazon Elite 100 award. Before this position, Schwartz was the CIO of Intrax Cultural Exchange, where his innovative Family Room application drove dramatic market share, revenue and profit growth. In 2006, CIO Magazine recognized this accomplishment with a CIO 100 award. In 2010, he was named one of the Premier 100 IT Leaders by Computerworld Magazine.

Jody Mulkey

CTO, Ticketmaster

An accomplished technologist and transformational engineering leader, Jody is known for building high-performance systems and teams. Prior to Ticketmaster, Jody spent over 14 years at Shopzilla, Inc., a leading source for connecting buyers and sellers online that reaches a global audience of over 40 million shoppers monthly. As Chief Information Officer there, Jody led the overall technology development and operations of the company. Part of the inaugural team at Shopzilla, then Bizrate, Jody built the company's data systems, analytics and infrastructure from the ground up.

Martin Thompson

High-performance and low-latency computing specialist

Martin is a Java Champion with over two decades of experience building complex and high-performance computing systems. He is most recently known for his work on Aeron and SBE. Previously at LMAX, he was the co-founder and CTO when he created the Disruptor. Prior to LMAX, Martin worked for Betfair, three different content companies wrestling with the world's largest product catalogues, and was a lead on some of the most significant C++ and Java systems of the 1990s in the automotive and finance domains.

Martin blogs at mechanical-sympathy.blogspot.com, and can be found giving training courses on performance and concurrency when he is not cutting code to make systems better.

Dan Greening

Managing Director, Senex Rex

Dan R. Greening coaches executives, managers and teams to help them gain and maintain global agility. Dan first used and researched Scrum, agile and Lean methods in 2007, and rapidly became a thought leader in the field, publishing groundbreaking work in agile metrics, portfolio management and capitalization. Now, Dan has distilled personal, team and organizational agility into a set of five agile base patterns: if you do them, you're agile; if you don't, you're not.

Emma Collingridge

Digital Design Programme Manager

Emma Collingridge is the Digital by Design Programme Manager at Stockport Council. She has a background in data analytics, policy development and business transformation. With expertise in organisational design and service change, she has senior management experience in successfully implementing innovative programmes that use the latest technology to improve frontline services for citizens and deliver value for money.

David Gee

EMEA Lead and Principal, NetDev team, Network Automation and Evangelist

David Gee is EMEA lead for the Brocade NetDev services team. Some of his previous roles include: founder of a high-profile specialist gaming website company, technical director for a VAR, principal consultant for an SI and core architect for an ISP. He has delivered courses on NETCONF and YANG and has more training sessions planned. In addition to his role at Brocade, David is also a podcast co-host on Ivan Pepelnjak's excellent education and networking technology website, http://ipspace.net. David blogs at http://ipengineer.net.

James Woolfenden

DevOps Specialist

A veteran DevOps contractor, James grew up in and around the IT business, getting his start in IT building Compaqs and XTs in then-upcoming Clerkenwell. He has worked in a wide variety of IT sectors and in multiple roles within the SDLC and now contracts for a leading agile/DevOps consultancy.

Matthew Skelton

Co-founder, Skelton Thatcher Consulting

Matthew Skelton has been building, deploying, and operating commercial software systems since 1998. Co-founder and Principal Consultant at Skelton Thatcher Consulting, he specialises in helping organisations adopt and sustain good practices for building and operating software systems: continuous delivery, DevOps, aspects of ITIL and software operability.

Pete Waterhouse

Senior Director, DevOps Strategy and Marketing

Pete has been involved in the development, support and marketing of software solutions for more than 20 years. He has held a number of management, consulting, technical sales and strategy positions in areas such as cloud computing, DevOps and IT business management. Pete blogs on a range of disruptive business and technology trends, with articles appearing in publications including InformationWeek, Wired Insights, DevOps.com and App Developer.

Next Steps

Mainstream adoption of DevOps is here. Is your organization ready to seize all the business benefits and opportunities it presents? At CA Technologies, we have built a portfolio of products and solutions on our DevOps expertise.

Visit ca.com/contact to learn more about how CA can help you close the gap between your developers and your operations—and keep your competitive edge in the application economy.

CA Technologies (NASDAQ: CA) creates software that fuels transformation for companies and enables them to seize the opportunities of the application economy. Software is at the heart of every business, in every industry. From planning to development to management and security, CA is working with companies worldwide to change the way we live, transact and communicate—across mobile, private and public cloud, distributed and mainframe environments.

For more information on DevOps solutions from CA Technologies, go to: ca.com/insights/devops

#BusinessReWrittenBySoftware

CS200-233592_1016