# Digital WTF

Coté

# Digital WTF

Coté

This book is for sale at http://leanpub.com/digitalwtf

This version was published on 2019-06-05

ISBN 978-1-7331395-0-2

*I been out on them choppy waves*
*and it's hard to say where this land*
*begins and that water stops*

# Contents

# Foreword

In 2005, I was roaring along with my fifth startup. I had built a consultant company, composed of 30 of the world's top engineers, working with the IBM product portfolio called Tivoli. Just for comparison, IBM had over 10,000 consultants, yet we still had more Tivoli certifications. In fact, at one point in the early 2000's, we won an award for the most IBM Tivoli Enterprise Certified consultants. To give you an example of how good we were, IBM's global consulting services would call us for training before they would call their own internal or commercial training classes. In a 10 year period, I had trained over 10,000 Tivoli professionals. When the Department of Defense Information Systems (DISA) called IBM needing a Tivoli trainer to teach classes for the Navy's Pacific fleet, specifically the NIPR/SIPR systems, I was the only person on the planet that was certified in all the classes they needed from a trainer. I was the king of the world. At this point, I had also co-authored 7 books on Tivoli. Needless to say, I was considered a world-renowned expert on Tivoli.

I am telling you all this because, for reasons I won't go into, only one year later, I was broke and had to move my family into my brother-in-law's basement. It's the old cliché: whatever doesn't kill you, makes you feel really crappy, but eventually makes you stronger. Forced to bootstrap myself again after 20 years in this industry, I started with one specific goal, to not meet the same people on the way up as I did on the way down. In order to achieve my goal, I decided to look at alternative tools to the Tivoli portfolio, eventually focusing on open source tool alternatives to the Tivoli tools. I decided the best place to start was to create a blog. Having come from a world where I worked exclusively with enterprise proprietary software, I found it easy to be cynical, while

also incredibly intrigued when making comparisons. Nagios and Puppet were easy targets to start with. Sometimes I would make fun of, and other times, I would give serious comparative analysis.

After a few months, I started getting some followers on my blog that seemed to have a similar sense of humor as mine. Some of them would make comments, and a few would actually create links or talk about my posts on their blog. One of these dudes was called Coté. Although, whenever he quoted me, it sounded way cooler than my original wording. Mind you, in the Tivoli-o-Sphere, there were no one-named people, so I decided to proceed with caution when responding. When I asked another new follower and kindred soul, Mark Hinkle, "who the hell is this Coté dude?" Mark pulled out his phone, called him, and made an introduction. Before long, Coté and I became blood brothers and eventually ran a podcast together for around 5 years called The IT Management Guys. Imagine the Car Talk guys but only we talked about open source, cloud, and sometimes spending 15 minutes talking about the best hamburger we had that week. Why? Because we could.

So before telling you my thoughts about Digital WTF, let me start by saying that having a conversation with Michael Coté is a rare, intellectual, and witty treat. It's kind of like when you're reading a great book, and you start worrying because the ending is approaching and you enjoy it so much. In other words, he is fun to listen to, and even more fun to read. It was no surprise to me that when I was given an early copy of Digital WTF, I freaking loved it. Michael Coté describes his *Digital WTF* as a paste-pot of interesting "digital transformation" stories, and with that Coté signature style, he keeps you guessing whether he's laughing with us or laughing at us. Most likely both. At one point in my early career, I lived in Texas for five years, so when Michael sneaks in his little Texan'isms like "cottoning on to new technologies," I get a bonus giggle. However, you don't have to be from Texas to get a kick out some of his non-colloquialisms like "hey, get off my back, tapered sweatpant milinums" referring to Starbucks as a software

company. If a chapter that includes phrases and names such as meatware, devops, Kent Beck, Melvin Conway, The Mythical Man Month, and Google SRE all peak your interest then this is definitely the book for you.

I often say when talking about my journey over the past 10 years, i.e., my post-Tivoli adventure, I have had the privilege to get to know a handful of incredible people. I joke that I feel like I should be paying them when I have conversations with them, and Coté is most definitely part of this group. So let me leave you with this, if you still are not clear on how much I recommend this book, I will be buying one of the first commercial copies of Digital WTF even though I was given a free one for early review. Why? Because he's Coté.

Botchagalupe

*a.k.a John Willis*

# Preface

I was reading a Mencken biography a while back, Disturber of the Peace. A book which is, magically, available as an audio version: there's something about hearing Anthony Heald's rendition of H.L. Mencken's cigar chomping voice that's comforting. Mencken wrote an endless stream of articles in-between editing his various newspapers and magazines. To make extra money and compendiums of his articles for easier consumption, he'd often "get out the scissors and paste-pot" to create a "book."

As a side note, while Mencken is, no doubt, an important figure in American letters, each time I try to read his material, I mentally file him away on that shelf I call "things to read when I have nothing else to do." This shelf, rests, no doubt, right next to my deathbed. I'm often stricken with the claustrophobic fear that once I become bound in a nursing home, my body so frail and sick that I'm reliant on my new family of nurses to accomplish the simple task of rolling over, that my mind will eat itself alive because I won't be able to read and consume information. Perhaps all this voice-activated technology will pay off them: hopefully it'll respond to geriatric muttering. "Alexa! TURN. THE. PAGE….TURN. THE. THE…NURSE!!!"

But, back to the present day, where I still have most of my faculties.

Times have advanced and we no longer need scissors or a paste-pot. We have computational cut-n-paste! Many - perhaps too many - books are composed of blog posts munged together - not unlike this one. I usually think of these books as rather like cheating, and, not very good. Despite this, there are good ones, The Hard Thing About Hard Things stands out, and, from the pre-Internet age numerous Hunter S. Thompson "books" which were themselves paste-pots.

Never one to shy-away from personal hypocrisy, that's exactly what this is, dear reader. A digital paste-pot of blog posts, columns, and other "small things" I've written over the years. Somehow, I ended up writing for a living, first programming and then more traditional forms, along with podcasts. On the Internet all this writing gets lost and you're never really afforded the chance to see the ongoing narrative. Here, primarily to see that for myself, I've collected together some of my pieces. They're heavily weighted towards things I've typed up in recent years at The Register. Some common themes have emerged: DevOps, programming, vendor-sports, and enterprise software.

Anyhow, I loathe reading through front-matter when starting a book. Enjoy the typing!

*Coté*

*Amsterdam, Fall 2018*

# Chapter 02 - Digital Transformation: WTF?

At some point the phrase "digital transformation" must have meant something specific, even pragmatically useful. Now, it means nothing in that it means everything new and helpful you'd do with computers. I remember hearing the idea of a "system of engagement" back at an Adobe analyst summit in 2008 or 2009. They even brought in Geoffrey Moore who'd just recently been putting together the idea of leveraging (oh, pardon me - I've been eating a rich diet of enterprise-speak recently) user-centric software to more closely know and sell to your customers. Thanks to web applications, mobile apps, and something no one remembers called Rich Internet Applications (RIA), you could monitor and analyze every single thing your customers did and even tailor your sales offerings, pricing, and application features to them.

Much of the early work here fell under the idea of digital marketing: figuring how to use highly targeted ads in Google, and later Facebook, to perfectly target promotions. With deep user tracking on the back-end and even "big data" analysis, you could start to know your customers like never before and sell to them more effectively, even stalking them across the Internet.

Seemingly overnight, all of the IT industry's efforts were dumped into "social" and "mobile" to use these new tools to sell to customers. As Jeff Hammerbacher quipped back then, "the best minds of my generation are thinking about how to make people click ads." The idea of "Chief Digital Officer" was born.

The dazzle wore off quickly. These tools were so easy and effective to use that everyone could do it quickly. With this new, targeted

and intimate channel to reach customers, existing brands glommed onto selling candy and toilet paper in Facebook, new direct-to-consumer brands from razors to bras thrived, and, as ever, Amazon was rocketing along like Leonard Smalls across the desert highway.

Once the competitive advantage of digital marketing ossified - becoming just what you needed to stay keep your head above water - "digital transformation" was adrift, searching for a tactical meaning beyond better advertising. What I saw, and continue to see, is that digital transformation now means using agile principles, cloud automation, and user-centric design to improve your software capabilities. More than just incremental improvement in existing capabilities, you do this transformation to improve your business.

These pieces discuss that new meaning and aspiration of "digital transformation."

# Digital transformation?! Your boss's PowerPoint New Year resolution, deconstructed

Hey, it's the new year. Time to let those annual planning slides shimmy over you, washing away the dangling tickets of last year like a purifying clean install. Somewhere amid pictures of robots shaking hands with meat-mawsand millennials writing on glass walls will, no doubt, be the details of your firm's "digital transformation."

At first, you may be shocked to hear that you're so analogue – weren't we up to our eyeballs in digital last year when we updated all the desktops and finally enabled the CEO's iPhone to check email? Then, as Dear Leader flips through some eye-popping figures around Uber and Tesla (all the money is in multi-sided platform businesses overflowing with customer data, now, you now), you'll start to think: "Oh crap. They're serious. Erm. So, what exactly is 'digital transformation'? (Should I be updating my LinkedIn?)"

As my writing during the past year attests, I spend much of my time surveying the kipple of decaying digital transformation efforts. They always start with grandiose trend chasing – AI! Blockchain! IoT! The Gig Economy! Augmented Reality! Drones! – but they end up with something more simple: just using software to automate previously manual-driven business processes.

It's certainly not as sultry a strategy as asking Alexa to machine-

learn her way into estimating how many nappies you'll need to order fortnightly based real time feedback from nappy-can sensors in your demographic…but more pedestrian applications of software will likely prove better at generating cash for your business.

## "Digitizing" is the new paperless & humanless

A project to "digitize" the green card replacement program in the US provides a good example of the simple, pragmatic work IT departments should be curating for 2017. Before injecting software into process it'd "cost about $400 per application, it took end user fees, it took about six months, and by the end, your paper application had traveled the globe no less than six times. Literally traveled the globe as we mailed the physical papers from processing center to processing center."

After discovering agile and cleaning up the absurd government contracting scoping (a seven year project costing $1.2bn, before accounting for the inevitable schedule and budget overruns), a team of five people successfully tackled this paper-driven, human process. It's easy to poke fun at government institutions, but if you've applied for a mortgage, life insurance, or even tried to order take out food from the corner burger-hut, you'll have encountered plenty of human-driven processes that could easily be automated with software.

After talking with numerous large organizations about their IT challenges, to me, this kind of example is what "digital transformation" *should* mostly about, not introducing brain-exploding, *Minority Report* style innovation. And why not? McKinsey recently estimated that, at best, only 29% of a worker's day-to-day requires creativity. Much of that remaining 71% is likely just paid-for monotony that could be automated with some good software slotted into place.

# Add robots here…

In retail and banking, digital tends to revolve around omni-channel programmes (selling our products in more ways than just the till and online, like delivery), adding in more analytics (help us find more paths to the customer's wallet), and cleaning up the crufty, slow-moving application stacks of the past. The last one usually goes under the banner of "enable innovation", which though vague and unhelpful usually just means "burn down the legacy stacks and slam in all that cloud stuff so we can actually deliver software faster."

Others will summarize the goals of digital transformation as increasing an organization's intelligence, agility, and customer-centricity. But it all amounts to the same thing: spinning up the IT Morlocks to actually get out there and provide new, software-driven capabilities to the business folks. These dry-cleaned Eloi then direct their new toys to either cut costs by becoming more efficient or grabbing more money by inventing new business models.

Interestingly, "mobile" is often further down the list; I suspect this is because mobile was the craze years ago and companies have either finished out their programmes here, or became exhausted trying. That said, I'd wager that most of these efforts were to simply reskin existing web-based apps into native mobile apps. There's still plenty of room to introduce brain-dead simple but clearly useful features like letting people turn off their credit cardswhen they think there's monkey business afoot after their teenager scuttled off to the mall. Just unlocking your hotel door with your phone ("the Internet of door-knobs") or glancing at your watch to see which airline seat you'll be stuffing yourself into does wonders for making life suck less.

Further back, "digital" has clearly ceased to mean "the social" as it seemed to when the term emerged years ago as companies were scrambling to figure out how many pictures of sandwiches they

should post a day on Instagram. Hopefully you've long figured out the right times of the day to tweet about your upcoming deals on laundry detergent and your most recent thought-leadership think-pieces on industrial-grade cement and ethically sourced goose down.

## Go digital without going crazy

When it comes to the Making My Life Less Tedious Department, it's encouraging that companies are cottoning on to new technologies. 451 Research surveys have found that during the past year, companies self-identifying as "early adopters" has doubled, up from a scant 9.5 per cent to 17 per cent. This is still paltry when you look at all the opportunities to automate those boring, low-value businesses processes, but at least it's progress in the right direction.

Still, that same outfit says that a whopping – but not unexpected – 75 per cent of organizations are on their back-foot when it comes to planning out their digital transformation: they're just now sorting out which slides to have in the back-up section of their digital transformation decks (I'd suggest any involving people wearing a VR headset, but I don't want to tell you how to live your life).

And if you're facepalming about how obvious and inane it all is, yes, folks: that's the point.

With all the breathless pixels spilled on AI, IoT, machine learning, my favorite darling "cloud", and other haunting tales of "digital disruption", it's all too easy to whack through the digital miasma and end up with a suite of future-shock ready "solutions" that have little to no bearing on your daily operations.

When your well-heeled strategy navigators are done vellicating through their master 2017 strategy deck, try your best to pull their eye-holes closer towards the basics that, while boring, will have a more profitable effect on your businesses and your customers' lives.

*Originally published in* The Register, January, 2017.

# You may not be a software company, but that isn't an excuse to lame-out at computering

I don't begrudge organizations who want us to start calling them "software companies". People are free to do whatever they like with such trivial labels, I guess. But the tick of such labelling has always been an annoyance to me.

## No, you're a company that uses software effectively

Most companies saying "actually we're a software company" are anything but. They very rarely sell software as their core business. Of course, I'd never shy from bombastic overstatement (or too much redundancy). These companies are trying to make a valuable point: they're now using custom-written software to do more than digitise paper-driven, manual processes and customise their ERP systems into cement. They're now able to program their business.

Everyone's favourite pizza provides a hot, steaming example. While Domino's boasts that you can order pizzas from your wrist and Papa John's makes it lickity-split easy to customise your pizza on an app (for some reason, you can't add anchovies except by phone – file a ticket!), these two companies are still fundamentally, well, pizza companies.

Starbucks has long been an example of a company comfortably creeping up the "digital transformation" curve. Their software-driven orders have been so successful that mobile orders have been known to clog their meat-space. Still, when I go there (hey, get off my back, tapered sweatpant milinums! I just want some coffee!), I'm happy to find coffee in my cup instead of a numbered stack of those mini CDs begging me to click on "startup.exe."

## Do you even computer?

In an era where Amazon and its three friends are trundling through everything, it's easy to get wrapped up in the need to transform to a software company. That said, would you even call Amazon a software company? Clearly, in their cloud business they are, but the retail business is more about ruthlessly creating and using software to, well, sell stuff. You can throw a "multi-sided platform" flashbang into the mix to befuddle this point, but at the end of the day, Amazon's nickname, "the everything store", tells you exactly what the company is.

Finance has for a long time used software of all sorts – custom and off-the-shelf – effectively and it's little wonder that they're one of the few industries to have quickly staved off Fear of Silicon Valley Eating Your Lunch. While incumbent banks have been slow to adopt mobile payments, they're now spinning heads at how quickly they're catching up. Banks have a good track record of acquiring pesky finance startups and they've been stuffing themselves to the gills with nerds. Recently, Goldman said that a quarter of their employees are engineers, supporting over 1.5 billion lines of code. JP Morgan Chase has somewhere in the region of 19,000 developers.

Banks have always understood IT well enough to gorge themselves on it; many an IT vendor salesperson has filled their wrist with heavy watches and pegged out their retirement accounts by going up and down Wall Street. Sure, I'll concede that you can get

all intellectually crafty and point out that money is, largely, just numbers in a spreadsheet somewhere, but it'd be odd to call these banks "software companies."

## Creating software is the art of failure...

The problem with calling yourself a software company – beyond the obvious fact that you're not selling software – is that you must now think and act like a software company. Software companies, especially young ones that are no longer just extracting maintenance fees, are built around one of the core problems of innovation: failure.

Failure in the software startup world is enshrined in the idea of failing fast. Software companies have an unnatural comfort with failure. They continually throw software at the wall to see what sticks, observing how people use the software and tweaking it to get the features just right. There are all sorts of fancy phrases like "product/market fit", but at the end of the day it's plain old common sense: you rarely get it right the first, or even 31st time.

Funding software is driven by the idea of failing – the more the better, even, so long as it's quick. Venture capital's model spreads risk over numerous startups, hoping for that one giant payoff. The creation of new software is an extremely risky business. While figures like 90 per cent failure rates seem at first astounding, after a few decades of anecdotes of startup failure across the industry (and at least two myself), that 10 per cent success rate starts to look amazing.

Companies outside of the technology world are ill prepared for this kind of gut-wrenching ride. Expectations are more incremental in business improvement rather than transformative: if we put more cash into our existing business, perhaps making it cheaper to run

in addition to simply selling more of our product, we can increase our return on spend. You throw together a business a case with the audacious notion that you know how much your new venture will make in the future and how much it will cost to get there. Thus, you can figure out the return on investment, or "ROI", that snipe that finance people make unsuspecting nerds hunt out in the Forest of Finance. A seasoned software innovator would sniggle at the notion that you'd trust such figures: hurtling into the unknown can't be put into a spreadsheet.

That all sounds daunting, but it's good to whiplash back to the fact that doing software actually is core to succeeding and surviving in business. While "that's fine for Amazon" may seem to apply to its profitless chewing up of every industry except mining and cement manufacturing (so far), their business and others' prove both the value of creating a programmable business, and that it's actually possible to do so. You just have to know what you're getting into and structure your executive minds correctly.

## ... so get started failing

Judging by surveys that show a still slow adoption of "digital transformation", there's likely a good five or even 10-year window open in various industries to become the "actually we're a software company" of your industry. Estimates vary, but surveys are showing that something like a third of organizations are actually doing anything about improving their software. The field is wide open for companies to make themselves more programmable by fixing how they do software. Now is the chance to grasp at some new innovation levers and actually do something different with your business. Either that, or look into Big Cement.

*Originally published in* The Register, January 19th, 2018.

# Chapter 02 - All the New Meatware

I've spent much of my career observing and commenting on how organizations (businesses and governments) write, use, and care for custom written software. This is distinct from "packaged software," sometimes called "Commercial Off the Shelf Software" (COTS): that software you buy and install. When you shop online at Amazon, get paint mixed at Home Depot, or sign-up for insurance in an Allstate office, you're interacting with software those organizations wrote and run themselves.

In the "cloud" era, where the toil of running your own software in your data centers you can easily shift your applications over to SaaS versions. While the costs may not always be cheaper, the overall cost of ownership does lesson - you no longer need all those operations people to care and feed all the hardware and ongoing updating of on-premises software. In theory, you break the problem of slow upgrades as well: SaaS companies tend to release new software multiple times and year and don't let customers stay on old versions.

As those applications move to SaaS models, the question becomes "what value is IT to our organization?" At one point, I'd bandy about the formula "IT - SaaS = what?" The answer, of course, was writing your own software, as I wrote in a report 451 Research back in 2014:

We believe that application development is, indeed, a vital and valuable part of the industry: our theory is that the majority of cloud spending originates with software developers as the prime movers. Applying the formula 'IT - SaaS = what?' it increasingly

seems the case that the 'what?' is custom-written software for ISVs, SaaS and increasingly companies like Nike and Starbucks that are relying on in-house software development for new products such as the Fuelband and mobile payments. Starbucks, for example, is estimated to have pulled in $1bn in sales from its mobile app.

So, you spend the bulk of your IT resources (money, time, and attention) on hordes of developers, designers, and product managers in place to create "programmable businesses": finally delivering on the dream that how an organizational functions could be coded and improved each week, if not day.

This mode of operating, of course, requires all sorts of changes in technology and people: hardware, software, and meatware. For whatever reason, most of this kind of thinking goes under the heading of "DevOps," a curious mixture of agile software development, high scale operations, and organizational process. The last is usually called "culture," but I tend to think of it as just "stopping doing dumb shit."

The pieces here comment and counsel on the "what?" in that hokey equation of mine.

# From dancing bears to blameless post-mortems - My History of DevOps

"The talks get a little repetitive, don't they?" she said as we were walking out of the elevator and through the lobby, escaping the latest two-day DevOpsDays nerd fest. Unable to resist the urge to mansplain, I meekly volunteered that most of the attendees are first timers, so, you know, maybe it's new to them. Upstairs someone had said they'd like to see more technical talks, and less, as they're called, "culture" talks. Of course, I hadn't attended any of the talks because, you know, a thought lord like myself goes to many of these and has seen "all the talks." Even I'm sick of all this culture stuff!

## Everything was going well until the people showed up

This emphasis on "culture" is well known to induce agenda and presentation nausea. For example, the most fashionable architectural style of the moment starts with humans: one wants to do microservices to take advantage of how humans can't help but build systems that mimic how they organize themselves and, thus, communicate with one another. It's all people, the latest microservices deck-flipper will say.

And then there's handling failure: instead of (only) hardening systems so that they never fail, accept that they'll always fail, and rapidly learn from failure, even relishing and rewarding it. Failure

is learning, comrade! This push to improve by failing brings about the "blameless post-mortem," perhaps the most baffling concept for the sassy old-timers in the glasshouse.

In the tech industry, we're never really sure which is more important: the tool, or how people use the tool. There have always been at least two humans involved, the builders and the users. The builders are the ones who create the software: developers, designers, operators, QA staff, product managers. And, of course, there's the people who actually use the software, the users, sometimes called "the customer," especially when it comes to consumer tech.

## The Hyborian age of computing

Before the recorded time of the web, The Mythical Man Month emphasized the best way to organize developers, namely in something analogous surgical teams - a sort of great man theory. Getting the right builders in place was key to great software. Of course, much revived now, there was Conway's observation, drawn up into a "law" that (put slightly wrong) said software architecture will model the structure of the organization that created it. Getting software to work well and do a job was something of a dancing bear for a long time: the quality of the bears dancing was not the axis of judgement, the fact that the bear could dance at all was the point!

In response to this, you saw a hoard of "usability" experts descend on the land. Here there were things like one way mirrors, user interaction testing festooned with cameras recording the user's every move. It was expensive, and slow. And in most cases, the results seemed trivial: this button's text should be bigger; no one understands this error message; the configuration wizard should probably have less than 30 panels.

Nonetheless, the cat was out of the bag. The technology was now good enough that we could pay attention to how well actual users

- humans - can use this software to get things done.

## Things get extreme

Around this time, in the '90s, early notions of agile software development formed. Any history of agile is fraught with a parade of agile'splainers with talk of Boehmian spirals, roses, and wikis. That's fine, and delightful over some snifters, but let's simplify it. In 1999, of Kent Beck's eXtreme Programming Explained described a method that integrated the builders and users together in a novel, just crazy enough to work way. It crystallized while working on Chrysler's HR system, so it certainly had "enterprise" chops: this wasn't some pizza-based method for creating new Space Quest episodes. It was for real jobby-jobs!

One of XP's core insights is that we have no idea what our software should actually do, and especially how it should be implemented, until we start trying. Rather than imaging the requirements a priori, it's only through an ongoing conversation with the user that we'll discover the right features. To do this, you would slice down the release window to something like a week incrementally co-innovating with the users, creating small pieces of functionality and asking them "whatdya make of that?" You'd conquer the unknown by shipping, and changing your approach as you learned more.

To do this, you had to do less each cycle, automate quality control with tests, and optimize the labor of the developers with pair programming. Even more bonkers, you'd pluck someone from "the business" - or even actual users! - to embed in the team, to be the voice of reason and fight for the users, as they say.

These ideas ruffled the feathers of contemporary practitioners to no end, they'd scoff and call agile people "cowboys" and other such derogatory grunts. "Agile" seemed bananas. Instead, people trusted their ability to predict what the software should do, confident that

they could maximize requirements fidelity and quality far beyond than those absurd, agile short release loops.

## Converting cowboys to suburbanites

Nonetheless, as failures continued to rack-up with this "big-up-front" approach, people kept returning to those tales of success from deep in the wild-west of agile. With a few revs and splattering on some enterprise seasoning, the precepts of agile slowly became what everyone was doing. At least, what people claimed they were doing, ongoing surveys on agile practices actually in use continue to show slow adoption over 20 years later. Everyone's agile in spirit!

Early on, in 2001, the agile manifesto codified a mantle of principals, all wonderful sounding and terribly humane. For my money, the crowning achievement was the idea to value "responding to change over following a plan." In other words, as that hard-working, humble golden retriever put it: I have no idea what I'm doing.

Among many competing agile thought-technologies, Scrum won out. There are many possible reasons why scrum was so widely and commercially successful: perhaps because of its highly structured nature, perhaps its training and certification system, and maybe because it actually worked! Many organizations still eagerly tell me how many certified scrum masters they have as a metric of how improved they are.

## Customers are people too

There's an often forgotten milepost at this point, a strange little book called The Cluetrain Manifesto from 1999. The cadre of authors posited that the web was rapidly breaking down any geographic barriers and asymmetric strategies that enterprises used to retain and cajole customers. Things like reviews in Amazon

and using eBay to find anything you wanted across the world broke down long cherished strategic controls companies relied on to maintain market share. It was a sort of pulling back of the wool and empowering customers to be smarter than the octopus global-nationals, as we called them back them.

Cluetrain concepts were much toyed with throughout the 2000s, with companies investing much blood and treasure in capturing market-share in, ahead of monetizing.

Paying attention to what people were doing with your software and improving the software to keep hold of their eyeballs longer was a popular business, and it still is. There's an ever growing pool of revenue in never-ending conversation markets. Last quarter alone, Facebook earned $9.3bn in revenue with in $3.9bn profit.

## In the land of eyeballs, the profitless win

Getting to those kind of eye-popping profits required new thinking when it came to both builders and users. As companies like Google, Netflix, Facebook, Amazon, and numerous others who lost to the buzz-saw of product/market fit built out their businesses, often their only success metrics were user growth and retention. They had to create exceptional software.

To do this, these companies competed on features, on the exceptionalism of their software. They had to start releasing software every week, if not every day to compete. As one of the Agile Manifesto principles put it: "Deliver working software frequently."

Of course, having the software actually work most of the time was important, as Twitter early on showed, somehow surviving, perhaps as the world's first example of the "move fast and break things" boast. Faced with the need to release software on demand, often daily, the enterprise approach of doing monolithic,

gut-wrenching releases wasn't cutting it. The developers had to start thinking about and how their software was managing in production.

## Programmable infrastructure

A common story from this era is the fateful day one of the programmers is selected to "run the servers." Shifting over to "ops," the programmer either goes mad, or starts doing what any competent programmer does when faced with a new problem: procrastinating and drinking. A few weeks later, they look at all that infrastructure as something to program, and start coding.

For me, a 2019 talk by Andrew Clay Shafer codified this thinking right around the time it was codified into DevOps. To a room full of agile lords and ladies, he proposed something wild and crazy: what if you were responsible for how your code ran in production? Perhaps you should start to understand, embrace, and improve that phase of your software's life.

This implied focusing on the people in the software development process and how they work together and behave. The people are just as much a part of the application as the software and the hardware.

The idea of a "blameless post-mortem" is a good illustration: in innovation mode, things are going to break and go wrong as you charge into the unknown. Systems will go down catastrophically, but you can't simply give up, and punishing people just takes you back to the overly cautious state where software is released infrequently. So, as described by the Google SRE book, you instead celebrate failure, even telling the entire company the harrowing tales of what went wrong and, importantly, how you fixed it. Of course, once fixed, the key is understanding the problem well enough to put new policies, practices, and technology in place to prevent the problem from happening again.

## Software Defined Meatware

As this type of navel gazing continued, organizations once again discovered that most of the problems were caused by errors in the human systems they'd built, the meatware. Technology was an issue, to be sure, and there's a parallel story about how the evolution of what we now call "cloud" provided an ongoing arsenal for all this, with exciting distractions along the way with names like J2EE, rails, and WS-Deathstar.

People, thought, were still the consistent problem. They just seemed to keep screwing up all this agile stuff, if they were actually doing it at all. Most still clung to the false comfort of big upfront planning and its illusory promise of hitting The Date.

You'd see the effects of this backsliding in instances like the US's rollout of healthcare (saved by, ironically enough, by a bunch of "cowboys" from out west. The private sector was, and is, no slouch at resisting agile either: they're just good at hiding it. The difference between them and the government is that enterprises can change more quickly when they're threatened. The "culture" at enterprises is more hopeful, perhaps, at least once backed up into a corner.

Just as the goofy social companies of the 2000s had to compete on innovation, large enterprises now feel the pinch from the numerous ankle-biting disruptors that are having a good go at eating the incumbent's lunch.

You see this reflected in executive comments in numerous quarterly calls. Some of them toss-up effortless word-salads of "digital" and "omni-channel," but others have clearly considered their strategies and are applying a software-first approach to business. While they may not know exactly what to do, most executives know they need to start doing something. As JPMC's CEO said a few years back: "Silicon Valley is coming."

So that's where we are now: from Chrysler's HR system, to keeping

Twitter up, streaming videos and sharing pictures of cats, to the very real need of old school multinational, global enterprises to compete based on software. Surveys show how shaken executives think the situation is, with many doubts that IT's not up to the task of *transforming* to the point where they can reliably create, refine, and run software. They know from experience that outsourcing doesn't work, so they're looking at their people, organizations, and technology. There's early indicators that it's working - tales of using this new software defined business approach to insurance companies cutting the claims process from a week to less than a day and doubling the industry sales average - but there's a massive amount of work left. Hopefully, we won't back-slide this time.

Originally published in The Register, October, 2017.

# You, yes *you*: DevOps' people problem

You've no doubt heard of DevOps. This is the process of getting developers and sysadmins working together closely on the same team to support a company's custom-written software.

I know, I know, Dear Reader: you've been doing this ever since operating that AS/400; no one really needs weekly releases; and, of course, the favorite: "this is just the current way for consultants to make money."

All signs point towards DevOps being not only all those things, but actually A Thing on its own. Ever since starting my career as a programmer, and through being an industry analyst, strategist, and, now, marketer, I've been motivated by the quest of learning how to improve the software development and delivery process. DevOps seems like the current, best method.

## What has the IT department ever done for me?

The primary motivations for doing DevOps are to ensure application uptime (usually for mobile and web apps) while at the same time ensuring that you can release new code to production, basically, at will, usually weekly or daily. Presumably, a company would like these benefits to be more competitive with its custom-written software, both with more features, but also by taking advantage of short, user-interaction feedback loops to constantly tweak their apps to perfection.

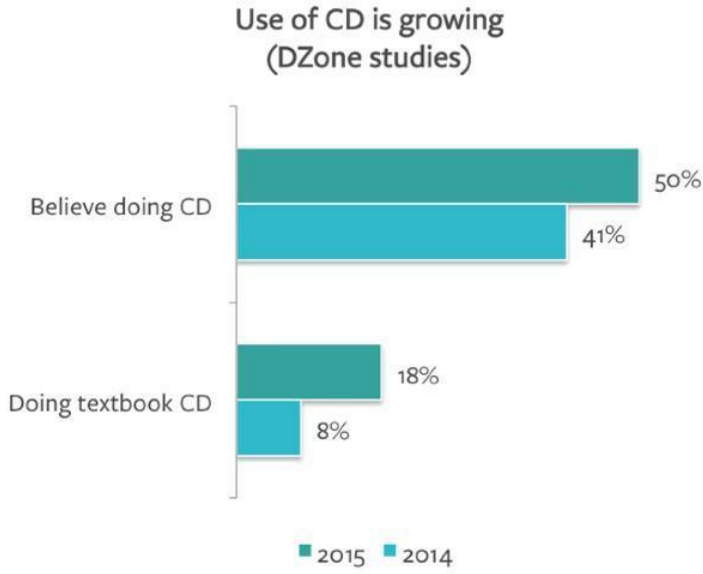
Things are not too joyous when it comes to IT actually delivering on this dream of helping companies innovate. When I want to gin up an excuse to drink heavily, one of my surveys to read at is one from the Cutter Consortium. In 2013, the survey asked if IT was a "[k]ey enabler for business innovation." At the time, 56% of respondents said yes. A year later, it was 43%. As you might be guessing, dear reader, come 2015, it'd dropped to 31%.

When it comes to innovation, over 3 short years IT has plummeted in usefulness. To put it bluntly: IT sucks.

As I've studied DevOps over the years I've found that DevOps is the "how" of solving this problem: a process and mind-set. Continuous delivery is the "what": the "tool" you put in place. This is why I look to continuous delivery as a tracer for DevOps adoption.

## You're doing CD? Yeah, sure...

Continuous Delivery (CD) is yet another one of those things that most people say they've done since the days of mainframes ... but reality is usually different, as seen by one study:




**From DZone's 2014 and 2015 CI/CD studies.**

A fair number of people think they're doing continuous delivery, but when compared to the textbook definition, they're more like dabblers, picking and choosing practices that are easiest and leaving out the rest.

While these numbers are low, the growth year-over-year shows rapid change and progress. There is a strong desire to improve, if only organizations can figure out how.

## Rub Some DevOps On It

Beyond the tracer of continuous delivery, there's some fresh industry survey data we can look at to see if DevOps is actually a thing. Gartner fielded a survey last Autumn; while it's all too easy to dismiss them as conservative and backward looking, isn't that

exactly the kind of thing you want when asking if DevOps has gone mainstream?

While it doesn't have the muscular impact of a year over year study, Gartner's September 2015 survey of 383 respondents showed good momentum for the adoption of DevOps: 29 per cent were actually doing something with DevOps, with 16 per cent taking a DevOps approach in production and 13 per cent piloting it. While small, those are impressive numbers for something as new as DevOps.

## Trying is the first step to failing

While longer-running bodies of work like the always excellent, annual Puppet Labs DevOps survey are showing that the ideas work, things are not so rosy when it comes to putting DevOps in place. More often than not when I've worked with groups that want their software processes with DevOps, they underestimate the amount of organizational change needed. They view software more like building a Lego kit. Creating good software is more like inventing Lego all over again, each time. Fostering that kind of continuous learning requires putting the process in place that creates metaphoric "innovation factories." DevOps thinking describes much of how those "factories" run, which is often much different than the status quo.

What I've learned is that it's a meatware problem: people's default to resist change is what holds back transforming to a DevOps mind-set. This is why the DevOps cult leaders go on and on about "culture."

The problem starts with managers who often assume they can just throw a copy of The Phoenix Project at their team and expect them to "do the DevOps." Instead, managers need to change incentives and behavior to lubricate change. But it's not just managers – staff needs to get with the program too. You know, you might actually have to go talk to programmers! Or worse: sys admins!

*Originally published in* The Register, January, 2016.

# The Slow Ascension of Agile

After roughly 20 years, agile software development has wheedled its way into most every developer's mind as The Way Good Software Is Done. Like flossing, while we can all agree agile is a good idea, we're not quite up to snuff on keeping all our teeth in our heads, so to speak.

A recent Gartner survey had 37 per cent of respondents saying they were doing agile, while 45 per cent preferred to float along with the traditional "waterfall" approach (the remaining said they were doing "lean," "iterative," or the always delightful "other"). While this isn't world domination, a 2015 report put waterfall at 56 per cent.

Survey data like this can be dicey, and it's best to treat them more like a wet finger in the wind than as rigorous science. That said, the wind seems to be blowing in agile's direction.

"I think from a tactics perspective, Agile is increasingly a 'solved problem'," said Forrester's Jeffrey Hammond when asked about agile adoption in the industry.

"We know many practices that work, and that have been well proven in the field," he added.

Proven as those techniques may be, once again, loose meatware is catching in the gears of progress. As Jeffrey adds, "from an adoption standpoint, Agile is a 'work in progress' mainly because Agile is as much about cultural transformation as it is tactics." Cultural transformation: it'll get you every time.

Indeed, looking back at that 2016 survey, you see that while easier practices like unit testing are widely practiced, onerous practices

like continuous delivery and pair programming are mostly ignored by the buffet agilists. Agile is taking its time along the innovation curve, but one gets the feeling that the down-slope folks are methodically being routed, if only through the slow-but-steady siege tactic of retirement.

## Beyond The War of the Story Cards

Early on in agile, there were some vicious battles around defining The One True Agile, especially as Scrum rose in popularity. For the most part, these battles were case studies of the narcissism of small differences, though mentioning "agile in the large" practices like SAFe can still be relied on to pop a true believing agilista's neck veins.

Several schools have descended from agile, primarily in the form of "lean" and "DevOps." In practice, these schools should be thought of as types of agile – at most, extensions – rather than so philosophically different as to be called distinct. They're nothing to start a holy war over.

"Lean," as its name would imply, comes from Lean Manufacturing, the continuous learning and "waste" removal process invented and perfected by Toyota. When you put lean Software Development practices in front of Lean Manufacturing people, they react similarly to how I imagine the French would react when presented a baguette à la Piggly Wiggly. The same idea is there, but it's been transmogrified by a new set of hands. In practice, lean software development focuses on putting a small batch approach in place, releasing software as frequently as possible to limit work in progress, and using an end-to-end mindset to discover and eliminate waste.

The idea of "waste" is perhaps the most intriguing and novel part of lean software development: unless an activity adds value for the customer, it's eliminated (más o menos). Once you start asking: "Will this help the customer?" many of the tasks in the so-called

SLDC melt away like chicken fat under high heat. A notable variant of lean is the Lean Startup method, which seeks to discover the market/product fit for any given piece of software. This is the school of "pivots" where you continually evolve your software, observing how it's used until you figure out something that people will pay for, or at least use.

## Enter the two pizza team

*El Reg* commenters' favorite, "DevOps," is the last descendant of agile, building from lean along with a dollop of its own special sauce. A canonical version of DevOps isn't a fully "solved problem" yet, but we're getting close. Originally, what became known as DevOps was solving for two things: enabling frequent deployment of software (many times a day) and maximum uptime. For most people, the idea of deploying software daily is the gut-proven opposite of "maximum uptime." However, the disciplined, small batchmentality practices of DevOps have been showing that the two can be done in tandem.

One of the key components DevOps adds to agile is the idea of a small, cross-functional team rather than "silo'ed" teams. Agile-minded developersland-grabbed QA early on, but mostly stopped there. In contrast, DevOps looks to gobble all the roles, from developers, to operations, to designers, product managers, and whatever other role is needed to make sure you can ship useful features frequently and keep the stuff up and running.

Though there are numerous roles, the teams are small. As Ben Terrett, former head of the UK Government Digital Service, put it: "[T]he best way to do this stuff is to get a multi-disciplinary team of people in house – designer, user researcher, developer, content person – you're talking a team of about twelve people." In Amazon terms: no team should be so large that it needs more than two pizzas for dinner.

# Admit it: you have no idea what's going on

Recently, I've been asked several times to address when waterfall is a good choice. The answer to that question is good insight into agile's trick. Waterfall is fantastic if you know exactly what you want to build, up front. What a wonderful project that'd be to work on!

As most of the people who develop software find, however, very few people know exactly what needs to be built ahead of time, least of all the actual users and customers.

Agile, instead, builds its thinking and processes around the assumption that we can't know what the software should be until we start deploying it to actual users. Doing so isn't easy, and while adoption has been slower than you'd expect, as was said long ago: if you are made to wait, it is to serve you better, and to please you.

*Originally published in* The Register, July, 2016, *with a more salacious title.*

# Software devs' new mantra: Zen dogs dream of small-sized bones

One of the primary principles of DevOps is moving from large software releases to a series of small batches.

What do we mean by "large"? Six-to-12-month (or longer) projects that follow the infamous "water-scrum-fall" model. While development teams may create builds weekly, the code isn't deployed to production and used by actual users each iteration.

"Large" has strong appeal. It optimises the planning, development, and deployment phases of software. Planning teams want time to gather all the requirements and detail them out in exquisite documents, making The Boss feel like they'll get exactly what they want.

Developers and QA nowadays tend to like working in an iterative manner, chunking their work into one- to two-week increments. However, operations feels that the highest chance of failure comes when you deploy, so why not minimise deployments? Each group creates the process that makes them feel like they're doing their job well.

Despite the warm and fuzzy feeling you get from a well planned, staged out process, taking a small batch approach is showing more success. As one IT manager at a large organization puts it: "We did an analysis of hundreds of projects over a multi-year period. The ones that delivered in less than a quarter succeeded about 80 per cent of the time, while the ones that lasted more than a year failed at about the same rate."

If that's "large", what do I mean by "small"? Here, it's reducing that entire cycle down from six to 12 months to a week - or even a day. And yes, you back there with your hand up: this means deploying a lot less code each time, hopefully just a handful of changes, or even just one.

Focusing on smaller batches is mostly about reducing key areas of software risk. Those risks being:

1. Bug swarms: If I have a week's worth of code vs half a year's worth of code, and something goes wrong in production, there's a much smaller set of code to diagnose and fix. This also speeds up your ability to deploy security patches.
2. Useless software: The biggest risk in software is creating software that users don't find valuable but that's otherwise perfect. With small batches, because you deploy each iteration to users, you can easily figure out if they find the software useful. And even when you get it wrong you've only "lost" a week (though, I'd argue you've "won" in gaining valuable learnings about what does not work).
3. Stymied innovation: Coming up with new ideas can take a very long time if you have to wait six months to try out new ideas and see how your users react. Instead, if you deploy a series of small batches, you can experiment and explore each week, hopefully getting into a virtuous cycle of steadily discovering new ways to delight users.
4. Budget overruns: A small batch mentality avoids "big-bang bets" that require a massive capital outlay at first and then a white-knuckling 12–24 months of waiting before shipping the code. If you're only focused on the next few releases, finance can adjust funding either up or down as needed. The existence of government IT projects going over budget serve as an example here (though, I assure you, private industry can be just as bad: they're just better at hiding failure).
5. Schedule elongation: Projects that don't force shipping can often find themselves forever stuck with just "a few more

> weeks" left before shipping. There's always new features to add, more hardening to do, and then it's the holidays all the sudden, and you've got a good month of downtime, which is just long enough to think of still more new features to add. Without an emphasis on shipping every week you eventually slow down.

Small batches let you improve the quality and usefulness of your software by creating an ongoing feature experimentation process. Small batches mean greater control over the budgetary and planning aspects because you can spot problems early on and act accordingly, theoretically – at least – after each weekly release.

Overall, The Business feels like it has more opportunity to manage, conferring upon those in charge a sense of empowerment and control. No more nasty shocks from IT.

But, and here's the problem: small tends to cause a loud clunking noise in the minds of The Business and of The Boss. These folks are not always comfortable moving beyond that big stack of promises. "What, Henderson? You want me to do small?! At Waddleforce &co, we're all about BIG!"

How do we in IT circumvent this and sell the merits of "small"? I suggest discussing small batches in terms of risk management and the "optionality" that the approach creates, something that business-heads usually understand and value.

As IT proves out the small batches approach, then the code crowd might have something to teach the suits.

*Originally published in* The Register, February 2016.

# So you're 'agile', huh? I do not think it means what you think it means

What if I were to tell you that we knew all the best practices for software development? That they've been proven by actual industry use over the past 25 years? But that, oddly, these practices are not widely done? Well, if you read these pages, you'd probably say: "Sound about right."

Agile is much spoken of, but not as broadly practiced as you may think. It's as if we all knew that the best way to cook a fine T-bone is to first let it come to room temperature – perhaps with a healthy handful of the de Camargue – but instead we just regularly yank it out of the fridge and throw it on a cold pan.

## O rly? You've been doing agile since AS/400s?

When I talk with large organizations, all too often they legitimise themselves by telling me how many certified Scrum Masters they have. From the hundreds I hear about, they've setup some kind of factory that's just rolling them off the line. Now, there's nothing wrong with scrum or certification, but it is an odd thing to use as a marker for agility. What matters more, of course, is if the developer teams are actually doing it.

Commenting on his team's experience doing agile, Lt Col Enrique Oti explained the situation this way: "'Agile'. That word should not

be used in the government. It's used everywhere. Everyone in the government now does agile training. Every organization I go to [claims to do] agile development. I went to an organization recently who'd been working on a project for six years doing agile. They had a Scrum Master! And I said: 'When does your user ever see your code?' and their answer was: 'Never.'"

Although he's talking about the US government and military, in my experience his statement applies many large organizations.

Surveys back this up as well, such as Gartner's annual agile survey. What's astonishing about this survey is how honest respondents apparently are: looking over the results you quickly get a picture that just about half of the organizations surveyed are agile. Summing it up this past June, Mike West points out that 41 per cent of respondents were doing agile with 41 per cent more doing waterfall, and the rest doing other methodologies.

Perhaps it's the $1,295 price tag on these 15 pages of astounding findings from our friends in Stamford, but a shockingly low amount managers seem to be sandbagging on finally moving to agile after a quarter of a century. (Of course, for the cheap, the DevOps Reports can get you a free version of these findings, plus a wonderful selection of Portland hipster visages.)

In general, then, it's wise to be sceptical of any claims about an organization being agile.

## Wagilefall

Even when development teams have nailed agile, pumping out builds weekly gleefully (or, monthly for the languid), as Oti points out above, they often are not able to actually deploy their code to production.

In cases like this, as venerable agile expert Israel Gat told me: "organizations learned how to fake agile. Many of the agile implementations I witness are actually waterfall on top of agile, waterfall

using agile terms." The teams are speedily working through things, seemingly moving fast, and that's what agile's all about, right? They're an agile dynamo trapped in a decadent waterfall process: wagilefall.

Mismatches like this are widespread and, to my mind, are a massive reason why DevOps is so attractive to those who dare enter that labyrinth of definitional confusion. It's been illustrative to think about this divide between fast-moving developers and never-wanting-to-deploy ops teams as a "wall": developers throw the build over this all to release management, walking away dusting off their hands as if everything is done.

Rather than bricks, this wall seems to be built out of help desk tickets. Filing away those requests to set up staging and production environments, let alone even the simplest resources like a licence for an IDE. And when it comes to actually deploying a build to production, file all the tickets you want, bub, you'd better schedule up some meetings if you want something that ground-shaking.

## But think of all those 'nice gates'"!

Resistance to change is, of course, not a new occurrence when it comes to IT. That said, again, the memos have been circulating for a good 25 years. Despite this, it's wise to be empathetic to staff who see going agile as simply more busy work for them. Donna Fitzgerald quoted one of her clients as saying: "It meant throwing away everything I spent years building. All my nice gates and all my vast number of required documents. It meant changing out the tools we use. It also meant that we needed to change our mindset about what was important and what the organization actually wanted us to do."

Yes, indeed, there's much work to be done after the old artefacts of comfort are thrown out. I recently had a conversation with a similarly beset person in a large organization. There were so many

agile methodologies in practices, along with the existing "waterfall" processes, that a team had been put together to map and rationalise this rat's nest of processes into a unified handbook of sorts. You can imagine how that project was turning out.

## Blame management

As ever, one of my core theories about improving how software is done is that, more than likely, management is largely at fault for previously hollow victories. In addition to the numerous reasons for staff resistance, management is often unwilling to follow through on the changes needed to bust through a wagilefall.

Do the infrastructure Morlocks hide behind a wall of tickets? Well, the developers aren't going to be able to change that: they'll need management to come in and burn down that wall. Are you getting ensnared in compliance and enterprise architect dead-ends? Again: management.

There are plenty of enlightened managers, but you'd be wise to figure out if you're working for them whether you're on some sort of path to agile awesomeness. If you find them wanting in vim, perhaps they're kind enough to take suggestions – if you're lucky.

## Sure, it'll get worse before it gets better

When I talk to people who have reached the other side of going agile, they tend to find that they're accomplishing the same goals of quality software, even with the same benefits of governance and discipline. They're just able to do those tasks more efficiently. Instead of doing audits after the fact, staying up late into the night and working through the holidays, compliance people can automate much of the raw information collection and leave work on time.

With smaller chunks of code in each cycles, operations staff realises that diagnosing any errors in deploys is more straightforward. Project managers, previously beset with putting together complex status reports that no one seems to ever actually read, find much more meaning in their work.

Somewhere in there, you'd hope, there's also an improvement to the actual software and the end user's experience, which usually trundles along for the ride. The studies tend to bear this out. As West put it commenting on the agile survey: "Successful agile organizations show significantly higher use of unit testing, DevOps, continuous delivery, continuous integration, test-driven development and refactoring, when compared with unsuccessful organizations."

More importantly: please, let the T-bone rest for a while before cooking it. Otherwise, you could have just gotten by with a much cheaper flank steak.

*Originally published in* The Register, December 11th, 2017.

# Pair programming: Oooo, oooo, that smell...

Of all the agile practices out there, "pair programming" is the one that elicits the most heckles, confusion, and head-scratching. The idea is that rather than having one person sitting at a screen, coding, you have two programming together. Those who practice it speak of it like most people do of their first time at Burning Man, while those who have never had the "experience" just can't see what the big deal is.

While finding them are hard, over the years studies of pair programming have consistently shown that it's an effective way to keep bugs out, write code faster, manage the risk of developer churn, and actually raise morale.

But – really? Looking at surveys, I'd estimate that somewhere south of 20 per cent of people do pair programming. If pair programming was so great, why do people find it so odious? I mean, who wants to work so close to someone that you can smell the effects of coding?

And as if it wasn't enough to keep that foetidly in the developer cubes, it's been wafting into the server room despite those cyclopean fans in there: operators are starting to pair as well.

## Four eyes are half the productivity as two?

The theory behind pair programming is straightforward. Programming is difficult and error prone: It's much better to have a buddy

helping along. In addition to actually coding together, it sometimes means having one developer write code and the other write tests right next to each other, in coordination. With two heads together, the thinking is that you write less bugs and get better test coverage.

Indeed pairs in studies over the past 20+ years have consistently written higher quality code and written it faster than solo coders. So, while it feels like there's a "halving" of developers by pairing them up, as one of the original pair programming studies put it: "The defect removal savings should more than offset the development cost increase."

## Safer coder killing

If the pairs rotate frequently, the theory says you'll get better diffusion of knowledge across the team: no one person builds up a fief of knowledge around, say, builds, or how the "Print Invoice" function works. This means there's a lower "bus factor," helping protect against team churn and brain-drain.

Large organizations I talk with - who're all trying to figure out the footwork for that "digital transformation" dance - use rotating pairing as a way to spread new technical knowledge, but also change that oh so mysterious "culture" in their organization.

## People actually like it

Much like alcohol and black coffee, pairing tastes awful at first… until you start imbibing of it repeatedly. In most of the studies, and the feedback I hear from organizations doing it nowadays, pairing practitioners end up liking it after just a few weeks. At first, true, the usually solitary programmer has to, you know, *talk* to someone else. They even have to get used someone else corrected them – horrors of all horrors!

But, with a rigorous enough schedule that allows for breaks and bounds the programming time to normal 9-to-5 schedules, most people end up liking pairing after a while. It only takes a few pints to dedicate your life to it.

It's hard to say why people like it more, but I suspect it has something to do with the fact that humans, fundamentally, like being social, so long as it feels safe. Also, most programmers and operations people take pride in their craft: they want to do good work (despite what those overflowing tickets queues are doing to them). If pair programming increases code quality, there's more to be proud of.

Managers of these programmers should also like the quality, speed, and predictability of pairing.

That predictability comes from an interesting side effect of how exhausting pair programming is. For one, it's harder to goof off – er, "research" – and attend meetings when you're pair programming. As the man from downtown said: "Always Be Coding."

And, on that kind of schedule, developers are straight up pudding-headed after seven or eight hours of pair programming. As one practitioner put it: "This makes pair programming intense, especially at the beginning. At the end of the first day, I couldn't go home. Before I could face humans again, I put my phone on airplane mode, ignored my usual online accounts, and went to the gym for two hours of self-imposed isolation."

Developers can only pair so long. They have to stop, so you just close up shop at 5. No more playing Doom until 10pm and then coding – er, I mean "working late".

It can come off as sounding a bit like nanny-management, but pair programming seems to induce developers to actually do the work.

## Yeah, but... no

While the research is sparse (and, really, when it's "n=whatever students enrolled in my CS class," it's a little fishy), from where I sit and what people keep telling me, pair programming works. Should you be doing it all the time, though?

I've heard practitioners say that you should at least do it for complex, difficult tasks. If it's some routine coding or operations tasks, then pairing may not be the nitro-charge you're expecting. Indeed, one of the studies suggests that pairing is the most beneficial for "challenging programming problems".

Put another way, if the task is "boring," maybe it's better to solo it. Still, I can't help but think that it'll be the boring tasks that end up biting you, especially when it comes to pair sysadmining. After all, how many systems have come down because of the boredom of DNS configurations?

*Originally published in* The Register, October, 2016.

# Why largile's for management crybabies

There's a stink growing out there in agile land: a debate over how to scale up agile in large organizations. Should we put frameworks like SAFe or the most awesomely named DAD in place to scale it? How about we do LeSS?

These "agile in the large" frameworks have been on the ascent in recent years. A 2015 Gartner survey found that DAD, SAFe, and LeSS had been adopted by 10–12 per cent of organizations. SAFe was driving the most interest with 34 per cent of respondents checking it out.

As ever when the adoption of agile matures, there's much gnashing over whether things are being done properly. On the one side are the agile poets and on the other are the agile formalists. The poets like the emergent, dynamic, small-batch nature of agile and don't want to bind teams to locked-down rules and requirements to "align" with the rest of the organization. The formalists want to put in place and document processes that ensure that thousands of people can work in lockstep on software.

## The infantilism of management

Compared to staid, friendly discussion, there's a particularly vitriolic conversation going on about SAFe now. Words like "infantile" are being thrown around! The fear is that SAFe focuses too much on keeping existing IT bureaucracies in place for the sake, you might say, of giving management something to do. Instead, the

agile poets say the focus should be dramatically changing how the organization works as a whole, not keeping all the separate groups in place that need constant "alignment."

It's this notion of "alignment" that acts as good dipstick for the discussion. If there are hundreds, thousands of people working on a unified portfolio of software, there's the chance for lot of coordination. As an example, in his recent book on scaling up DevOps, Gary Gruver uses the example of putting omnichannel retail in place, a seemingly simple, but very complex system spanning multiple back ends, shipping, and in-store systems, not to mention the mobile and web apps buyers use.

## The alignment anti-pattern

"Alignment" then is the need for groups to come together and plan out how their sub-components interact together to create the entire system. Next thing you know, you're scheduling meetings, writing Word docs, and spending weeks integrating various systems together. A typical straw-person nightmare of slow software development.

The critics of "largile" are worried that this focus on alignment should instead be on removing the need to coordinate and align groups, at least manually. At a technical level, this means removing as many dependencies as possible and, usually, giving each team more responsibility. The whole idea of DevOps is an example of dependency erasing at the process level: by collapsing together the roles of development and operations, you strip out the time and hand-off errors that occur when you throw the application "over the wall" of the operations. The fear is that you end up with exactly the same process as you had before. Renew those Word and PowerPoint EULAs!

## Requirements cathedrals

A few anecdotes from my flâneuring about in the enterprise world illustrate this alignment anti-pattern. I spoke to the people at a large, US health insurance company recently trying "the new way" of developing software. They started the project with several hundred pages of requirements that business analysts had built up like a perfect cathedral. After throwing this pile of paper to a unified, balanced "two-pizza team" who walked through the user problems and how to start the discovery cycle for solving them with weekly builds, most of the cathedral was dismantled and ignored.

Once this team started deploying software weekly and studying how the user interacted with the software, they learned what was actually needed and changed the requirements appropriately. The team removed the need to "align" with others in their organization. Sure, there were external systems to cope with, but removing the need to coordinate and take ongoing input from parts of the organization that weren't close to the actual users speed up the schedule tremendously, delivering months ahead of time.

Taking a similar approach, a large bank scoped down the coordination needed across organization by pushing responsibility down to the team level. They were able to speed up their delivery by 57 per cent (so precise!). At a micro-level, the act of pair programming removes the need to "align" with code reviews as they happen while the pair codes.

The hope of much of the container and cloud crew nowadays is that cloud automation removes a huge amount of infrastructure, networking and security functionality, removing the need to align on those glide paths. Looking at Gruver's recent book again, the idea of standardizing on CI/CD pipelines is another tool to automate alignment, removing the need to align in the meat ware levels.

Time and time again, the goal of the agile poets to "remove" the need to align, not facilitate it. As one large European bank put it:

When we were doing big design upfront, downstream changes had to go through a rigid change control process. We wound up being busy with our own process rather than delivering value, and either we didn't deliver or we delivered late.

## Don't pave the cowpaths

Now, I don't think proponents of largile practices would say any of this is their intention. Indeed, I wouldn't be surprised if agile in the large practices didn't marble my own examples above. The agile poets would argue that, regardless of intention, the effect is to once again prove out Larman's Law: the bureaucracy will be in place, just with new words describing the infinite process gates and alignment drones.

I'm not convinced perfectly either way. As usual, someone's probably spun the dichotomic dial so hard it's busted off. I've spent recent years studying the seemingly impossible task of scaling agile techniques up to large organizations. As the DevOps Reports have found, organizations with 10,000+ employees are 40 per cent less likely to be high performing than organizations with 500 or less employees.

Clearly, it's hard for large companies to improve how they do software. There are real needs to align and coordinate between organizations. Thinking back to that omnichannel example, one team of four to 12 people can't build and maintain all parts of the system. The goals of (buzzword alert!) "microservices" try to address this alignment problem at an architectural level, but that school of thought is relatively new.

This is still uncharted territory for many, and I suspect it'll be the usual situational problem: how you scale agile will depend on your organization peccadilloes. A recent Gartner study found that a meagre 27 per cent of surveyed organizations are using agile approaches for "most or all" of their projects, a shockingly low

number. Before these organizations even worry about doing agile at scale, there's plenty of work to be done at the team level.

One thing is for certain: you don't want to simply keep doing the same thing. If you find that the org chart, the flows of information and – gasp! – approval processes are exactly the same as before your largile transformation, you're probably doing it wrong.

*Originally published in* The Register, November, 2016.

# “Do the DevOps?” No thanks! Not until a ‘blameless post-mortem’ really is one

What drives organizations to change their ways? What’s the match that lights the powder keg of actually doing something new and different in IT? That’s the question I usually get from organizations that want their approach to software to be more “agile”, who want to go through “digital transformation”, and, yes, “do DevOps”.

Despite glee about cleansing themselves with the buzzword of the week, they feel like they can’t get their organization to go along with it. While upper management might be pounding the table, shipping in crate after crate of DevOps Handbooks, the rest of the organization languidly keeps to their old waterfall ways of doing software – maybe wagilefall, if you’re lucky.

How do organizations that “go agile” actually motivate themselves to get out of bed in the morning?

## Kick them in the pants

It’s common to trickle down blame to individual staff and the so called “frozen middle” who keep existing processes in place. Higher level executives, though, aren’t much better according to a recent Altimeter study. Of the 500 executives surveyed, only 37 per cent said their organization was proactively investing in “digital

transformation" (let's just assume that means "improving how we do IT around here to help run the business better"). Put another way, 63 per cent seemed content with their IT.

In my experience, most organizations who are looking to improve their software capabilities are motivated by a sudden, unexpected, often fierce competitor. Many insurance companies, for example, were spooked by Google's foray into car insurance. Fear motivated them understand what it would mean to have their market changed by companies like Google.

While the search giant decided to shut down the experiment after about a year, responding to this digital apocalypse premonition left several insurance companies with new capabilities and agile aspirations. They'd been given a kick in the pants that woke them up from their "if it ain't broke, don't fix it" stupor.

For most businesses, this kind of external threat is required to start any type of IT improvement plans, let alone something as high-falutin' as "digital transformation" or even DevOps with all its needs to radically change corporate culture. Without that well understood, and felt threat from outside, driving change can be too difficult for more organizations.

## We fear change

Of course, simply kicking in a pair of well pressed slacks will only get you so far. There's the entire rest of the company that needs to put on their big boy and girl pants and find the will to change as well.

Below the higher levels of management, a more risk-averse culture thrives. Middle-management's job is to keep things stable, to keep the building from burning down. Which is all pretty easy if things never change, hence, "frozen middle". When looking at new ways of running IT, middle-management often sees only the possible

downsides. Never mind all this "blameless post-mortem" stuff, I'm the one who'll get blamed and punished, they quickly realise.

Worse, in organizations that desperately do need to change from a large, multi-year delivery cycle for software (read: "waterfall"), the risks actually are huge. While big bang projects may seem like a gallant steed at first, with such long release cycle - on the order of years - these projects can easily blow up: they're liking using older infrastructure and application layer stacks and will also succumb to the trap of delivering perfectly the software that was specified three years ago and is no longer relevant today.

In such big batch projects, as one "change agent" put it: "A mistake could cost $100m, likely ending the career of anyone associated with that decision."

We can all agree that ending your career at middle-management wages is no good; unlike with executives, there's no metallurgically coloured parachutes that land you safely in a lovely little chalet while the plane hurdles into the mountains.

Below that managerial permafrost, staff are full of anxiety as well. A recent survey of 1,000 managers in UK found that 49 per cent per cent of employees quake in their boots when "digital transformation" is mentioned. It's little wonder given all the claims of how new technologies are going "optimise" staffing needs in IT, let alone how radically different all this new free-loving agile stuff is going to be.

As I mentioned a while ago, managers I talk with say anywhere between 30 to 70 per cent of staff "don't make it" to newly transformed organizations.

## Stop hitting yourself

Based on the recent bevy of large orgs actually improving how they do software, clearly, there's hope for getting over all this

trepidation. There are enough examples of large organizations switching over to a more agile, even "DevOps-y" approach to software.

What's clear is that in the best cases, senior management champions the change, often assigning an executive as "Chief Trouble Maker", as one executive described himself. These organizations also focus on creating safe spaces for innovation and change in process, usually over the course of several years. It's tempting to think that you'll figure out all this transformation overnight, but with a large org, it's better to temper expectations to reality. A misstep on managing the expectations of how long it'll take could easily kill moral and nuke your plans early on.

It's important to actually pay attention to winning the hearts, minds, and KPIs of actual individuals. As that fear of change and sense of nothing but downside shows, people need to build up trust in a new process. External cases and decrees of The Good DevOps News aren't going to help much. The most successful persuasion is building up a string of internal success stories – those skunk works teams that can then be marketed as "look – see – it does work here!"

Staffing those teams is tricky. On the one hand, as Brian Gregory told meback when he was heading up the switch to agile and DevOps at Express Script, you have to choose a team of mavericks who want nothing more than to try new things and take risks. On the other hand, if you create too much of a "10x developer" team, everyone will look at them and think "well, that's fine for them, but I'm a *normal*." As a manager at an insurance company told me more recently, "when you get to the end of the pilot, you want co-workers to look at your team and see someone they can relate to."

## Get ready to battle your own doubts

Finally, you, who'll be, let's face it, the "change agent", are going to go through some rough patches. You've got to line up some trusted

peers and mentors who you can call to pick your sad-sack staff up off the ground when things go poorly. There's going to be more meetings with more smiling jerks than you've ever encountered. Few people will fight for you out of the good of their hearts and many will be looking for the right opportunity to slip a knife into your ideas. There's only so much of that annual bonus pool to go around, after all.

It'll feel like everything and everyone is against you. "You're in that valley of despair," as Opal Perry at Allstate told me earlier this year. Things seem dire, but with plodding success, she added, "then you start to come out". If you plan for the resistance to change, deploy some mind-hacking tricks, and start building up some proof of success from within your own business, well: if that doesn't work, there's always bimodal.

*Originally published in* The Register, November 17th, 2017.

# DevOps isn't just about the new: It's about cleaning up the old, too

As one of my coworkers used to say when confronted with The Latest Development Improvement Methodology: "Why don't you come down here and chum this stuff?" – except he used the language of a sailor.

In trying to implement the latest breakfast cereal agenda, DevOps, one of the primary chumming tasks is dealing with all your "pre-DevOps" software and services.

We call this "legacy" and it's more or less the result of too much unaddressed "technical debt."

The techniques for dealing with legacy never leave you feeling good: just like eating a box of cereal, over the kitchen sink, all the way down to the green leprechaun dust. But, there are some pragmatic ways of making sure legacy doesn't totally wreck your DevOps efforts to create more resilient, more productive software.

## Identifying legacy

First, if you're starting from scratch, with no existing software, with the crisp scent of Expo white board markers still lingering in the air, you have no legacy problems. Enjoy your tasks of creating legacy code for the future you! However, in most large organizations, you'll have plenty of legacy code and systems.

I use two tests to identify legacy code:

1. It's running the current business. The code is keeping the lights on, running however many decades of existing business process has gotten your company to where it is. For example, people often (factually) joke that the IRS is still running systems from the Kennedy era.
2. You're afraid to change it, mostly likely because it is poorly understood and has poor test coverage. This is compounded by Michael Feather's legacy code dilemma: to add unit tests, you must change the code. To change the code, you need unit tests to show how safe your change was.

For those lucky few who don't need to evolve their software (and their cursed users), dealing with legacy code isn't an issue. But the rest of us need techniques to manage the risk of working with legacy code.

## Quarantine the slow movers

As in dealing with any pack of zombies, the first thing you want to do is identify and then isolate as many of your legacy applications as possible so that you can ignore them, freeing up time to focus on the feisty ones. In enterprise architecture management, this means doing some basic portfolio analysis. And, sure, I bet you have whole teams of people who do this already… right?

They know all the applications you're running, the amount of money they bring in ("business value"), their expected life-span and end-of-life plans, have identified key stakeholders and developers who know not only the software but the business it supports forward-and-backward.

Yup, we all have that functioning at 110 per cent 'cause we're "enterprise"! And yet…somehow we can't do anything because of all these legacy systems pulling us down… Now then, ready to actually put some portfolio management into place?

First, figure out which of the 1,000's of applications you have are low value and not worth spending time on. Figure out how to stop worrying about them. The second wave of quarantining is to find applications that haven't been fully virtualized yet. With minimal changes, you can squeeze some resource savings (time, money, and attention) out of applications by virtualizing them.

After this, you're left with smaller set of applications that you care about. To some extent, you're admitting defeat with these un-quarantinedable applications. On the other hand, you now have plenty of work for all those change resistant folks you have who aren't feeling the DevOps breakfast cereal vibe, if that's a concern of yours. Now, that you've cleared out some underbrush, what do you do with the trees that are left over?

## Fork-lifting, strangling, and re-writing

The most common methods I see for dealing with the leftover legacy applications are to either attempt to move them to your new platforms and methodologies, introduce an API facade in front of them and slowly let them rot out as new code builds up behind the facade, or to start re-writing them.

"Fork-lifting" the application into a full on DevOps-driven, continuous delivery approach can work if the application was written to be, generally, self-contained and didn't depend on vendor-proprietary services or things like network file shares.

These are usually simple applications, and you're usually not lucky enough to have them live through the initial quarantine filter. This is often known as the "lift-and-shift" approach, and, as Forrester's John Rymer points out, this approach looks the easiest but has the worst long-term payoff. This is because simply changing how you manage the lifecycle of the application without changing

the application itself can limit the benefits of a DevOps-driven approach, namely, the ability to quickly add new features while maintaining a high level of availability in production.

In those instances where your new applications must use legacy software and services, you can use the "strangler pattern" to lessen the annoyance of legacy. While you may wish this pattern was named after the psychopath, it's named after the plant that slowly takes over trees.

The first step is to introduce a new layer of abstraction – an API or set thereof – that fronts the legacy services. Instead of calling back to that big database or ERP system directly, you call to your own facade on-top of it. That part is easy enough, and standard, the hard part is planning for the eventual rot-out of the old system. Judiciously, you start replacing capabilities in the legacy system with new code that's more aligned with your new approach to software development, using some mild routing intelligence behind the facade to figure out when to call the legacy code versus the new code. Eventually, as with the strangler vine, only new growth is left.

Finally, you often have to bite the bullet and just re-write it. While this is the most time intensive, and, if done slapdash, risk-laden choice, if done properly it gets you the frequent change benefits of continuous delivery driven by a DevOps approach to process.

With legacy code, there are no easy outs, or secrets. The most important thing is to be aware of that and not be bamboozled by people who are happy to sell you a perfect solution to your legacy "problems."

Often, the right answer is to carefully do nothing and instead to focus on your net-new software without letting your legacy software and processes drag you down.

This way of ensuring that neither the old or new approaches to software rocks the boat for the other is more of how I think of "bi-modal IT": decoupling those two parts of your portfolio so that they can independently evolve without negatively affecting the other.

*Originally published in* The Register, April, 2016.

# Change review boards are probably a no-op, at best

One of the more wickedly astonishing findings from the current DevOps Report is that change review or advisory boards have little effect on a company's performance. In fact CABs – as they are called – tend to slow down IT's ability to release software quickly and regularly, negatively affecting organizational performance.

I don't think many people would say they like or even believe in change review boards – except the architects on them … well, at least *some of them*,hopefully.

Nonetheless, if continued existence demonstrates faith in a concept, we in the IT industry seem to believe fervently in review boards: I encounter them at almost every organization I speak to. When IT moved more slowly and we were delighting ourselves with ITIL and other PRINCEs of process, review boards seemed like a good idea. After all, not too long ago we'd just emerged from the switch-over to the "distributed computer" (read: Wintel boxes) whose conclusion felt like finally bringing law and civility to the Wild West.

That huge mess of new hardware and software spawned entire clean-up crew industries in systems management, breathing new life into aging mainframe management companies once they had acquired the rascally newcomers like Tivoli. We certainly didn't want some runaway IT projects built by a bunch of cowboys who'd leave us city-folks behind to clean up the mess. We needed a process

to assure our future selves' sanity and which would allow us to get home in time to watch *Seinfeld*.

In recent times, though, the need to ship software more frequently has created a new set of expectations for IT and, thus has been a driver for innovation in the software release cycle. For many, IT's goal is now to ship software weekly, if not daily, giving their organizations the capabilities to operate like software companies. So a review board that itself meets monthly to look over a huge pile of changes becomes a massive road-block… and if they don't seem to be effective, why have them?

## There's no escaping reviewing

Of course, the trick is that "reviewing" is still occurring, but since everyone started following the Toyota Way principle of "lean thinking", the reviewing is now done closer to the actual work. Instead of relying on change review boards, the application teams themselves do peer review with some even going as extreme as doing paired programming. There are many practices and technologies that help accomplish the original goals of those review boards too.

Standardized testing is also done more and more by the actual application team and also has become highly automated. It's not like these fast-moving DevOps people are just shipping code gleefully, they're testing and reviewing at almost a nauseating level for old timers who enjoyed throwing the testing tasks over the wall to QA. A recent Gartner study on agile practices in enterprises found that 75 per cent of organizations were doing unit tests and a third had automated acceptance testing. That said, pair programming was only in place 23 per cent of the time: that's apparently still a weird meal for most to swallow despite the praises its practitioners sing.

To be a bit hand-wavy about it, the way we write and run applications is picking up much of the review board's work as well. The actual cloud platforms used to run applications are creating much

more resilient software that with things like the ability to roll-back problems and isolating poorly behaving services. Meanwhile architectural practices like microservices and12 factor app principles are describing how to design and write software that's designed for this resilience and speed of delivery.

## So what's an enterprise architect to do?

The role of the enterprise architect seems to be evolving as work is pushed down to the actual software teams and as staff on those teams become more "balanced" with all the roles needed on the team beyond just developers. There's a certain kind of architecture needed to sustain independently operating applications teams, and it looks like "architects" are well situated to be those enablers. This, of course, is in subtle but important contrast to being the change review "approvers".

This all reminds me of an old anecdote from the lean manufacturing world. At one US car factory that was going lean, trying to "catch up" with the Japanese, one of the senior presidents observed that the factory engineers were always very busy in their offices, doing some sort of work. "I do not think the problems are in that office," he told the factory general manager, "I think they are on the factory floor."

The implication for us in IT, of course, is that problems are not solved, nor software created in change review board meetings, but by the teams who are creating and struggling with the software every day. Findings from studies like the DevOps report are now showing this, and it's large companies that seem to suffer the most.

When that same study sliced up the findings by company size, it found that organizations with more than 10,000 people were 40 per cent less likely to be high performers than 500-people outfits. There

are many other factors causing that friction; if you're one of those large organizations, it's worth revisiting CABs.

*Originally published in* The Register, May, 2016.

# The developers vs enterprise architects showdown

One of the more wizened roles in ITis the enterprise architect, or, "EA" for those in a hurry. Now, those cowpokes over in the wide open office plans of DevOps country have little regard for these EA types. It's a bit of a "what have you done for me lately" situation: last we checked in, these EAs were saying no to cloud and before that they'd put in place something called "SOA" which turned into a clever, if unintentional, ruse to fly in the WS-Deathstar.

As I loaf around the DevOps circuit, the future of enterprise architects has become my top, unsolved mystery: what role do they have in this fully autonomous, heavily automated DevOps world?

## You shall know us by our trail of diagrams

EAs have a poor history of improving the lot of developers. Their focus has been on driving out duplication, mandating all too often baroque services and frameworks, and rounding up any rogue technologists who are trying new things, er, non-approved technologies. They certainly seem to show up to meetings, especially recurring ones with vague agenda like "review project status." (Whether EAs are "good" at meetings is left as an exercise to the reader.)

Usually they're armed with all sorts of diagrams, slides, and digital three ring binders. Just binders, and binders full of diagrams and six deep nested sections titles.

Those diagrams are like the primary totem of EAs: I once met with an airline EA who had an entire wall covered with a giant collage of boxes and lines describing how the entire company was wired together. "Now, tell me how I make a 'cloud strategy' out of that!" they demanded as we were sitting at their deluxe, intra-office mini round-table, the sign of real big wheel at an enterprise.

To be fair, these diagrams are intended to be helpful and, if you stared at them long enough, would actually be so. Someone has to keep up with what the overall big picture is, how it fits together, and as our mini round-table baring friend was suffering through, keeping everything up to date, all flexible and crouched down ready for the next industry curveball. "What are we gonna do about AR?!"

And while it takes a lot of skill to toil to aligning those boxes and arrows up correctly - have you ever noticed how "snap-to" grid points just have no aesthetic when it comes to arrow-ended lines? - EAs are infamous for not having touched a line of code since, well, that time way back when they did all this DevOps stuff on mini-computers but didn't call it "DevOps."

This malady presents in two extreme forms, First, the diffusion of innovation suffers: EA's who recommend fantastically new technologies at a mile a minute (they're probably saying "serverless" now, but hungering for some new word to chew up like a pack of sunflower seeds on a tee-ball pitcher's mound). Second, the laggards who in a voice that I can only hear in the sound Droopy say things like "hhhmmm, let's put it on the ESB."

## The DevOps work release program

Still, that idea of making sure everything fits together well and that IT is actually helping the business side achieve their goals

seems like something you'd want to keep. Take, for example, the scale of JP Morgan Chase with it's 19,000 odd developers. Even the most skeptical of us likely feel like there's some role in centralized governance at such scale. I've been talking with EA types at DevOps-minded organizations for most of the summer and there's a few recurring roles for reformed enterprise architects that keep coming up.

## Demilitarizing the EA police

One such EA at a financial company described the shift in their thinking as moving from "policing to partnering." Several years ago, an EA came in and put in place a traditional enterprise architecture, set of governance, and all the great diagrams. It didn't work out.

Here, we have the traditional "policing" mode of doing EA which is often more about enforcement than, if you'll pardon the use of vacuous terms for alliteration, enablement. After the policing debacle, that team now takes more of a "partnering" stance with the rest of IT. The goal is more to make doing the right thing easy rather than making the hard thing punishable by reprimand-by-meeting.

## Blinking cursors over spinning slide transitions

This also means scouting out and verifying new technologies to use, while also keeping an eye on standardizing on technologies like platforms and build pipelines. A standardized build pipeline provides, in fact, a hidden control point for governance. Just as failing tests won't let a build through, using unapproved runtimes and frameworks can halt a build. Similarly, with a good platform (the new, vague soupy word to use for "all that PaaS and container

orchestration stuff") in place you can control which languages, libraries, and even ports are open and network connections are made.

Most all of that governance about healthy and sound development and architectural practices can be baked into your infrastructure. You can see some intriguing work being done here in projects like InSpec from Chef and in the upper levels of the ever evolving container orchestration stacks. As any lazy parent knows, deflecting blame to some soulless enforcer like an egg timer is a much more effective way of getting children to comply with your wishes then just playing off your parental authority. So it goes with EAs and developers as well.

## Your microservices Gordian knot is adorable.

Planning out and managing microservices seems like another area where EAs have a strong role for both initial leadership and ongoing governance. Sure, you want to try your best to adopt this hype-y practice of modularizing all those little services your organization uses, but sooner or later you'll end up with a ball or services that might be duplicative to the point of being confusing.

It's all well and good for developer teams to have more freedoms on defining the the services they use and which one they choose to use, but you probably don't want, for example, to have five different ways to do single sign-on. Each individual team likely shouldn't be relied on to do this cross-portfolio hygiene work and would benefit from an EA like role instead minding the big ball of microservices strong.

# More of the same, just done differently

Though we'd like to think that the whiz-bang, new-fangled hotness of DevOps would erase the need for enterprise architects, as with agile, it seems to be more changing how EAs go about their jobs than getting rid of EA.

Some functions - like policing governance - can and should be automated, but still based on policy the EAs create and continually evolve. Also, who's going to pay attention to policing if all those DevOps teams are actually doing a good job?

The relationship between developers and EAs has always been terrible, so it's little wonder that individual contributor movements like DevOps are sick and tired of EAs. Nonetheless, especially in large organizations that don't have the liberty of dealing with just five or ten applications that help users graft party hats onto pictures of towering tempeh sandwiches, scaling up DevOps in enterprises likely needs much of what an EA does.

On the other side of the coin, the EA should actually know what they're doing, and know the latest technology and processes that could help their business and developers. The EAs mindset needs to change as well; those that create and run the actual applications have supremacy in a DevOps-minded organization. Enterprise architects need to treat these teams as customers, product managing their work appropriately. Maybe they could even work with those teams occasionally to see how the grub down in the trench is working out.

If DevOps people scoff at the idea of working with EAs, the feeling is usually mutual. EAs probably need to take the first step in mending the relationship. At worse, it'll keep EAs relevant. After all: "good job filling out our TOGAF architecture library!" said no CIO ever at the annual review.

*Originally published in* The Register, September, 2016.

# How many “modes” does this thing need?

There’s a debate going on right now about the best way to run IT: specifically, all those custom applications and services inside organizations. Do we try new, agile approaches, or stick to the old, methodical processes?

Gartner did much to start this discussion with their bi-modal concept:

Bimodal IT is the practice of managing two separate, coherent modes of IT delivery, one focused on stability and the other on agility. Mode 1 is traditional and sequential, emphasizing safety and accuracy. Mode 2 is exploratory and nonlinear, emphasizing agility and speed.

Mode 1 deals with predictable, well understood tasks, while mode 2 is for exploratory tasks, all those known unknowns and unknown unknowns out there. Gartner even works Cynefin in there for some complexity theory seasoning.

When you net it all out, much of how Gartner describes bimodal IT is pretty similar to the “slow down and think more about how to solve your problems, and focus on outcomes over processes” school of thought – aka the “stop doing dumb stuff” vibe that you hear from the DevOps world.

A key motivation for having two modes is to protect the new, agile teams from being clobbered by the old, waterfall-y teams and their Big Process ways: it can be too hard to change culture enough enough at scale to survive lift-off, and then you’re just stuck back in the muck.

## Sad mode[1]

Most people take all this to be sanctioning "old IT," resulting not only in freezing the use of new IT stacks ("cloud!"), but, also freezing any changes to the culture and process around those "mode 1" applications. The opposing camp, then, tends to see the result of bimodal as somewhat the opposite of Gartner's goals of encouraging and creating organizational oxygen for innovation.

As such, you can imagine that the "unimodal" (as we'll call them, eh?) camp asks the question "why wouldn't you just run everything in awesome mode?" It's a good point: the way the debate has been framed implies that some staff will be beset with operating in "sad mode" until the pink slips rain down. Rather than fixing how all of IT runs, if it's left to fester, Jez Humble says, your legacy IT will eventually eat you up from the inside:

[L]eaders that fail to move beyond Gartner's advice will end up falling further and further behind the competition. They will continue to invest ever more money to maintain systems that will become increasingly complex and fragile over time, while failing to gain the expected return on investment from adopting agile methods.

Gartner rival Forrester has noted several times that the happy/sad mode approach can lead to low morale and, thus, one would infer, less than ideal productivity. And as another dip-stick into the sump of sentiment around this issue, the reaction to bimodal as a keynote topic at the recent OpenStack Summit was along the lines of "I think I just threw up a little bit in my mouth".

In defence of the counter-counterpoint, as it were, there are some systems in which failure can be very expensive; thus, change carries more risk. Perhaps we should give those systems extra time and attention, or leave them alone entirely. We have this notion that

[1] I originally heard this sad mode/happy mode framing from Bridget Kromhout. I will assume she made it up. Triple Gold Star!

rapidly changing software, though it may delight users with a weekly cornucopia of new features, will cause downtime.

"Move fast and break things," as the West Coast motivational posters put it. "Yeah, not so much with my systems of record," the ITIL-set would retort.

Of course, the promise of the new, DevOps-y way is that there's ample testing and architectural resilience to remove such fears. If you can deploy at will, you can also patch and even rollback at will. Your operational maturity is a safety net. The DevOps set are interesting in proving out that unimodal is the one true way and, as indicated by Humble, believe the evolving research shows you can both move fast and avoid breaking things, if not make your software and staff morale downright better.

Thus far, much of the commentary has come from analysts and consultants. They of course imply that they're channelling the voice of the customer, but it's always good to go to source directly. In discussing how one of the US's largest insurers, Allstate, has been revamping their approach to IT department Matt Curry gave some advice on bimodal IT.

"It creates this dichotomy of competition and resistance," in the IT department Curry says, "and that's not really what we're trying to create." What they're trying to create is more collaboration and understanding amongst staff to propel a unified IT department forward. Instead of going together hand-and-hand, as Curry adds, "bimodal drives this wedge into your organization and it's a terrible, terrible thing."

(I'd be remiss if I didn't point out another, delightful, response to bimodal: the so called "trimodal" approach.)

## Can't we all, just get along?

I always get the sense that the two camps are, more or less, talking about the same thing: giving IT the processes and culture

needed to be more innovative and, thus, helping out the larger organization more. The two camps are just approaching them from different angles, constraints, and symptoms they're addressing. And, of course, with much of its content locked behind an expensive paywall, we can't expect many of the free-wheeling DevOps-set to be reading up on bimodal.

Perhaps Gartner would do well to jump, rather than toe-tip, into the fray. These two camps should sort out their differences and start talking with one voice. After all, we'd all benefit from the software at large enterprises and governments sucking less, and that's what both of these camps are after.

*Originally published in* The Register, June, 2016.

# Victory! The smell of skunkworks in your office in the morning

While it's easy to start up a few, flashy new DevOps teams, releasing to production each week and flaunting the ball-and-chain of enterprise governance, scaling that change to your organization will always be challenging, if not crushingly impossible.

When it comes to scaling the skunk-works, I'm reminded of a conversation with a struggling enterprise architect. I often use the company's mobile app and it's updated frequently, integrated well with iOS, and provides an overall very pleasant experience. Such results are normally unexpected from this type of aged, highly regulated, lumbering enterprise. As this enterprise architect was masterfully telling me why their organization was doomed, I piped in: "but the mobile app is pretty good - excellent even!"

"Oh. Well. I mean sure. But that's the mobile team," the EA said in an almost: "You kids today and your: 'I've-got-it-all-figured-out attitude - these olds'!" tone, "They're *different*."

My first thought was: "Er, well, maybe you should go figure out what they're doing right." More broadly, this situation pointed to the too-common anti-pattern of letting successful skunk-works teams live in isolation too long. There are two approaches I've seen for airing out the skunks and spreading change wider.

## The smiling knife roll

Changing an organization from within is extremely difficult. Most staff were hired to do a specific job and if they've achieved seniority, they've mastered their daily tasks and figured out how to max out their performance reviews. Few people are excited to change such a comfortable statis. And in larger organizations outsourcing contracts often act like concrete poured atop bleached coral.

In these circumstances, brute force and fear is often the fastest way to scale up the team of skunks. This starts with a tops-down mandate from executives who are deathly afraid of being "disrupted" by the dog-under-desk startups. Once vendors and consultants with their slides of startup logo-doom circle through, these executives get that "sense of urgency." Focused, annual freaking out is common, but what's key is that the executives actually round up budget and corporate attention to spend on this change.

More than likely, the next move will be to hire an outside maverick who carries a well-cared-for roll of knives who immediately engages in bureaucratic knife fighting. Internal champions can work as well and in some more staid, closed cultures may be all that's possible.

Either way, what's key is slicing away the organization mould and focusing on building up new staff and teams who are amenable to change. Otherwise, as so many tales of cowed mavericks show, nothing happens and you all too soon hear the soft clanking of those knives being rolled up and shown to the door.

As the new leader builds up their posse, they must also start the hard job of internally marketing how fantastic this new DevOps stuff is by highlighting the success of relevant projects. A well planned series of small projects can build momentum and build up the internal folk-lore of success. Coupled with that, internal conferences with staff from the new teams are often used to convince the coral encrusted that following new methods might

just be a good idea, and even make their lives better.

Hearing tales of success and, it's hoped, an easier work-life from peers is often the only way to convince staff. After all, management has always been going on about "change," and look where we are now - changing again!

There are uncountable micro-level tactics to discover and fix. Management must discover and iteratively try to fix these issues rather than relying on teams to heal-thy-self. Cajoling senior developers to pair program is a good example. Often, senior developers are threatened by the prospect of sitting at a desk with "junior" developers. Pairing seems to threaten their status as The Canny Guru who's grey-beard status is often relied on, for example, to solve Sphinx-level COBOL riddles.

But here's the incentive our knife-roll cuts with: in theory, by nature of being a senior developer, The Canny Guru actually enjoys writing code. At home, they've likely Python'ed up some sort of Raspberry Pi contraption to mist their iguana when the terrarium hygrometer comes up foul. Instead of being in meetings all day to dispense sage-insights, when the senior developer pairs, they find that they're back to writing code most of the day. They return to their old joy. And, if attending meetings and putting together well spaced out diagrams is their new love, well, it's clearly time for them to be, er, "promoted" to management.

## organizations are immutable

There's a handful of case studies from financial companies and government agencies following the principal I like to call "organizations are immutable." For those not up on their nerd talk, this means you can't change an organization once it is set up. The sunny side of this, is that you can still create new ones.

In this method of introducing a new approach to software, like DevOps, a brand new group is created that operates under the new

bureaucratic norms, slowing adding new projects to the new group. High-level management blesses this new organization to sweep its arm across a messy governance table of half drank ticket queues and droning CABs, starting with a tabula rasa.

The existing - now "legacy" - organization continues to operate as needed, but slowly, projects and people are moved to the new organization. Service dependencies from new to old are mediated and hidden behind APIs and facades, trying to decouple the two into a sort of reverse-quarantine: the old organization is blocked off from infecting the new group.

The new organization follows all the new-fangled notions, from the pedestrian practice of free lunches and massages, to paired programming, to fully automated build pipelines… all enabling the magic of small batches that result in better software.

The magic of this method is that it avoids having to unfreeze the glacier, namely, the people who don't want to work in a new way. Instead of doing the hard work of changing the old organization, management slowly moves over willing people, reassembling them into new teams and reporting structures. The old organization, though perhaps de-peopled, is left to follow its waterfall wont.

Publicly, companies have said that something around 30 per cent of people won't "make it" to the new organization. When I talk to executives in less than public forums - for some reason, always in poorly lit places like bars and parking garages - they say the number can end up being close to 70 per cent.

The "digital services" agencies created by various governments in recent years are some of the most documented examples here. Several large corporations have applied this pattern as well, often calling the new organizations "Labs." The entirety of the old glacier is far from melted, but the rate of liquefaction is having actual, real business effect. Who among us, after all, can resist ordering pizza through a watch?

## Success is the best deodorant

Either approach is still difficult and takes time. Based on what I've seen the first year will yield anything from 10 to, perhaps, 50 applications managed in the new fashion. It could be more if you're blessed with people and software that is easily massaged into your new process. In the second year you can expect to up that rate much more.

Changing how a large, 50-plus-year-old organization with thousands of applications is much harder than failing to success with all those hats-on-cats applications you see from the the ramen-fed, high-hemmed skinny jeans set. Once you build up a streak solid wins, though, introducing DevOps is easier to ripple through the organization… so long as management actually does their job of, well, managing.

*Originally published in* The Register, April, 2017.

# ROI Smoke Bombs and Diversions

At this point in the innovation curve for something like DevOps it's fashionable to start asking "Where's the Return On Investment?"

Answering that question is always tedious. For the hopeful, starry-eyed practitioner, spitting up the ROI figures is akin to the irrelevant water-carrying and wood-chopping trials imposed by a kung-fu master. Except instead of cold rice with snow-white topknots, it's dreary spreadsheets with pearly toothed finance flacks.

If you're lucky, your organization will be dead-set on taking on that "survival is not mandatory" mindset, ignoring questions like ROI. But, most everyone else has to fill cell range C45:G60 with all that water and wood.

## First, go drink

Your first inclination will be to crack jokes about flossing and Blockbuster. "What's the ROI on flossing, you ask? Well, do you like having teeth?" You'll follow up with erudite commentary on all the Blockbusters out there who were rearranging the deckchairs on the RMS ROI as it descended into the icy depths.

This is not helpful. Find yourself some colleagues, get a few pints, and have a laugh play-acting this out. Once you're back from second breakfast, try some more helpful approaches.

# What is this "ROI" you speak of?

When finance and management interrogators ask about "ROI" and "business cases," I find that they're mostly asking three questions:

1. Will this fit in the budget?
2. Are we paying too much?
3. Will this change actually work?

Sometimes they're asking all three questions, sometimes just the first two. Sometimes they're using you to practise their Cenobite impersonation with implements scrounged up from about the cube-farm. More likely, they're asking at least one of these three questions.

# Will this fit in the budget?

Of all the ROI questions, this is the easiest to answer. If you know the budget, you just need to figure out how you'll meet or come under it. When looking at DevOps, this means you'll first establish the baseline cost of following the "old" way, like staff's pay, tooling, and the expected cost of fixing screw-ups. Then model how DevOps concepts such as "two pizza teams" and "reducing release cycles" will lower your costs.

If your teams spend less time communicating with other teams, there's less time in meetings, clicking up presentations, and coordinating what to do after the meetings. Communication is more effective and efficient if you're all on one, small team.

You want your product teams spending 90 per cent plus of their time on product, but they're probably spending more like 20 to 30 per cent. Fewer, silo'ed teams will result in fewer errors caused by handoffs between teams. Meanwhile, DevOps' smaller batches of

code and weekly release cycles will increase the resilience of your applications (faster time to recover) and the productivity of your software (as you iteratively release, observe the use of, and improve your software's usability).

## Cost-cutting? It's possible...

If you want to pull out the trimmers, also look at staff reductions. Several large organizations I've spoken with have drastically reduced their operations and QA staff after modernising their software development and delivery approaches.

You can dress this up by saying those "resources" will be reallocated to "more high value activities," but if you're slotting in a huge amount of automation and pushing routine testing to the product team you may find yourself with a sizable thumb-twiddlers' budget.

When fitting into an assigned budget, your ROI answer is on the subject of "doing more with less."

## Are you paying too much?

We all like a good deal, and can agree that getting fleeced is a poor outcome. You'd like to know you're not overpaying. With a process change like DevOps, the tough question is "paying for what?" There are costs associated with modernising your software approach like buying new tools and hiring consultants (or "coaches") to help change your organization.

When it comes to tools - which usually means software, SaaSifed or otherwise - you're talking procurement negotiating and producing a proof of concept. There'll be alternatives for your development toolchain, for where you run your software (public cloud or on-premises), fees for middleware you use, and support and maintenance costs.

There are no easy answers, just models and competitor matrixes to work over. The raw tools here are standard technical tests to prove out the alternatives and the track records of other users, good and bad.

You might also ask if an outsourcer can do it more cheaply than your organization. Answering this question requires more of an assessment of the your organization's willingness to change, and not only the staff, but management as well. The change is not easy; executives I've spoken with estimate that anywhere from 30 to 70 per cent of people "won't make it".

## Will this actually work?

You've crafted up numbers for a business case, horse-traded your way to a good deal, and ensured that your people can pull it off. And seemingly, true to Larman's Law, people keep insisting on more justification.

Other than table-flipping your way into a new job, I've found three useful tactics here:

1. Other people's success, first hand - doubt about success tends to revolve more around "we already do that, we're just OG enough to not call it DevOps" and "we're not good enough." Occasionally (and always in comments from you, dear *El Reg* readers) there's the cry of "it's an Augean Stables' worth of offal." While there's no end of success stories when it comes to DevOps, rather than sending an email full of links that'll never be read, arrange actual meetings between your doubters and credible people from *other* organizations who've been successful with DevOps.
2. Hide - creating a "skunk works" is a tried and true method to bootstrap a new process, ignoring the haters in dry-clean creased dark denim. If you fail, there's massive risk. But if you

succeed, you've demonstrated that the new way is effective and to be trusted. Someone might even thank you.

3. Start small - do a series of small projects to prove out the new process. These can't be "science projects" and instead need to be something that's small, yet important to your organization. In doing these little projects, you're building up credibility for the new process and also learning how to do it.

## What's the ROI on ROI?

Finally, you should assess your own return on your time spent on cleaning out the stables. Will it be worth your time, personally and professionally, go to all these meetings and hustle up justifications for all the naysayers? Ideally, the answer is yes, of course, that's my job even! But carefully look at your situation, the political climate in the office, your chance of success, and the pay off you'll get. If you're on the wrong side of that Deming quote, it's best to enjoy the deck-top orchestra while you elbow your way into a lifeboat.

*Originally published in* The Register, September, 2016.

# Go DevOps before your bosses force you to. It'll be easier that way

Some people are making very bold claims about what DevOps can deliver. Here's one: "High-performing IT organizations deploy 30x more frequently with 200x shorter lead times; they have 60x fewer failures and recover 168x faster," according to the first bullet point of the 2015 annual Puppet Labs State of DevOps report.

With claims like those, managers in all sorts of organizations are starting to sit up and evaluate "doing the DevOps." And it's time to worry.

That's because, inevitably, those considering and then mandating a new DevOps direction will invariably have different interpretations of "what is DevOps". The desired goals and ways of getting there will be shaped by this understanding and follow from there. For better, or worse.

The problem is magnified because once a new idea has taken root and gained buy-in at executive level, challenging it is nigh on impossible – until the inevitable train wreck happens.

If you are in an organization where DevOps is in the air it therefore behoves you to make sure the management fully understands what DevOps entails, and especially understands that it's not a quick win.

## No quick win

Unlike virtualization, which became a quick way to save money with capacity management optimization, DevOps is not simply a technology that one puts into place to optimize an existing way of operating. DevOps – and the broader "cloud native" approach to custom written software development and delivery – is about changing how your organization functions, often along with which tools its uses, to drastically improve your software. You don't get eye-popping results like the Puppet Labs' reports by simply twiddling some knobs in the stack.

So, before you embark on your DevOps quest, it's good to make sure your organization understands what DevOps is and ensure that it is fully – nay, smartly – behind it. It's all too easy to launch DevOps programs based on misconceptions. Let's look at three of the more common ones that are easy gut-checks.

## DevOps is automation

The most common misconception is that "DevOps" means simply the use of Puppet, Chef, Ansible, Salt, or one of the other new automation frameworks. While much of the early history of DevOps is inextricably tied to these automation technologies, their use is, at most, merely necessary but is not sufficient for full DevOps.

Instead of just automation, DevOps also includes a very evolved agile style to product development. Software development is in the name. Without amindful approach to improving the actual software being developed, you're merely automating eventual failure – or, at best, mediocrity.

## The creation of a DevOps team

One of the more pernicious DevOps misconceptions is the need to create a separate DevOps team. The counter-point, here, is that "DevOps" is an end-to-end approach to improving your organization's software: from product management, to development, to QA, to deployment, to operations.

One of the chief theories of DevOps is that separating out the roles – and, thus, people – in that full process introduces more damage than "savings": each role ends up locally optimizing and losing site of the big picture. Hence, the constant barrage of "worked in dev, now ops' problem" slides in DevOps presentations. These separate silos also introduce "waste" in the form of the communication between teams as software moves through the various life-cycle "gates." Adding yet another team introduces even more waste.

The notion that a separate team, or person, handles all the DevOps related concerns belies a misunderstanding of what DevOps is at it's core: sweeping changes to how the organization operates, end-to-end.

## DevOps will save you money

Coming from a decade of near alchemical cost savings from virtualization, IT management is conditioned to think first about cost savings. When it comes to new technology adoption measuring savings is the easiest, most dramatic, and, thus, most addictive way to show ROI.

DevOps at first seems to have the trappings of a great cost savings initiative. The idea of having "one team" feels like you're reducing headcount. When crossed with the idea of a "full-stack developer" – those mythical beasts who can code and do systems management like some short order cook whose skills range from flapjacks

to foie gras – management can quickly start to salivate at the ideas of getting more from less staff. Confusing DevOps with just automation can add to the alluring mirage just over the horizon's edge of quick-savings.

While I would argue that cost savings do result from a mature DevOps-driven organization, they certainly won't come in the short term, nor will they be easy to model up-front. The "savings" are in things like quality of software and better uptime in production. These types of savings are all about "sucking less," which doesn't exactly model well in a spreadsheet.

## Be the baby, not the bathwater

We're very close, if not right at, the apex of DevOps's "inflated expectations."This year and next, I'd expect almost every organization to start asking the question: "How can we benefit from DevOps?" and putting "strategies" together to do so. The prognosticators at Gartner are predicting DevOps project failures at a rate of 90 per cent by 2018.

If you are part of staff who's responsible for implementing management decrees, now is the time to ensure your organization doesn't get saddled with some misconceived implementation of DevOps. You – and your organization – want to be part of the 10 per cent, not the 90 per cent. It's up to you to make sure that should any bathwater get thrown out, the baby of DevOps doesn't go with it.

*Originally published in* The Register, March 2016.

# You can't find tech staff – wah, wah, wah.

In a recent survey, the number of executives worried about a skills gap in IT grew from 49% in 2016 to 60% this year. Other surveys shore up this finding as well: a Cloud Foundry Foundation survey from late 2016 had 64% of their respondents worried about getting the skilled staff needed.

"Is there a skills shortage? No question about it," RedMonk's James Governor told me late last year, later adding "and we expect it to get worse."

While the systemic problems that cause a skills gap in "Silicon Valley" are finally being consternated over, in "the real world" outside of tech companies, the problem is likely even more dire.

Some (cough Oracle cough) are of the opinion that you shouldn't worry about skilled IT staff (particularly developers) and should instead focus on, you know, managing the procurement of more software. This opinion isn't too well reflected in my Friedman'ing in-and-out of stuffy conference rooms. Leadership tends to be more interested in improving their software capabilities rather than outsourcing them (because, you know, outsourcing has worked out so well in the past).

How might this "skills gap" be addressed?

## Bucolic programming

Location is one of the first self-imposed constraints on the supply of IT talent. All too often, companies like to hire in the big, hustle-bustle cities. There's an initial logic to this: just like a lumberjack

goes to where the wood is, a tech company will go to where a it thinks a pool of talented people are.

Fairly quickly, of course, this cements you into one locale which quickly becomes congested with competitors looking to out-benefit and poach your hard won staff. "Well, our free coconut water is organic! And we have *three*fridges full of craft beer!"

It turns out there's there's plenty of cities full of people who know how to computer. In the past month I've been to London, Riga, Kansas City, and Auckland. While the first is certainly, you know, a big deal of place, the last three wouldn't commonly be thought of as "hotspots" for tech talent. The locals said that, sure, hiring was hard, but not impossible. From what I could tell, each city was overflowing with as many foul tasting, locally crafted IPAs as any recruit could want to orally ruminate over while they figure out how to install kubernetes from source late into the night.

## Pay more to drive supply

A popular to encourage supply growth, of course, is to offer higher prices. In the US, "normal" developer pay seems to slide somewhere between $80,000 just over $100,000. That's already a pretty lux price, but it reflects the high value of the work done. And, if people are hard to find, perhaps the price isn't lux enough.

If talent is short, perhaps organizations should pay more. Scarce talent demands high pay. After all, companies seem happy to pay those elusive CEOs and VPs top dollar to attract the right talent, eh?

But, let's be a bit real: telling companies that they should be spending more probably isn't going to be well received.

## Training

Instead of find just *new* staff, you could also increase the productivity of your existing "supply" by better training your existing people. Management often believes they're doing enough training, while staff consistently believes the opposite.

Further up the staffing pipeline, coding "bootcamps" are promising. Early results are proving out the theory that IT skills can be taught in a vocational setting, instead of needing the highly vaulted, but high cost Computer Science degree.

For example, as she told me recently in a panel discussion, coming from the world of musicals, Chloe Condon drank from the fire-hose of a 12 week boot-camp and now finds herself nicely employed: "For people like myself, who, maybe have had a whole career before going into computer science there are definitely ways to ramp people up."

## Widen the hiring pool to drive supply

Supply is also low because we've been narrowing our filters. There are, after all, only so many male programmers in coffee stained Tiny Rick t-shirts, wobbling atop flip-flops, to go around. With some tweaking on demographics, you'll find there's of people who'd love stuffing themselves into a too-old-for-that t-shirt to type up bash scripts and kotlin for you. (If you're lucky, they might actually dress like an adult too - bonus!)

In addition to making sure your recruiting isn't limited by biases, it'd be nice if people actually wanted to work at your company because the culture was welcoming. As Governor put it, there are plenty of people you could be recruiting, but "you're not talking to them, in a way that is appealing. And you're not creating an organization that either encourages them to join or will sustain them in enjoying the company when they arrive."

As the past few years have shown, there's still an offensive culture at too many companies that can easily repel talented people whose skills were once considered so priceless.

## "There is no talent shortage"

"I get frustrated having coming out of the job search as a junior engineer only about a year ago, everyone's always saying 'where are these unicorn - this diverse talent out in the universe?'" Condon said of recruiting efforts that are too narrow, "A lot of places literally aren't letting them through the front door by requiring a CS degree" and the other trappings of a stereotypical developer.

With the right mix of training and widening our recruiting filters, there'll be plenty of people out there to fill everything from our dreary cube-farms shaded by stacks of TPS reports to the overly-lit open floor-plan offices smelling of leftover kombucha. The supply problem may never be solved perfectly, but it can certainly be made better. The talent is out there.

*Originally published in* The Register, October, 2017.

# Removing grumps from the DevOps punchbowl

*My editor at* The Register *suggested changing "grumps" to "Robin" so as not to offend* The Register *readers too much, which I did for publication. Never let it be said that* The Register *don't care about those kind, good meaning people who leave comments on that newspaper's articles. I changed it back.*

You know the grumps. Here you are, doing the DevOps so hard you've broken the spine of your *DevOps Handbook* and Robin won't get with the whole culture thing. They sit in the stand-up meeting, arms crossed, each morning mumbling "well, I wrote some code" and take that long, loud sip of tea. A grump will sabotage your improvement dreams. Something must be done.

## Maybe they're right: change is exhausting

Perhaps your grump has it right. This round of transformation might be the same squiggly pit of offal as the ones that came before. Throughout their career, they've been force marched through several searches for excellence and are now ready to ensconce themselves in a lovely, little cottage curating their model-train collection. Change is tiring, especially if every five you have to change again because the old system didn't work.

Sure, DevOps (and the broader meatware of agile and lean software) emphasize continuous learning, change, and adaptation as

part of the process. That might make you assume that past improvement initiatives were static - as is often the positioning of The New Methodology. To be sure, whatever the process du jour, an organization tends to calcify, cementing in tickets and change advisory boards like rebar to keep the stable and static. However, it's not like anyone who comes up with an IT methodology sets out to make a crappy one. ITIL doesn't kill good software, people do.

We are told by the high-performers of DevOps that empathy is an important tool. As much as it hurts with grumps, let's try. The grump likely had an incident, if not many, of transformation betrayal and have since learned the proper way to ride the stack-ranked wave without drowning. They have no reason to trust that it'll work this time and be worth the risks of trying something new.

Using a trick from the ancient tome of transformation, Leading Change, look at the alignment of your HR policy: "Performance appraisal. Compensation. Promotions. Succession planning. Are they aligned with the new vision?" If not, it's an indicator that grumps are justifiably crotchety. A recent Forrester study shows that performance appraisals mismatches are widespread. Improving your software capabilities is largely about improving customer satisfaction, but the study found that "[o]nly 20% of individual developers use customer satisfaction to measure success."

People are rational, if they sniff out that you're a facile change agent, singing the praises of a new way and then doing nothing to change the compensation system, they'll wrap themselves in a coat of many tickets.

## Volunteers only

Perhaps you can't turn the grump's frown upside down, or it's taking too long. While it doesn't work in the long term, many organizations using the volunteer model to handle the grumps. This

model sets up a new organization and only takes volunteers for the first handful of projects.

Jon Osborn describes this tactic at Great American Insurance Group: "we used the volunteer model because we wanted excited people who wanted to change, who wanted to be there, and who wanted to do it. I was lucky that we could get people from all over the IT organization, operations included, on the team…it was a fantastic success for us."

The theory here is that organizations are immutable: once you put a system in place, it can't really be changed. If that system can't be changed, you can't change the incentives and rules of the game, meaning you can't change how people behave. Setting up a new organization gets around that problem, potentially addressing the grump's paranoia. The volunteer model also allows the grumps to self-select and stay behind. There's plenty of ERP systems to manage, afterall.

## About that model-train collection

Finally, after going through this grump sleuth, you could end up with unchanging grumps, bottlenecks on two legs. In the free-wheelin' US of A, you can just fire these people - yay, capitalism!

Executives tell me that anywhere between 30% to 70% of your staff will have difficulty shifting to a new way of doing IT. One manager grimly told me - literally, in a poorly lit parking garage - that they should have ditched more. Of course, training and hiring replacements is a huge problem. Cutting heads should be the last resort least the grave you're digging for them turns out to be yours.

European organization have a problem here they've institutionalized caring about the welfare of people: it's very hard to get rid of people. In this case, yet again, quarantine people with the volunteer method. The retirement as change management strategy method is

slow moving, but perhaps you can rely on highly regulated, high barrier to entry markets to buy you time.

## The beatings will continue until...

While it's fine to, you know, actually care about the mental well being of people, making sure staff are happy increases your chance of success. "[R]esearch has shown that 'companies with highly engaged workers grew revenues two and a half times as much as those with low engagement levels," as the recent DevOps book Accelerate summarizes a 2012 study.

A grump will not only slow down their part of the software delivery life-cycle, but also start to drag down others, harshing that sweet bowl of OODA loops they're looking to chomp through. My sense is that a minority of these grumps are acting as completely rational actors. They've been conditioned by decades of corporate culture and policies that validate their change paranoia. While you may end up having to throw them out, it's best to first be empathetic and humane, even using them as a canary in the change coal-mine. There will be some that are fixed in mindset and can't be helped, but most of them will likely help you find and fix systemic problems, getting you one step closer to improving how your organization does software.

*Originally published in* The Register, May, 2018 *as "You're in charge of change, and now you need to talk about DevOps hater Robin."*

# How to avoid getting hoodwinked by a DevOps hustler

It being June, we're almost halfway through this year, and how's progress on those Digital Transformation Initiative slides doing? Maybe you need a quick jump in improvement to buy some time for August vacations, and then ensure you can get enough actual change and a few successful projects in place by the holidays.

While they'll ship themselves to your door, those gifts aren't going to buy themselves. At this point, finding an outside expert to push your jalopy-cruiser's KPI gauges up is a common tactic. And why not? It might actually work, and if it doesn't, you can always whip out your blame-storming finger to buy time while you look for a new job.

## 'What's this thing on my wrist, then?'

The larger the organization, the less likely management will have the skills to look at their own watch to tell the time accurately. That is, they'll need to hire some outside consultants to help shuffle the organization along to the golden path of DevOps. Traditional-minded watch-reading assistants have been biffing it of late, with well over 70 per cent of senior management registering dissatisfaction with traditional consultants in a recent survey. You're going to need some sluice-fiddling to find the good change consultants.

## Provenance

As any angler will tell you, going to where the fish actually are is a key step in catching one. Where should you find these experts, then?

"Here's a hint… strong DevOps consultants don't come from vendors," said Matt Walburn late of Target's DevOps team and now at Amazon. Now, now: if I may be biased a bit, perhaps a vendor will hire such people to staff a practice, but, sure, that heuristic will serve well more frequently than not.

Put another way, you'd like to find someone who's Done This Before. Without getting too far into a phrenologic alphabet soup, you want people who have experience both with the technology and the meatware.

Walburn characterizes a good pedigree thus: "A strong background in not only the technical and tooling side of DevOps, but also in driving enterprise-wide people and process changes." For example, a passing familiarity with the woes of DNS and Westrum-type spotting are good signs.

## Judge a person by how they've failed in the past

Many can claim to have such a broad skill-set, but they must be vetted somehow. Sniffing out a track record should primarily be driven by word-of-mouth and talking with past clients. When asked for a lengthy list of referrals, if a prospective consultant pleads that their past dance partners are unwilling to talk, be immediately suspicious. Any organization that's successfully changed is tripping over itself to tell others how awesome it went.

As a bonus qualifier, ensure that the referrals tell you plenty of stories of woe: no mercenary time-teller is without defeats and

warts. A referral is likely being overly glowy (read: dishonest or just indolent) if they don't tell you what went wrong.

And, ever looking for adding more heft to any blamestorming that must happen post bed-soiling down-the-line, building up a good list of credible people who said it was a good idea is a fine defence. Well documented due diligence is a solid parachute.

## Anyone can make pretty slides

It'll be tempting to qualify consultants by the beauty of their conference talks and, of course, written material (yes, dear readers, the ragged irony-crow sitting here my shoulder doesn't escape me). It's easy to fall into a Socratic trap of dismissing good rhetoric: just because one can speak pretty doesn't mean they're wrong. However, short-listing consultants based on their conference performances and spoken work takes some skill. As a general rule, you should differentiate between successful self-promotion and actual DevOps community appreciation, and, thus, accreditation.

"If someone is considered expert by people other than themselves, they typically show up in more places than their own self-promotional efforts," says Bridget Kromhout, head of the global devopsdays organization and suffer of reviewing many conference talk submissions. So, if all you see is a series of lambo videos, Twitter verified badges, and sponsored keynote talks at conferences suffixed with the word "Expo," be suspect.

Every good consultant knows that a book is a deft calling card. When you're using the bookshelf as a selector, ask yourself if the tome contains tactics you can actually put in practice, rather than just Sunday morning bromides. For example, Gary Gruver's books practically order you to put continuous integration in place, first thing. Other books, which shall go unnamed, spend much time painting the corporate inferno and telling you just about the end-state of bliss, but little on how to actually get from the path of sin

to the blessed elevator. You want stories of suffering, and how the expert fixed it.

## Final judgement

Once hired, you'll need to measure these consultants to make sure they're actually doing their job and worth their salt. This is devilish, as with all measurements involving meat-sacks that haven't been yet replaced by robots. How would you attribute causal success to the consultant rather than the organization… or apocalyptic failure when it was actually a bad plate of oysters your sysadmin had the night before that $440m bash-script error? If success has many fathers, failure kills all fathers involved.

The easiest way to track a consultant's progress is to ask the teams they work with if they've been helpful. If you don't mind long-term studies, you can track things like total number of teams that have run through the consultants fingers, crossed with those teams new-found abilities to release software more frequently with fewer bugs. But, will the consultant even be there long-enough for that?

Of course, you'd like to also account for the consultant's contribution to the success of the business that's driving all this change in IT. That can be dicey, however, if you're not honest with your assessment. Numerous consultants I've shared noon cocktails with as we Quincy, M.E. their client's train-wrecks have built compelling cases of organizations that eat themselves to failure despite all that good counsel.

To rate a consultant, you must clearly define the job you wanted them to do and the blast radius of success you allow them. This of course, requires knowing what your own goals and metrics are, and, dare I say it, what strategy your business is pursuing and how it'll know when it's failed and succeeded. And who among us, after all, has the time to figure that out?

Better to hire a consultant to come up with your strategy.

*Originally published in* The Register, June, 2017.

# Barriers to DevOps in government

There's just as much pull for DevOps in government as there is in the private sector. While most of our focus around adoption is on how businesses can and are using DevOps and continuous delivery, supported by cloud, to create better software, many government agencies are in the same position and would benefit greatly from figuring out how to apply DevOps in their organizations.

Just 13% of respondents in a recent MeriTalk/Accenture survey of 152 US Federal IT managers believed they could "develop and deploy new systems as fast as the mission requires." The impact of improving on that could be huge. For example, the US Federal government, by conservative estimates, spends $84 billion a year on IT. And yet, the Standish Group believes that 94% of government IT projects fail. These are huge numbers that, with even small improvements, can have massive impact. And that's before even considering the benefits of simply improving the quality of software used to provide government services.

As with any organization, the first filter for applicability is whether or not the government organization is using custom written software to accomplish it's goals. If all the organization is doing is managing desktops, mobile, and packaged software, it's likely that just SaaS and BYOD are the important areas to focus on. DevOps doesn't really apply, unless there's software being written and deployed in your organization or, as is more common in government agencies, *for* your organization as we'll get to when we discuss "contractors."

When it comes to adopting and being successful with DevOps, the

game isn't too different than in the business world: much of the change will have to do with changing your organization's process and "culture," as well as adopting new tools that automate much of what was previously manual. You'll still need to actually take advantage of the feedback loop that helps you improve the quality of your software, in respect to defect, performance in production, and design quality. There are a few things that tend to be more common in government organizations that bear some discussion: having to cut through red-tape, dealing with contractors, and a focus on budget.

## Living with red-tape

While "enterprise" IT management tasks can be onerous and full of change review boards and process, government organizations seem to have mastered the art of paperwork, three ring binders, and red tape in IT. As an example, in the US Federal government, any change needs to achieve "Authority To Operate" which includes updating the runbook covering numerous failure conditions, certifying security, and otherwise documenting every aspect of the change in, to the DevOps minded, infinitesimal detail. And why not? When was the last time your government "failed fast" and you said "gosh, I guess they're learning and innovating! I hope they fail again!"

No, indeed. Governments are given little leash for failure and when things go terribly wrong, you don't just get a tongue lashing from your boss, but you might get to go talk to Congress and not in the fun, field-trip "how a bill is made" kind of way. Being less cynical, in the military, intelligence, and law enforcement parts of government, if things go wrong more terrible things than denying you the ability to upload a picture of your pot roast to Instagram can happen. It's understandable—perhaps, "explainable"—that government IT would be wrapped up in red-tape.

However, when trying to get the benefits of continuous delivery, DevOps, and cloud (or "cloud native" as that tryptic of buzzwords is coming to be known), government organizations have been demonstrating that the comforting mantle of red-tape can be stripped. For example, in the GSA, the 18F group has reduced the time it takes to get a change through from 9–14 months to just two to three days.

They achieved this because now when they deploy applications on their cloud native platform (a Cloud Foundry instance that they run on Amazon Web Services) they are only changing the application, not the whole stack of software and hardware below the application layer. This means they don't need to recertify the middleware, runtimes and development frameworks, let alone the entire cloud platform, operating systems used, networking, hardware, and security configurations. Of course, the new lines of application code need to be checked, but because they're following the small batch principles of continuous delivery, those net-new lines are few.

The lesson here is that you'll need to get your change review process—the red-tape spinners—to trust the standard cloud platform you're deploying your applications on. There could be numerous ways to do this from using a widely used cloud platform like Cloud Foundry, building up trusted automation build processes, or creating your own platform and software release pipelines that are trusted by your red-tape mavens.

## Contractors & Lost Competency

If you want to get staff in a government IT department ranting at you all night long, ask them about contractors. They loathe them and despise them and will tell you that they're "killing" government IT. Their complaints is that contractors cannot structurally deal with an Agile mentality that refuses to lock-down a full list of features that will be delivered on a specific date. As you shift to not

even a "DevOps mindset," but an *Agile* mindset where the product team is discovering with each iteration what the product will be and how to best implement it, you need the ability to change scope throughout the project as you learn and adapt. There is no "fail fast" (read: learning) when the deliverables 12 months out are defined in a 300 page document that took 3–6 months to scope and define.

Once again, getting into this state is likely explainable: it's not so much that any actor is responsible, it's more that the management in government IT departments is now responsible to fix the problem. The problem is more than a square peg (waterfall mentalities from contractors) in a round-hole (government IT departments that want to be more Agile) issue. After several decades of outsourcing to contractors, there's also a skills and *cultural* gap in the IT departments. Just as custom written software is becoming strategically important to more organizations, many large IT departments find themselves with little experience and even less skill when it comes to software and product development. I hear these same complaints frequently from the private sector who've outsourced IT for many years, if not decades.

The Agile community has long discussed this problem and there are always interesting, novel efforts to get back to insourcing. A huge part is simply getting the terms of outsourcing agreements to be more compatible. The flip-side of this is simplifying the process to become a government contractor: it's sure not easy at the moment. Many of the newer, more Agile and DevOps minded contractors are smaller shops that will find the prospect of working with the government daunting and, well, less profitable than working with other organizations. Making it easier for more shops to sign up will introduce more competition rather than the more limited strangle-hold by paperwork, smaller market that exists now. The current pool of government contractors seems mostly dominated by larger shops that can navigate the government procurement process and seem to, for whatever reason, be the ones who are the most inflexible and waterfall-y.

Another part is refusing to ceed project management and scoping management to external parties; and, making sure you have the appropriate skills in-house to do so. Finally, the management layers in both public and private sector need to recognize this as a gap that needs to be filled and start recruiting more in-house talent. Otherwise, the highly integrated state of DevOps—let alone a product focus vs. a project focus—will be very hard to achieve.

## Addressing budgetary concerns with waste removal

Every organization faces budget problems, but tech startups seem immune to such fetters. We call them "unicorns" because they have this mythical quality of seemingly unlimited budget. The spiral horn-festooned are the exception that proves the rule that all organizations are expected to spend money wisely. Government, however, seems to operate in a permanent state of shrinking IT budgets. And even when government organizations experience the rare influx of cash, there's hyper-scrutiny on how it's spent. To me, the difference is that private sector companies can justify spending "a lot" of money if "a lot" of profit results, where-as government organizations don't find such calculations as easily. Effectively, government IT departments have to prove that they're spending only as much money as necessary and strategically plan to have their budget stripped down in each budgetary cycle.

Here, the Lean-think part of DevOps can actually be very helpful and, indeed, may become a core motivation for government to look to DevOps. My simplification of the goals of DevOps are to:

1. Ensure that the software has good availability (which it does by focusing on resilience vs. perfection, the ability to recover from failure quickly rather than avoiding all failure by rarely changing anything). This is something that recent failures in US Federal government IT can appreciate.

2. Enable the weekly, if not daily, deployment of new code into production with continuous delivery. The goal here is to improve the quality of the software, both bugs and "design" quality, ensuring that the software is what users actually want by iterating over features frequently.

Those two goals end up working harmoniously together (with smaller batches of code deployed more frequently, you reduce the risk of each causing major downtime, for example). For government organizations focused on "budget," the focus on removing as much "waste" from the system to speed up the delivery cycle starts to look very attractive for the cost-cutting minded. A well functioning DevOps shop will spend much time analyzing the entire, end-to-end cycle with value-stream mapping, stripping out all the "stupid" from the process. The intention of removing waste in DevOps think is more about speeding up the software release process and helping ensure better resilience in production, but a "side effect" can be removing costs from the system.

Often, in the private sector we say that resources (time, money, and organization attention) saved in this process can be reallocated to helping grow the business. This is certainly the case in government, where "the business" is, of course, understood not as seeking profits but delivering government services and fulfilling "mission" requirements. However, simply reducing costs by finding and removing unneeded "waste" may be an highly attractive outcome of adopting DevOps for governments.

## "Bureaucracy" doesn't have to be a bad word

As with any large organization, governments can be horrendous bureaucracies. Pulling out the DevOps empathy card, it's easy to understand why people in such government bureaucracies can start

to stagnate and calcify, themselves becoming grit in the gears of change if not outright monkey-wrenches.

In particular, there are two mindsets that need to change as government staff adopt DevOps:

1. Analysis paralysis—The almost default impulse to over analyze and specify with ponderous, multi-100 page documents the shifting to a more Agile and DevOps mindset. A large part of the magic of DevOps and Agile think is avoiding analysis paralysis and learning by doing rather than thinking in .docx. Government teams not familiar with smaller batch, experiment-based approaches to software development would do well to read up on *Lean Startup* think, perhaps checking out *Lean Enterprise* for a compendium of current best practices and, well, mindsets.
2. Stagnant minds—large organizations, particularly government ones, can breed a certain learned helplessness and even laziness in individuals. If things are slow moving, impossible to change, and managed in a tall blade of grass gets cut style, individuals will tune out rapidly. If DevOps is understood as a practice to help jump-start all too slow IT organizations, it'll often be the case that individuals in that organization are in this stagnated mindset. One of the key challenges becomes inspiring and then motivating staff to care enough to try something new and stick with it.

Again, these problems frequently happen in the private sector. But, they seem to be larger problems in government that bear closer attention. Thankfully, it seems like leaders in government know this: in a recent Gartner, global survey, 40% of government CIOs said they needed to focus more on developing and communicating their vision and do more coaching. In contrast, 60% said they needed to reduce the time spent in command-and-control mode. Leading, rather than just managing, the IT department, as ever, is key to the transformative use of IT.

## More than rats dragging pizza

In any given time, it's easy to be dismissive of government as wasteful and even incompetent. That's the case in the U.S. at least, if you can judge based on the many politicians who seem to center their political campaigns around the idea of government waste - and win! In contrast, we praise the private sector for their ability to wield IT to…better target ads to get us to buy sugar coated corn flakes.

Don't get me wrong, I'm part of the private sector and I like my role chasing profit. But we in the "enterprise" who are busy roaming the halls of capitalism don't often get the chance to positively effect, let alone simply help and improve the lives of, *everyone* on a daily basis. Government has that chance and when you speak with most people who are passionate about using IT better in government, they want to do it because they are morally motivated to help society.

The benefits of adopting DevOps have been clearly demonstrated in recent years, and for businesses we're seeing truth in the statement that you're either becoming a software organization or losing to someone who is. As government organizations start to think about improving how they do IT, they have the chance to help all of us, "winning" isn't zero-sum like it can be in the business world. To that end, as we in the industry find new, better ways to create and deliver software, it behoves us to figure out how government can benefit as well. That'll get us a even closer towards making software suck less something we'll all benefit from.

*Originally published at* FierceDevOps, September 2015.

# Addressing the DevOps compliance problem

Satisfying the mythical auditors is often one of the first barriers to spreading DevOps initiatives more widely inside an organization. While these process-driven barriers can be annoying and onerous, once you follow the DevOps tradition of empathetic inclusion—being all "one team"—they can not only stop slowing you down but actually help the overall quality of the product. Indeed, the very reason these audit checks were introduced in the first place was to ensure overall quality of the software and business. There's some excellent, exhaustive overviews out there of dealing with audits and the like in DevOps. Here, I wanted to go through a little mental reorientation for how to start thinking about and approaching the "compliance problem."

## Three-Ring Binder Ninjas

In this context, I think of "auditors" as falling into the category of governance, risk and compliance (GRC)—any function that acts as a check on code as and how the code is produced and run as it goes through its lifecycle. I would put security in here as well, though that tends to be such a broad, important topic that it often warrants its own category (and the security people seem to like maintaining their occultic silo-tude, anyhow).

The GRC function(s) may impose self-created policies (like code and architectural review), third party and government imposed regulations (like industry standard compliance and laws such as HIPAA), and verification that risky behavior is being avoided (if

you write the code, you can't be the same person who then uses that code for cash payouts, perhaps, to yourself, for example). In all cases, "compliance" is there to ensure overall quality of the product and the process that created it. That "quality" may be the prevention of malicious and undesired behavior; that is, in a compliance-driven software development mindset, the ends rarely justify the means.

In many cases, the GRC function is more interested in proof that there is a process in place than actually auditing each execution of that process. This is a curious thing at first. Any developer knows that the proof is in the code, not the documentation. And, indeed, for some types of GRC the amount of automation that a DevOps mindset puts into place could likely improve the quality of GRC, ironically.

## Establishing trust and automating compliance

Indeed, automation is one of the first areas to look at when reducing DevOps/GRC friction. First, treat complying with policies as you would any other feature. Describe it, prioritize it and track it. Once you have gotten your hands around it, you can start figure out how to best implement that "feature." Ideally, you can code and automate your way out of having to do too much manual work.

There's work being done in the US Federal government along these lines that's helpful because it's visible and at scale. First, as covered in a recent talk by Diego Lapiduz, part of what auditors are looking for is to trust the software and infrastructure stack that apps are running on. This is especially true from a security standpoint. The current way that software is spec'd out and developed in most organizations follows a certain "do whatever," or even YOLO principal. App teams are allowed to specify which operating systems, orchestration layers and middleware components they want. This

may be within an approved list of options, but more often than not it results in unique software stacks per application.

As outlined by Diego, this variation in the stack meant that government auditors had to review just about everything, taking up to months to approve even the simplest line of code. To solve this problem, 18F standardized on one stack—Cloud Foundry—to run applications on, not allowing for variance at the infrastructure layer. They then worked with the auditors to build trust in the platform. Then, when there was just the metaphorical or literal "one line of code" to deploy, auditors could focus on much less, certainly not the entire stack. This brought approval time down to just days. A huge speed up.

When it comes to all the paperwork, also look to ways to automate the generation of the needed listings of certifications and compliance artifacts. This shouldn't be a process that's done in opaque documents, nor manually, if at all possible. Just as we'd now recoil in horror at manually deploying software into production, we should try to achieve "compliance as code" that's as auto-generated (but accurate!) as possible. To that end, the work being done in the OpenControl project is showing an interesting and likely helpful approach.

The lessons for DevOps teams here is clear: standardize your stack as much as possible and work with auditors to build their trust in that platform. Also, look into how you can automate the generation of compliance documents beyond the usual .docx and .pptx suspects. This will help your GRC process move at DevOps speed. And it will also allow your auditors to still act as a third party governing your code. They'll probably even do a better job if they have these new, smaller batches of changes to review.

## Refactoring the compliance process

To address the compliance issue fully, you'll need to start working with the actual compliance stakeholders directly to change the process. There's a subtle point right there: work with the people responsible for setting compliance, not those responsible for enforcing it, like IT. All too often, people in IT will take the strictest view of compliance rules, which results in saying "no" to virtually anything new—coupled with Larman's Law, you'll soon find that, mysteriously, nothing new ever happens and you're back to the pre-DevOps speed of deployment, software quality levels and timelines. You can't blame IT staff for being unimaginative here—they're not experts in compliance and it'd be risky for them to imagine "workarounds." So, when you're looking to change your compliance process, make sure you're including the actual auditors and policy setters in your conversations. If they're not "in the room," you're likely wasting your time.

As an example, one of the common compliance problems is around "developers deploying to production." In many cases and industries, a separation of duties is required between coding and deploying. When deploying code to production was a more manual, complicated process, this could be extremely onerous. But once deployments are push-button automated with a good continuous delivery pipeline,you might consider having the product manager or someone who hasn't written code be the deployer. This ensures that you can "deploy at will," but keeps the actual coders' fingers off the button.

As another intriguing compliance strategy, suggested by Home Depot's Tony McCulley (who also suggested the above approach to the separation of duties) is to give GRC staff access to your continuous delivery process and deployment environment. This means instead of having to answer questions and check for controls for them, you can allow GRC staff to just do it on their own. Effectively, you're letting GRC staff peer into and even help out

with controls in your software. I'd argue that this only works if you have a well-structured platform supporting your CD pipeline with good UIs that non-technical staff can access.

It might be a bit of a stretch, but inviting your GRC people into your DevOps world, especially early on, may be your best bet at preventing compliance slowdowns. And, if there's any core lesson of DevOps, it's that the real problems are not in the software or hardware, but the meatware. Figuring out how to work better with the people involved will go a long way towards addressing the compliance problem.

*Originally published in* FierceDevOps, December, 2015.

# Great, we're going to get DevOps-ed. So, 15 years of planning processes – for the bin?

In large organizations, the question is rarely "what are these newfangled practices and technologies," but more "how could we actually do them here?"

DevOps[2] has been here nigh on 10 years, and in the past three or so of those, large, normal organizations, like Allstate and Duke, have been learning its mysteries.

"I think that for the IT staff, once they try it, they will never do it another way," Allstate agile transformation manager Matt Curry said when I asked him about applying DevOps. That's something you hear over and over again when it comes to putting DevOps in place.

While putting improvements and changes in often seems like something that can't happen at your organization, the results are too enticing to ignore, and the business side of the house is expecting big things for IT, like yesterday. "Based on our business feedback," Duke Energy's director of digital strategy and delivery John Mitchell told me, "it's 10x better."

[2]"Yes, but what is DevOps?!" you maybe screaming or typing. Let us just assume, for now, that it means: "Improving the quality of your software by speeding up release cycles with cloud automation and practices, with the added benefit of software that actually stays up in production."

# Less analysis paralysis, more continuous planning

A focus on improving software with DevOps techniques requires an organizational mind-shift. In the traditional mindset, even in the past 20 years of supposedly doing agile, software was seen as a lengthy project, executed to fulfill a tome of requirements, targeted at a specific launch date. Slow and careful release trains and planning also limited the number of releases each year, putting a damper on the feedback loops which improve software in a small batch approach.

Most organizations, then, have taken a project-oriented approach to software. This means that IT staff and contractors are forced into a huge, up-front analysis and commitments used to manage them to schedule.

Former CIO of US Citizenship and Immigration Services (now at AWS), Mark Schwartz says: "To demonstrate that [IT staff and contractors were] performing that type of work responsibly and for the business to verify that it was doing so, the scope of each task had to be defined precisely, bounded, and agreed upon in advance. The work had to be organized into projects, which are units of work with a defined set of deliverables, a beginning, and an end."

Now, as any overly clever agile-cum-DevOps fan-girl will quickly point out: "Yes, but where's the part that ensures the software is actually useful?" Of course, such a thing is the goal of all those controls described by Schwartz.

A more contemporary view of software, though, is angling to discover exactly what the software should be by systematically understanding users, discovering what works (and doesn't!), building the software, observing how people use it, and then starting the process over again. This reorientation changes the organizations' planning process: "it's only when we started shifting the focus on

'outcomes'," John Mitchell told me, "[that] we start to see that there is a new approach we can take in front of the planning process."

In general, people have only the foggiest notion of what their software should actually do until they start trying. Thinking that you can deeply understand the problem you're solving, Allstate's divisional chief information officer claims, vice president technology and strategic ventures Opal Perry told me in an interview last May, "[is] a traditional pitfall where we thought we knew with absolute certainty where we were going and it turned out we thought we were going south. But we need to go north."

This means there's not only much less time spent on up-front planning, but much less time along the way spent verifying that developers have been following the plan. Instead of verifying the status of projects, you verify that actual business value is delivered in the form of software that's useful.

## Project Management

With all the talk of "products, not projects," you'd expect all those PMP-types in the Project Management Office to freak out. Which, to a certain extent, is always a good idea for those who enjoy paychecks. However, as many noted, PMO capability is still needed, especially for more complex applications.

Recently, after giving a long, DevOps soliloquy at a large enterprise, an astute project manager beset with modernising a rats' nest of mission critical, but aged services soliloquised back at me. They made odd poetry out of a long list of cross-service dependencies, regulations, COTS-uses, data concerns, and integrations. "Yeah. Sounds like you need some project management," I recall saying in my snarky character: "Good luck - next question!"

Less glib, Matt Curry outlined a heuristic for getting enterprise-grade project management involved. "PMO is super helpful when

my batch sizes are large and my feedback loops are long," he said. "When batches become significantly smaller and the feedback loops are shorter the need for that [PMO] is lessened. The second place that project management is useful is when you have a lot of external coordination."

## Finance

Handling financing in a DevOps-oriented organization takes some care. Previously, because IT purchased their own kit for development, QA, and staging labs would require a capital expense (capex) approach. The amount of servers needed, of course, was a drop in the bucket compared to the amount of hardware needed for production, which were even larger capital expenses. With a DevOps approach, which typically depends on using public cloud, these expenses switch to operating expenses (opex).

The application teams, of course, love operating in an opex model because it speeds up finance planning and lab building times: they can get to the value of actually creating and releasing software quicker. However, if the accounts don't pay close attention, they're gonna have a bad time.

Namely, while the opex of the pre-production environments may seem smaller than upfront, capex, once the application moves to production, the opex might blossom like algae in a stagnant creek. This is especially true if the application is cursed with success, chewing up opex capacity at an unpredicted rate. If you can effectively manage 10,000 machines in production, Israel Gat, a renowned independent software & IT consultant, points out, financially you might be better off running in your own data centre. The exact cut rate for that number will always be debated - with server vendors tossing endless FUD into the debate - but it's worth finance keeping a close eye on where compute should be done and how it'll effect planning.

# Tickets no more

With the promise of de-crudding the process of acquiring IT assets and release management, it's little wonder that traditional IT service management changes as well. Perhaps it's little wonder that ticket-driven IT is decreased. Duke Energy's John Mitchell notes: "It's so nice to not have to ask, plan and wait for infrastructure. Also, with our cloud engineering team co-located with the software engineers, they solve problems in real-time instead of [waiting on] tickets. It's so cool watching one of our hipster mobile devs walking and talking like best buds with a big burly ops engineer."

This measurable, in your face metric is also a good way to motivate those BOFHs. "It wasn't easy to win them over at first," Brian Silles said, " But once they saw 35–40 backup tickets per week go down to mostly zero, they got on board."

# But think of the poor CABs!

Then, there are the basics of trying to put 15 pounds of tickets in a 5 point bag: "if I'm doing 8 or 15 releases a week," HCSC's Mark Ardito asked, "how am I going to get through all those CABs?" The Change Advisory Boards - who hardly ever "advise" so much as stick you in a box of pain until you confess to your enterprise architecture policy subversion - needs to speed up whatever benefit they're bringing. Most organizations I talk to are baking much of their policy enforcement into their automation, build pipelines, and platforms. It's also clear that the usual 9 to 5 of enterprise architects needs to change (exactly how is still fuzzy).

Something like Chef's InSpec is finding early success here to enforce policy in the pipeline and monitor drift in production, while the cloud native platforms and add-ons like the various Cloud Foundry distros, Red Hat OpenShift, and Istio all have components that seek to make robots out of those CABs.

## Starting

Finally, after all that ironic up-front planning and contemplation, there's the method for choosing and sequencing your first applications to rub DevOps all over. The resounding advice from those who've done it - or realised they should have - is to start small. "We started small," John Mitchell said when reminiscing about starting up, "Then [when] we started getting noticed, more business pouring in."

The likes of Home Depot have spoken extensively about the process of starting with small projects, then building up to larger projects. These initial projects aren't "science projects," but have actual business value (like running the paint and tool-rental desks in Home Depot's case). Success means creating actual business value (read: less suck, more cash). On the other hand, as you learn how to do the DevOps, mistakes along the way have less negative impact than, say, bringing down the .com site.

Sometimes, though, you have to go big or go home, as the wide-toothed, neck-vein popping set like to say. "Ultimately, it is a matter of the cash flow situation of the company," says Israel Gat, "Starting small is less risky, but operational/financial parameters might force you to adopt an 'all In!' strategy."

Once you select software to work on, the process of good design-think kicks in. But instead doing - you guessed it! - up-front analysis and specification, designers stay involved during the whole process. This means expectation and organizational changes for your design people and departments: they're now in the soup every day, not just contemplating chamfering in their tidy work-spaces.

## The only easy day was yesterday

Once the engine starts, it has to be maintained, which is typically a change in mentality and motions for "leadership." The organization

needs to continually crank-down on wastes like time waiting in ticket and review board queues, relentlessly squeezing out efficiencies where possible. The most vital, helpful part of DevOps is something it stole, outright, from Lean manufacturing: continuous improvement. DevOps itself has been undergoing changes as technologies automate some of the more manual steps and these large organizations bring more learning to the practice, perhaps even "killing off"DevOps as it evolves to whatever's next.

At the leadership layer this emphasis on continuous learning implies creating and maintaining an organization that's always eager to get better and, even, change dramatically. The MBA-wonks call this "a sense of urgency," and as documented long ago, if the organization doesn't have that urge to change, little will happen. What I've seen in recent years is that, sadly, unless there's an external threat to the organization - cough, cough, Amazon, cough - not much will change, despite whatever decrees an executive or an eager young DevOps expert will spew into the organization. There's relief though if this sound exhausting. As my more macabre thoughtlords and ladies are fond of (mis-)quoting: "It is not necessary to change. Survival is not mandatory."

*Originally published in* The Register, March 6th, 2018.

# The many-faced god of operational excellence, DevOps and now 'site reliability engineering'

Someone's been kicking up the "NoOps" ant pile again. There it was, sitting there finally rebuilt after the annual upturning, and The Lord of Cartography, Simon Wardley says: "I think you'll find that the new legacy is going to be DevOps." That said, it is winter, so the ants are moving a bit slower than usual.

## But we've only just started...!

This "LessOps" vibe matches my own anecdotes as flit about the IT departments at large organizations. At the same time, so many IT departments are hungry for DevOps, they want to understand it and put it into practice, to be sure. Surveys are showing growing interest: the annual DevOps report says the number of respondents working on DevOps teams rose from 16 per cent in 2014 to 27 per cent in 2017.

A 2016 Forrester survey reports that 69 per cent of respondents reported adopting "processes that embrace or resemble DevOps". And while I can't help but think that those 69 per cent are those of you, dear readers, who leave comments for me professing that you've been doing DevOps since Churchill's second term – and "just didn't call it that" – let's take Forrester's survey here as useful.

# What was that middle part again?

First, what exactly is DevOps? As John Willis, one of the co-authors of the DevOps Handbook told me: "Unfortunately, DevOps means whatever the definer wants you to believe and no definition is wrong." He went on to give his definition by way of describing the end state: "DevOps is about service as a supply chain and all the things that enable fast, resident and consumable delivery of the service."

Another dean of DevOps, Gene Kim, described it much the same way, as quoted in Gary Gruver's excellent book on scaling DevOps: "DevOps should be defined by the outcomes. It is those sets of cultural norms and technology practices that enable the fast flow of planned work from, among other things, development through tests into operations, while preserving world-class reliability, operation, and security.

"DevOps is not about what you do, but what your outcomes are. So many things that we associate with DevOps, such as communication and culture, fit underneath this very broad umbrella of beliefs and practices."

As ever, successful technology-driven definitions quickly become a description of the outcomes rather than how you get there. But wait! Our DevOps report friends took a bold swing at defining exactly what DevOps is, first by practices, then by effects, and then by outcomes. Their reports are full of excellent, science the shit of it charts that catalog DevOps practices.[3]

To me, the key to figuring out where DevOps begins and ends – what it is as a practice – is asking what's done after a functional agile development team does a build. How does the organization deploy the build to production, then ensure it runs in production, and then ensure that it can be upgraded on-demand?

[3] At one point, I had a chart copied from the reports here. I didn't want to take the time to get permission to use it, however. I mean, really - do they want to show up here?

Early on, the answer was to automate, automate, automate. Instead of manually deploying builds to production, you'd use Puppet or Chef, for example. Then you'd use containers, and then came the idea of "cloud platforms" that dictated exactly how you'd package, deploy, and manage your software and gave you close to zero options about the stack below your application. Each of these was built around the idea of "the wall" between developers and operators, and removing the negative effects of that wall.

Developers would make their build, then throw it over to operations staff who'd have to figure out how to deploy the build to production and then manage it ongoing. This wall introduced so much variability in configuration management that, inevitably, someone would forget to configure the DNS servers and the whole system would go down each time a build went to production. I'm greatly oversimplifying here, but solving that problem of frequent deployment drove a tremendous amount of DevOps thought and innovation.

If you re-read Willis's delightfully concise definition, this notion fits in pretty well. Back-solving from the goal of more frequently deploying software (that, as a bonus, also stayed up), DevOps discovered a host of "culture" practices and issues that it's become much more famous for.

And at some point, the venerable practices "agile" were added whole-hog into the mix. And why wouldn't it be mixed into the batter? The end goal is creating better customer and user experiences, which means not only ensuring that the software runs in production, but that it's well designed.

Todd Underwood, a site reliability engineer at Google, summarized the process and cultural consequences well a few years back: "DevOps seeks to integrate operational concerns into the software and business practices and software/skills capabilities into operations." This can include operations staff actually embedding with the developers, especially those developer teams who have very little

operations skills.

## Who automates the automaters?

Early on, the notion of DevOps was that a unified team of developers and operators (I mean, it's right there in the name, right?) would figure all of this out, working hand in hand and all carrying pagers to create, deploy, and then manage their applications. A huge amount of work in DevOps centered around automating the end-to-end process of getting software into production, so it's little wonder that "configuration management" is often seen as "DevOps". But, as the tools and practices started to coagulate, these teams did more than just automating configuration management, they'd build "platforms" out of the standard stacks processes they'd been following.

Getting your teams to build platforms, being "full stack developers" as we used to call them, seems excellent, at first, until you're lucky enough to operate at enterprise scale. For example, say the 19,000+ developers at JP Morgan Chase. At that scale, you get a real 1 + 1 = -3 effect because you're duplicating all those stacks. Using production monitoring and management as a tracer for this anti-pattern of too many stack developers, 451 Research's Nancy Gohring told me: "[This] leads to the situation where some enterprises have 50 monitoring tools, sometimes including multiple deployments of the same tool. That seems inefficient."

Ideally, to "scale", you want to not only automate the toil of IT management, but standardize and centralize it. You don't want developer teams building their own stacks and managing their production applications in unique ways. You want to automate the automation. As Google's Kelsey Hightower put itrecently talking about serverless and DevOps: "Once we get the practice right, it should turn into technology."

## How dare you say my bash scripts aren't proper programming!

The idea of Site Reliability Engineering, or "SRE", fits better in this view of what DevOps is. SRE-think is not focused on pulling developers into a unified team with sysadmins. Instead, the goal is to get sysadmins to start thinking like programmers, actually writing code and developing systems for production use. Sysadmins are no longer responsible for just running what developers give them.

Sure, they spend time troubleshooting production problems like a classic sysadmin would, but once the hair-on-fires are extinguished, the immediate question is: "OK, how can we change our platform to automate all this ops toil?" The idea, as Underwood put it, is to: "Write infrastructure that doesn't require that kind of procedural automation."

Technologically, the re-emergence of platforms as a service (PaaS) and the growing dominance of Kubernetes for management are automating much of the manual, one-off processes of DevOps. (You should know, dear reader, that I pay my mortgage by working at one of the vendors that peddles such kit.)

The end result is enabling developers to focus on their applications, not actually carrying a pager or worrying about whatever a "DNS" is. As Allstate's Matt Curry described it to me: "The goal is to eliminate cognitive overhead for the developers and keep their pipelines as simple as possible. They should get a ton of operational value for free just by pushing to an environment, be it monitoring, release process, security scanning, architecture patterns, or anything else that is repetitive and fairly consistent between deployments." Similarly, good SRE staff take a code-first approach to solving problems and make operating production as simple as possible.

## Who's SRE'ing my résumé updates?

Does this mean we can start using all those DevOps books as kindling? Well, hardly. Doing software well has always passed through many names, but the general end goals have remained the same. We can all agree with a sentiment put well by Microsoft principal cloud developer advocate Bridget Kromhout: if DevOps means "good, tool-enhanced cross-team collaboration. I sure hope that never goes away."

DevOps is "a cross-team practice, not a task," she adds. "SRE is the new Ops Engineer, but DevOps shouldn't have been considered to be a job in the first place. We don't hire a collaboration expert to do all the collaborating."

The organization treating development and operations as all part of the same concern – creating and running good software – shouldn't be lost. The DevOps practices of being more humane in working with people seem, well, humane and pragmatic. Most importantly, the emphasis on continuous improvement and the injection of lean thinking have helped lead to huge improvements in organization I talk with. As someone who's always complaining about how boring the "cultural" aspects of DevOps are, they'll probably be the most long-lasting, important precepts.

There may be a few stick-in-the-muds who get all roiled at the idea of DevOps shifting around, even "dying" if you're the kind of person who likes that hyperbole (why are you looking at me?). Clearly, "DevOps" is evolving as it spreads into mainstream organizations and as newer technologies automate what were once manual practices and disparate tools. That doesn't mean the core goals and "culture" zoom away. Just as with "serverless", you're supposed to take the idea seriously, not literally.

So, if you buy all that, consider this some 2018 career advice: it's time to go update your résumé and say you've been doing "SRE" this whole time, but just not calling it that.

*Originally published in* The Register, February 6th, 2018.

# A print button? Mmkay. Let's explore WHY you need me to add that

With all our attention on the Morlocks – down there with the whirring gears of Kubernetes and steaming clouds – it's easy to lose track of the Eloi: those concierges of productivity that smooth out the rough, upper crust of The Stack to make sure your software is actually usable.

The trend now is to call this layer "design", which is just fine. "UX" always seemed a bit like an X Games sport, and even the Morlocks know that "UI" is just a part of the story. Exposed bricks and pale-wood desks only go so far in seasoning good software. There's an actual methodology there as deep and sculpted as any DevOps-cum-SRE discipline.

Let me give you, my Morlock friends, a framing that – though likely shallow and far from complete – will give you a good tour guide to the surface of The Stack.

## The parable of the print button

Your first encounter with "design" tends to follow a predictable flow: "UIs, amirite – how hard can it be? I just need you to add a print button to this page. Should take, like, five minutes, right? 20 tops?"

Before you can scoot off to digging into the latest, real-life BOFH accomplishment, the designer coughs, Jeeves-like. They straighten

up, and step off their standing-desk mat, running their fingers through their forest green-tipped hair to smooth it out. Armed with a fresh stack of sticky notes, your designer says: "Well, first, let's explore why you need a print button."

Fear not, ye five-minute-feature fiends, this is the process working. This is the first step on the journey of software that doesn't suck (rather, in most cases, software that sucks less).

What your frosted-tip friend is driving at is the first stage of design. As Erika Hall recently riddle-listicled: "Only after you have a goal will you know what you need to know. And you have to know your question before you can choose how to answer it." The question is: why do they need a print button? Is a print button the best way to accomplish that task? Do they even need a print out?

Answering these questions gets to the bottom of what you're actually trying to do. A sort of annoying application of the five whys (are there, really, any enjoyable applications of the five whys?).

This is a good path to go down, but it can feel just like a sysadmin longbeard who, when asked how to write an awk script to break-up a CSV file, will say: "Why would you want to do that?" instead of just answering the damn question.

## The sticky notes are made of people!

The next stage, "research", is where you start to discover the answers to these questions. Of course, this means first knowing exactly who these "theys" are. Your design friend might bring in a friend – all thick-rimmed bespectacled, sporting a fresh Mackle-more up top – to draw up some "personas". These are profiles of your us-… ah, pardon me, that's a term that's too easy to backslide to… people.

Sagely stroking his Maestro's buttered beard, he'll sort out what

the motivations, wants, and needs of these people are. Each will be given a name, and perhaps a delightful sketch or properly bokeh'd photo of people like "Pat". Does this person even have a printer? Perhaps "print" is the wrong action needed here for poor printing Pat.

As with all great small-batch questions, we should gather some actual user studies to validate our theories, or invalidate them to then come up with new theories.

## A million one-way mirrors

I don't presume to know the art our well-oiled and styled design friends practice at this stage – gazing into the daily work of your software's factotums. But there is a craft – even art – to discovering the mysteries of the print button.

In recent years, all our lower-level IT has actually contributed a great deal to design research. In previous decades, the best a designer could hope for when observing users (sorry) people was using baroque, one-way mirror rooms with dozens of cameras perched to spy on people as they moused through the UI. Now, thanks to highly network-dependent apps, designers can watch every single thing every user ever does. While that might be creepy in the hands of hucksters and black-bag millennials, it's pure gold for designers.

Way back in 2009, taking advantage of this methodology a famous Microsoft study (PDF) found that only a third of the features in their software were used as intended, or used at all. Think about all that fat in the other two-thirds that could be trimmed! We expect software to constantly evolve, better fitting what's actually useful and productive for the people using it. These millions of one-way mirrors give us a sharp knife.

## Small batches rule everything around me

With all this in hand, to cut a long story short, the designers iterate, small-batch style, until they systematically figure out the why and how of that print button. Perhaps it turns out that people don't actually want to print out that form, they just want to archive it somewhere: why not email instead?

This feedback loop, done weekly or even daily, is what improves modern software and is the source of much of the business value in "digital transformations".

And then we come back to you, dear Morlocks. All that work putting a release pipeline in place, a solid cloud platform, and ensuring production resilience is a huge part of what's empowering designers to do much of this.

Improving and optimising your stack wasn't just an exercise in lean efficiency and Taylorian lead time reduction. The actual goal was to improve your organization's software. DevOps friends, this is where you can crawl out of your warrens and take a victory lap in the sun for once. Try not to eat too many designers while you're up there.

*Originally published in* The Register*, February 20th, 2018.*

# The best outsourcers fire themselves

Outsourcing. Let's talk about it. The agile and DevOps people can't stomach the idea and will tell you that, intuitively, outsourcing something as core as software development ruins any chance of enterprise success. But whither comes this bone-deep skepticism among the cloud cognoscenti? Surely there's value to be had. Surely.

## You get what you pay for

The story frequently plays out the same. I've certainly heard it innumerable times in those affordably furnished Fortune 500 conference rooms I spend my time in. A frisky new CIO comes onboard, charged with ship-shaping a flabby IT department. Their first move, of course, is to sizzle off all the fat, namely "non-strategic" IT services. You know, those things like infrastructure management and operations, maybe even writing software. A deal is signed. It's always a large deal. The CEO lauds the CIO's optimization efforts, shareholders crown the CIO with the almighty halo. Our CIO opportunistically scrambles up the career cliff from, say, a Fortune 200 company to a Fortune 100 company (I mean, who among us doesn't like more money?). They're off to slice through the new place with that sharpened halo.

Meanwhile, back in the IT department, necessarily ornate service agreements and contracts are put into place between the outsourcer and the organization. The outsourcer is wonderful, sure, but we need to make sure it holds up their end of the bargain. To make

sure it is meeting SLAs, maybe we should get someone to audit and project-manage the outsources. What's that? One person isn't enough? Well, maybe we should create a new team of people who audit and manage them. We'd better update the runbooks too and setup a meeting to see about updating the SLAs.

"That outsourcing transformation sure did save us a lot of money," I've been told, years later, by IT staff. "Now that we're trying to be all DevOps," they pause for dramatic effect, perhaps leaning back and sighing, "well, just sending in a ticket to ask the outsources for a development database takes two to four weeks. If you fill the ticket out wrong, golly, that's when you really feel the 'cost savings' slice deep in your shins."

Sometimes, at this point, our story resolves. Commonly, a friend told me as I was sailing slices of rare steak through L'Entrecôte's green sauce, outsourced projects boomerang back to the in-house staff. An enlightened (and likely new CIO who was brought in to, yet again, fix the mess in IT) pulls the project back in-house, giving it over to some capable developers. Seeing the quality of the code, or, the simple lack thereof, staff will often have to start over, chucking the outsorcerer's code down dark chutes of the refactoring abattoir. That's not the ideal scenario, but it's better than another narrative that too often plays out.

## My own people can't be trusted

It's not like we IT nerds have done a great job building confidence in our efforts over the decades. After years of trying to get IT to, in their view, actually do something useful, executives often throw up their hands and start booking meetings with outsourcers. You can imagine this table-flipping executive saying: "I'm not gonna get what I want anyway, so I might as well get it for a third of the price!"

Bringing in outsourcers is proof that the organization already mis-

trusted its in-house IT. Inevitably, the usual causes of failure - ever changing requirements, overly ambitious deadlines and budgets, etc - will plague the outsourcers' efforts. No one is immune from the difficulty of creating good software. This struggle increases the layers of outsourcer oversight. Quickly, this method of project management becomes the norm for the entire organization. As was once said: "Trust, but verify." Now, the organization holds both in-house staff and outsources at length, papering over its learned mistrust with endless three-ring binders, enterprise architect reviews, and Project Management Office callisthenics.

CIO Mark Schwartz calls this insidious cycle the "contractor-control paradigm," and points out the quick slide into busy-work that follows. With so many oversight processes to manage, a new problem emerges: managing each project becomes a unit of work itself, another project that must be managed! Often, this meta-project-management layer becomes the highest priority.

After all, if the weekly status report is all red, or even formatted incorrectly, the Big Boss will get suspicious and start casting about to change the paradigm yet again. The entire system is built on management-by-mistrust, so any small slip can only be proof that the trembling project manager in front of you is, indeed, papering over bad news. It becomes vitally important, then, to get through the Monday status meeting, so button up your deck! Never mind, you know, if the software actually works well or not.

This state of mistrust and upward happy-talk couldn't be further from the Agile state organizations wanted in the first place. Each layer of the organization mistrusts the layer below it and fears telling the truth to the layer above it. What a fine mess we've gotten ourselves into, and all just to save a few billion dollars!

## You can't spend EBITA in the grave

Looking at industry surveys, this poor state of affairs checks out. "[N]ot even a third of [VPs and middle-managers] view their [outsourcing] engagements as being very effective at driving out significant cost or making their operations more flexible and scalable," as Phil Fersht put in his analysis of his firm's annual outsourcing survey, "Their bosses are slightly less cynical, but still the vast majority is underwhelmed."

Ironically in this history, executives now need their organizations to master software development. In the same analysis, Fersht suggests that there's a ray of light for outsourcers if they, themselves, can only transform. Executives still believe in the dream of outsourcing, that they can pay for a "true partner" in innovation.

Successful organizations I've spoken to want outsources to be training wheels for their transformation. Finding in-house talent is a constant bugbear, though likely fixable with a small bit of thought. Businesses want something beyond "augmentation," however, from outsources. In the state they're in after decades of letting in-house talent atrophy, these firms want an outsourcer to help get the engine going, get staff trained up by working on real projects… and then leave.

My rule of thumb, then, with outsourcing software development, is to ask the outsourcer what their plan to fire themselves is. How long will they need to be around, exactly? If they don't have one, then they're likely planning on sticking around for a long time. And next thing you know, you'll be catching a lot of boomerangs which, lacking the right equipment and skills, often turns out poorly for people who depend so much on their fingers.

*Originally published in* The Register, March 29th, 2018.

# Chapter 3 - Vendors-Sports

When this happened, I can't say, but early on I became enamored with the tech industry. Following the software each company released, their internal machinations, their understanding of the market, the FUD they deployed, and, well, everything about them. It made me a good industry analyst, if I do say so myself.

I lucked out early in my analyst career because open source was on the ascension and the important of developers was apparent. Equally important, and mostly forgotten, blogging was born early in my career as well. In fact, it's through my blog that James Governor and Stephen O'Grady discovered me and pulled me into this new career path I'm on. For that, I'm ever thankful.

Since then, my hobby has been watching the tech industry like others follow sports of celebrities. At parties, I'm silent and hidden in a corner. I only come alive when people start talking about software and, better, what software companies are doing. What's up with IBM? Have you noticed Microsoft's turn-around?

As with any study, the longer I manager to stay alive the more enjoyment I get out of this hobby. You see the same patterns, you learn more history, and you have a growing body of stories you can use to fortify your mind-palace. Here's a few of the better lounge chairs in there.

# Will the blighters pay this time? Betting big on developers

Not too long ago, selling middleware and tools to software developers was a big business.

Large technology empires were built on a single premise: that computers need software, written by software developers who need a panoply of infrastructure tools and middleware - and that you could charge the developers for that infrastructure. Household names like Borland, BEA, and Rational chugged along merrily.

While vendors counted their middleware lucre, developers debated "Java vs .Net" like it was some kind of important existential concern. Disruption came from marrying up the rainbow-and-sandals world of FLOSS to the free-loading spirit of the web… and the developer market quickly had its Napster moment. Free (and - sure, sure - open source for those who care) technologies like Perl, PHP, Python, and Ruby coupled with free, "enterprise grade" web services like Apache (and then nginx) all but eviscerated the "developer market" in the 2000s.

The finishing slice came when the Java ecosystem open-sourced, through JBoss, and then Sun capitulated to open source, and finally the perfection of SpringSource arrived. The list goes on - MySQL, anyone? Soon, the idea that you'd make money by *selling* to developers was laughable. Instead you made money by selling the company to another company: MySQL sold to Sun for $1bn, and SpringSource sold to VMware for around $400m.

# You've got to make friends

While the developer market withered, developers remained an important force, "king-makers" even, as my former colleagues at RedMonk put it. Winning developer allegiance is still key to building ecosystems around companies and platforms from historical cases like Microsoft and Java, to Apple's success in recent years. But making money from developers directly was another story.

A developer might shell out $50 to $100 for a nice text editor or IDE: but pay for middleware? Nope. Running that middleware as a cloud-based service, however, is a clever hack around this parsimony. Developers are driving much of the spend on IaaS and PaaS.

Developers need somewhere to run their software, and buying new servers doesn't appear to be where they're putting many of the new applications. There's certainly a market there: taking out SaaS, [our bottoms-up forecast at 451] puts the public cloud market at around $20bn in 2016, with the majority in IaaS and about 25 per cent in PaaS. When you look at Amazon's portfolio, it's essentially a stack of middleware, all run, and charged "as-a-Service."

A quick look at the customer logo-porn pages of competing IaaS and PaaS providers shows the importance of developers; many of these companies, like Rackspace, have established developer relations programs and before getting snatched up by IBM, SoftLayer was self-admittedly focused almost exclusively on developers. With all the hand-wringing about "shadow IT" - namely developers going out-of-band to pay for cloud-resources - the connection between developers and spending is even more direct. It seems developers are perfectly happy to pay for middleware, if only you'll run it for them.

## Follow the money… the VCs' money

Incubators like HeavyBit are betting on the return of the developer market as well. Companies in their portfolio service developer needs, from IDEs like Codeenvy, to QA platforms like Rainforest, to the API documentation service Apiary. Of course tracking VCs' bets is more about doing once-removed crystal-ball-gazing on what the mega-vendors would acquire in three to five years than accurately tracking broader trends.

And to that point, IBM's movement back towards the developer market at its recent Pulse conference provides one of the better indicators for which direction the developer market is going. Over the past year, IBM has been reorienting large swaths of the software group's portfolio around not just cloud, but cloud as a medium to go after developers.

Armonk's finest tripped over each other to paint the vision of enterprises getting back into software development, driven by the need to interact with customers in new and exciting ways ("social") and on new and exciting platforms ("mobile," as well as tablets).

Big Blue released a beta of its new middleware stack, BlueMix, based on Cloud Foundry. Filling out the new middleware stack more, IBM picked up Cloudant, a Database-as-a-Service company, one of those previously free chunks of downloadable middleware that's now Monetizable-as-a-Service now. Indeed, at 451, we estimate the DBaaS market was worth $150m in 2012, growing at a CAGR of 86 per cent a year to $1.79bn in 2016, attractive to any vendor strategist.

With all of the cuts and bottom-line optimisation going on at IBM, this much focus on an entirely new middleware stack, targeted at developers, is a large signal of intention.

IBM's not the only one placing developer bets. VMware and EMC bet on developers in 2013 when it combined the SpringSource,

Cloud Foundry, and Big Data assets together into Pivotal, with ex-VMware, now Pivotal CEO Paul Maritz saying they were building "a new platform for a new era." This was quickly followed by GE $105m investment into the developer-oriented company.

These "new platforms" are certainly out there, from the Jawbone Up that tells me I don't walk enough and sleep poorly, to new marketing applications that better track and seek to program my buying habits. Where there's software, there are developers.

Will the developers finally pay for the tools they use to make their write and run software? In the consumer space of $19bn exits, oddly enough, perhaps not: many of the old ways hold true – there is still DIY pride and 20-year-olds with nothing better to do than code all night.

Outside of the Ramen-noodle-coated technology world, however, as more devices get IP addresses and need software accordingly, it's *not* full-on bonkers to think that there will be more developers at "normal" companies. And that's the meat-and-potatoes of any "infrastructure" play: the mainstream companies which would rather purchase tools and middleware than quickly polish off another cup of Ramen before firing up a bare-bones editor to type up yet another chunk of middleware from scratch.

*Originally published in* The Register, March 2014.

# Uncork a bottle of vintage open-source FUD

"Yeah, but is open source a safe choice?" Surprisingly, I've been asked that frequently of late. Larger organizations in particular are giving me the old squinty eye. The folks in these conference rooms and tentacular email threads are often looking to replace decades old stacks of IT and get their "digital transformation" on, so perhaps they can be forgiven asking such a dated question.

Having been out on the choppy proprietary seas so long, these organizations are a bit wobbly with those sea legs and can't rightly say where the land begins and the water stops.

## Who are these people?

Most of the open source questioners come from larger organizations. Banks very rarely pop up here, and governments have long been hip to using open source. Both have ancient, proprietary systems in place here and there that are finally crumbling to dust and need replacing fast. Their concerns are more oft around risk management and picking the right projects.

It's usually organizations whose business is dealing with actual three dimensional objects that ask about open source. Manufacturing, industrials, oil and gas, mining, and others who have typically looked at IT as, at best, a helper for their business rather than a core product enabler.

These industries are witnessing the lighting fast injection of software into their products - that whole "Internet of Things" jag we

keep hearing about. Companies here are being forced to look at both using open source in their products and shipping open source as part of their business.

The technical and pricing requirements for IoT scale software is a perfect fit for open source, especially that pricing bit. On the other end - peddling open source themselves - companies that are looking to build and sell software-driven "platforms" are finding that partners and developers are not so keen to join closed source ecosystems.

These two pulls create some weird clunking in the heads of management at these companies who aren't used to working with a sandals and rainbow frame of mind. They have a scepticism born of their inexperience with open source. Let's address some of their trepidation.

## If all your friends jumped off a bridge...

There was a time when using open source seemed a little odd. IBM's billion dollar Linux R&D spend in 2001 was quite the eyebrow-raiser, but they could smell the revenue wafting in from the future. By 2008 a significant amount, if not a majority, of buyers were comfortable with open source.

That positive sentiment has only grown. This year's, 2016, long-running Blackduck open source survey found that 78 per cent of the respondents were using open source software to run their businesses (well mixed in with closed source, of course), up from 42 per cent in 2010. Similarly, Forrester's recent surveys shows that just 13 per cent of developers have yet to use open source. Even in the seemingly staid world of manufacturing only 10 per cent of respondents have never touched open source.

Put another way, the usage of open source to support, if not outright run a company's core businesses is normal. Like, *totally* normal.

You'll be using open source sooner rather than later. How can you make sure it goes well?

## Keeping an even keel

How to answer this hidden generation of doubters? Make like it's 2006 all over again: marshal your arguments, plan a strategy, prove it works.

First, you should only aim to use open source projects and software that have a stable, long life. Part of what you pay for with closed source software is the comforting sense that the company will be there for years to come. At least, you *should* be getting that guarantee. You don't want to build your business on-top of a fly-by-night technology - open or closed! - that leaves you stranded on an EoL island.

To sniff out stability, I'd evaluate the project's community in three areas:

- Is the community relatively free of conflict? If developers and stakeholders are generally congenial, the long term prospects are better than if they're antagonistic.
- Is the community thriving? You want the code to be continually updated, with the freshest features and whizbang thought technologies (REST! Responsive UI! Microservices!). Otherwise you'll be stuck with "legacy" frameworks, slowing you down like barnacles, and equally hard to remove.
- Does the community generally stay on the same course, or is it always changing tack? Multiple identity crises and dramatic changes can add a tremendous amount of instability to the project's roadmap, requiring you to change yourself or stay left behind on old, unsupported versions.

# Put on those sandals and slide up the rainbow

Beyond just using open source, many companies are now in a position where shipping open source looks to be a vital, strategic option. As previously "analog" devices like turbines, planes, buildings, cars – and, yes, even coffee machines – are essentially highly networked computers. Businesses can benefit from gussying up their their new "platforms" with open source. While companies like Microsoft, Oracle and Apple show that you can build platform communities around closed source platforms, using open source as a tactic is valuable, perhaps even easier.

Attracting developers (and their patron companies) to your new platform is difficult, no matter open or closed. By my reckoning, building the community of business partners and developers is easiest when open source is involved. Partners are suspicious of the commercial motivation of the platform owners and look to open sourcing to provide a type of mutual assured success and continual access to the core platform. Developers simply like having freely available code and continue to put more faith and interest into open source projects than closed source ones.

All of this conjecture amounts to a strong suggestion for companies considering anything having to do with sticking software, APIs, and network connections into their devices: think seriously about the benefits (and drawbacks!) of open source your core platforms. The tried–and–true "open core" model provides plenty of room for high-priced, closed software wrapped around a delicious open core. Keeping your platforms 100 per cent closed has the potential to, well, close off too many opportunities.

While all of this may seem blindingly obvious to the old salts out there, as more industries inject software into their devices and businesses, this topic will keep floating back up. Us tech-obsessed people will scratch our heads and wonder what's wrong with them.

But it's all new to those being eaten by software. Hopefully they'll figure out that it's smooth sailing with open source.

*Originally published in* The Register, September, 2016.

# Pizza, roaches, and Java

A while back I answered my doorbell - it was the pizza. After transacting for the hot pie, the older delivery man with a Just Like Dear Old Dad mustache asked: "Are you a programmer?" pointing to the OpenStack logo on my hoodie sleeve. "Yes," I said, "well, I used to be." He asked me what programming language he should learn and quickly added "JavaScript?"

Taking the pizza, I said: "Java. There's lots of jobs in Java."

"You mean Java*Script*?"

"Well, I mean, everyone knows JavaScript. But there's always work in *Java*."

Java is the great shoggoth of the programming world. Seemingly eternal, ever shifting and growing, but above all massive enough to mindlessly roll over any competitors, even the withering of time. Over the years, Java has been declared dead many times, but despite numerous visits to the grave it has constantly remained one of the top three languages in use.

The nature of Java changes constantly, and despite an ever-fragmented and quarrelling community and coming and going vendors, what seems like disorder is a wonderful advantage of constant adaptability and, thus, stability over time. The most recent fear and uncertainty comes from a shift in the nature and usage of Java's enterprise-y face.

## The money in Java

The business – nay, "enterprise" – version of Java has always provoked the most vitriolic responses from the overall development

community. "Why do we need all this bulk?" the Ruby and Python kids have always harrumphed in their tight pants. "Why don't we just code this mortgage application approval workflow in Go?" the shiny object squirrels on Hacker News comment.

But like a baffled, aloof grey-hair in dad jeans, squinting over his glasses at that hand-sized Android phablet, Java Enterprise Edition (Java EE) chugs along. This variant of Java aims to add in the common frameworks used by organizations to go all "enterprise grade" in their applications: transactions, RBAC, database integration, common methods for web development, and numerous other components all corralled together into a standard stack.

Most of the these components can be used on their own - most famously the servlet spec which defines a widely used method for writing web applications - but having the all-in-one kit is supposed to afford you the usual 1 + 1 = 3 benefits. The additional angle vendors of Java EE stacks trade on is actual production performance and manageability: that silo'ed bundle of uptime needs for the pre-DevOps set.

As you'd expect from anything going by the nom de guerre "enterprise," Java EE products and services have driven a huge amount of revenue for decades in the Java world. In 2015, Gartner estimated that the application platform market was $7.8bn. But share of Java EE revenue in that mix has been shrinking in favour of non-Java EE options. The lions of the Java EE market, IBM and Oracle, have been dropping in revenue share while non-traditional Java EE vendors have been rising quickly, with growth in the double digits in 2015.

Looking back, Java EE has been an incredibly reliable bucket of technologies, running numerous, mission critical applications across companies of all sizes. Plenty of colorful personalities have come and gone and repeatedly kicked over rose-scented piles of Java muck to the delight of train-wreck coroners like myself. For example, the ever bombastic and florid JBoss crew and their head iconoclast, Marc Fluery, never failed to delight in calling out the

slow Java standard process and mercantile interests of Java vendors.

In exactly the opposite direction of sharp-elbow pizzaz, the calm and plodding Spring community led by Rod Johnson slowly changed the usability and nature of Java development as a sort of friendly rebel next door.

Both of these communities sought to strip down the "bloat" of Java, making it smaller, and easier to use. Like so many Java startups, each of these firms was gobbled up by the mainstream: JBoss was acquired by Red Hat in 2006, while Spring found its way first into VMware in 2009 and then into Pivotal (my employer).

## Dead again

Last year, a report from Gartner declaring the end of Java EE's dominance invigorated all the latent tension in the community. As one of the report's authors put it: "People don't need 90 per cent of the stuff sitting in Java EE to build modern enterprise applications."

Predictably, such statements drove counter-arguments of analyst payola and ignorance in the ever-delightful ad hominem style (I'm looking forward to finding out how much of a moron I am this time in the comments section - KISSES!). Vendors who benefited from a decline in Java EE only offered wry smiles and download figures, while the original analysts gave the equivalent of a tired shruggie as they stuck by their guns and pdf-splained their original analysis.

What gets lost in this near annual Java flagellation is that Java's ever-green viability and usefulness is driven by its ever evolving nature. What worked in one decade isn't the best approach in the next. Early on, Java and Java EE were the equivalent of a fully operational battle-platform, loaded with features that could dominate any transactional workload to space-dust.

It also provided a widely understood standard for enterprises development: you could easily find programmers who knew the

stuff and didn't need training. That aircraft carrier is overkill for today's buzzword-blizzard of architectures (microservices! Cloud-native!), and now, a stripped down Java that centers around agility, speed, and (still!) reliability is the endless hunger Java is feeding.

You see this in usage of Java EE as reported in the long-running Zeroturnaround Java surveys: in 2014, usage of Java EE was at 68 per cent and this year has declined to 58 per cent.

Meanwhile, interest in Java remains strong: IDC estimated that there were five to seven million Java developers in 2016. While Java EE as a lifestyle may be on the wane, Java as a whole is holding strong. There's been some recent threats from Go and Swift, but reports from firms like RedMonk show Java's amazing ability to go along with changes in the industry and stay on top.

Oracle's occasional, goofy product management decisions since acquiring Sun Microsystems (the original steward of Java) has made the community nervous over the years. There have been delays (I mean, it's software after all - delays are a feature, not a bug) that have slowed down the evolution of the official Java standard, but open source members of the community have stepped in over the years to keep the train moving.

Oracle's continued pursuit of lawsuits and occasional licence true-ups have a tendency to freak the community out as well, but more provide blog-screed fodder for the loyal opposition. And, you know, maybe it's sometimes worth paying for something that your entire business relies on - just sayin'.

However it gets packaged up and sold, Java consistently ends-ups satisfying the application development and runtime needs of most organizations, from desktop, to cloud, to pocket-computer. "We just like roaches, never die, always live." Indeed, in recent years the official Java EE standards have added in slimmed down profiles as well, matching the desires of the ever evolving Java community. The memo was received, and read.

Despite all the inside-bickering, lawsuits, a shotgun wedding to

Oracle, drawn-out releases, and rivals from PHP, to Rails, to Swift, Java is still in wide use and shows no signs of finally dying. Jobs-wise, you'd be hard-pressed to find a better language than Java as your primary programming language if you wanted to switch from dropping off hot-pies to writing code.

As they say in this "gig economy" world: you're either delivering pizzas according the code-dictates of a programmer, or writing the code to tell pizza delivery drivers which door to knock on.

*Originally published in* The Register, March, 2017.

# O HAIOps! Can AI deliver BSM dreams, or just more BS?

"So, let me ask you," he said, smoothing out his goatee with one hand. "Five planes have been circling for hours, delayed. You can land one," a long pause, "how do you choose?"

Us 20 something developers had been corralled into a windowless room to meet with this new "CTO," but we were pretty sure they didn't do much right clicking in Rational to extract those handsome sequence diagrams. "The one with the least gas!" one of my colleagues said. "The one that has the highest revenue!" I interjected. And the joker of the group: "Pick randomly!"

"No, no, and no. You pick the one with the most elite status travelers," he said. This was unfair. None of us traveled as much as he did. I don't think we could put together the proper attire for a wedding or a funeral amongst the six of us. "These are your best customers, so you want them to be happy. And *that* is what Business Service Management is!"

Confounded, we went back to what any self-respecting 20 something developers at the dawn of the 21st century were doing: writing a bespoke ORM framework.

## Maybe this time we'll get it right

This notion of measuring the "business impact" of all those whirring disks, CPU gauges, and web applications has always been with us.

It's the height of any high-falutin' systems management ethos. It doesn't matter if the CPU is pegged, what matters is if customers can book a car, buy a book, or see real-time, targeted ads for the removal of unwanted belly fat and the reduction in their monthly mortgage payments with the use of one, natural, miracle pill – all the while shopping for MAGA baseball caps.

Of course, just as vendors get close to solving the problem, a new array of middleware and infrastructure comes along (mobile, cloud, voice, blockchain, or whatever new technology Charlie Munger will hate on next year in Omaha) changing even the most basics of the problem space. And, so here we are, some 15 years later, and systems management vendors are still trying to reduce the time IT fritters away on the pea-soup of metrics and tickets that hurtle towards them each day. "Business outcomes," friend.

But wait! There's a new way to focus on the business value: AIOps. First, it meant "Algorithmic IT Operations," but clearly, that dog don't hunt, so it now means "Artificial Intelligence for IT Operations" - the second "I" is silent.

The phrase pretty much defines itself, but here's how Gartner's Andrew Lerner puts it:

AIOps platforms utilize big data, modern machine learning and other advanced analytics technologies to directly and indirectly enhance IT operations (monitoring, automation and service desk) functions with proactive, personal and dynamic insight. AIOps platforms enable the concurrent use of multiple data sources, data collection methods, analytical (real-time and deep) technologies, and presentation technologies.

Yup, yup: the kind of one dish dump dinner our MoMs are always after.

Not to be left out, Forrester calls the whole notion "cognitive operations":

Software that applies advanced analytics and machine learning to

analyze historical IT operations data and make predictions that expedite management, speed problem resolution, prevent developing problems, and attach business significance to problems resolved or prevented.

CogOps! Barkeep, when you see this PDF reader empty, just fill it right back up!

What's shared between all definitions is applying Machine Learning to whatever large corpus you have lounging about with the aim of solving and predicting ops problems. That seems fair: if Google Photos can cluster together all my pictures of late 20th century clown paintings, the ops wizards out there should be able to automagically find patterns and even start to predict when things will go wrong in the glass cage.

## In action...?

Gartner tells us that by 2019, 25 per cent of enterprises will use AIOps to support "two or more major IT operations functions." Achieving growth rates that'd make Munger's head pop, in its "Market Guide for AIOps Platforms" report, the same put the number at a whopping 40 per cent in 2022, saying it was at about five per cent penetration in 2017. Hopefully between those two estimates there was some revision of the prediction models: that's some insane growth.

However, that five per cent explains why it's so hard to find enterprises who're gushing about their success with AIOps.

"Using AI/ML in IT ops could be as simple as automatic thresholding," 451 Research's Nancy Gohring told me. "On the more useful end of the spectrum, it means presenting users with some actionable information." Triggering those actions comes from machine learning all that historic data, as she explained: "the alert could be based on automatic thresholding or analysis of historical incidents, potentially across a vendor's customer base." That could then be

presented "along with possible suggestions for a way to solve the problem - again, based on analysis of historical activity."

For example, one insurer who applied Moogsoft's AIOps approach used the machines to reduce their event storms by 99 percent, while HCL worked with the same steer and carved down their ticket queue by 62 per cent - so precise!

## There's no cold winter in heaven

When you ask most people in the industry about the term AIOps, they roll their eyes. While it *sounds* cool, they collectively admit that the name overpromises. As Gohring told me: "Many vendors are confused by the term and end users are wary of it in the same way they're wary of the more general AI/ML concepts - they want to understand how much of it is marketing BS and how much of it might be of value to them." Sure, "AI" is a bit aspirational, but calling it "MLOps" sounds more like a character cut from early *Adventure Time* concept art.

IT will likely get much benefit from applying ML to it's ever growing data sets in the "punk era of IT" we now call "cloud." While it's delightful to poke fun at the name "AIOps," it's likely the usual innovation in AI that we quickly forget and then take for granted. "[T]here have been a number of successes [in AI] over the years," Moogsoft's Dominic Wellington says, "The problem is that each time, the definition of AI has been updated to exclude the recent achievement."

In one of my all time favorite analyst PDF subheadings, Forrester leaves us with this inspiring send-off: "We've Seen Too Many False Prophets, But A Messiah Is Coming." Cancel your haircuts, my BOFHthren - lo! - make ready to wash feet!

*Originally published in* The Register, June 5th, 2018 *as "AIOps they did it again, played with your heart, new acronym shame."*

# So you want to become a software company? 7 tips to not screw it up.

Hey, I've not only seen this movie before, I did some script treatments:

Chief Executive Officer John Chambers is aggressively pursuing software takeovers as he seeks to turn a company once known for Internet plumbing products such as routers into the world's No. 1 information-technology company.

…

Cisco is primarily targeting developers of security, data-analysis and collaboration tools, as well as cloud-related technology, Chambers said in an interview last month.

Good for them. Cisco has consistently done a good job filling out its portfolio and is far from the one-trick pony people think it is (last I checked, they do well with converged infrastructure, or integrated systems, or whatever we're supposed to call it now). They actually have a (clearly from lack of mention in the piece Chambers is quoted in) little known-about software portfolio already.

Most every large, traditional systems company wants to, sooner or later, get into software. Here's some tips:

# 1) Don't buy already successful companies, they'll soon be old, tired companies

Software follows a strange loop. Unlike hardware where (more or less) we keep making the same products better, in software we like to rewrite the same old things every five years or so, throwing out any "winners" from the previous regime. Examples here are APM, middleware, analytics, CRM, web browsers…well…every category except maybe Microsoft Office (even that is going bonkers in the email and calendaring space, and you can see Microsoft "re-writing" there as well [at last, thankfully]). You want to buy, likely, mid-stage startups that have proven that their product works and is needed in the market. They've found the new job to be done (or the old one and are re-writing the code for it!) and have a solid code-base, go-to-market, and essentially just need access to your massive resources (money, people, access to customers, and time) to grow revenue. Buy new things, which implies you can spot old vs. new things.

# 2) Get ready to pay a huge multiple

When you identify a "new thing" you're going to pay a huge multiple on company valuations of 5x, 10x, 20x, even more. You're going to think that's absurd and that you can find a better deal (TIBCO, Magic, Actuate, etc.). Trust me, in software there are no "good deals" (except once in a lifetime buys like the firesale fro Remedy). You don't walk into Tiffany's and think you're going to get a good deal, you think you're going to make your spouse happy.

## 3) "Drag" and "Synergies" are Christmas ponies

That is, they're not gonna happen on any scale that helps make the business case, move on. The effort it takes to "integrate" products and, more importantly, strategy and go-to-market, together to enabled these dreams of a "portfolio" is massive and often doesn't pan out. Are the products written in the exactly the same programming language, using exactly the same frameworks and runtimes? Unless you're Microsoft buying a .Net-based company, the answer is usually "hell no!"

Any business "synergies" are equally troublesome, unless they already exist (IBM is good at buying small and mid-sized companies who have proven out synergies by being long-time partners). It's a long-shot that you're going to create any synergies. Evaluate software assets on their own, stand-alone, not as fitting into a portfolio. You've been warned.

## 4) Educate your sales force. No, really. REALLY!

You're thinking your sales force is going to help you sell these new products. They "go up the elevator" instead of down, so will easily move these new SKUs. Yeah, good luck, buddy. Salespeople aren't that quick to learn (not because they're dumb, at all, but because that's not what you pay and train them for). You'll need to spend *a lot* of time educating them and also your field engineers. Your sales force will be one of your biggest assets (something the acquired company didn't have) so baby them and treat them well. Train them.

## 5) Start working, now, on creating a software culture, not acquiring one

The business and processes ("culture") of software is very different and particular. Do you have free coffee? Better get it. (And if that seems absurd to you, my point is proven.) Do you get excited about ideas like "fail fast"? Study and understand how software businesses run and what they do to attract and retain talent. We still don't really understand how it all works after all these years and that's the point: it's weird. There are great people (like my friend Israel Gat) who can help you, there's good philosophy too: go read all of Joel's early writing of Joel's as a start, don't let yourself get too distracted by Paul Graham (his is more about software culture for startups, who you are *not*—Graham-think is about creating large valuations, *not* extracting large profits), and just keep learning. I still don't know how it works or I'd be pointing you to the right URL. Just like with the software itself, we completely forget and re-write the culture of software canon about every five years. Good on us. Andrew has a good check-point from a few years ago that's worth watching a few times.

## 6) Read and understand Escape Velocity

This is the only book I've ever read that describes what it's like to be an "old" technology company and actually has practical advice on how to survive. Understand how the cash-cow cycle works and, more importantly for software, how to get senior leadership to support a cycle/culture of business renewal, not just customer renewal.

Finally, I spotted a reference to Stall Points in one of Chambers' talks the other day which is encouraging. While a bit dated, the

book makes a good case that 87% of companies stall out, unable to innovate and grow. 13% win out, but that's cold comfort.

It'll be hard. And while the software's margins draw up a ball of saliva in your mouth like a hungry wolf watching a three legged calf wobble towards a pond, beware: software is not hardware.

Good luck!

*Originally published* March, 2016.

# Eventually, to do a developer strategy your execs have to take a leap of faith

I talked with an old colleague about pitching a developer-based strategy recently. They're trying to convince their management chain to pay attention to developers to move their infrastructure sales. There's a huge amount of "proof" and arguments you can make to do this, but my experience in these kinds of projects has taught me that, eventually, the executive in charge just has to take a leap of faith. There's no perfect slide that proves developers matter. As with all great strategies, there's a stack of work, but the final call has to be pure judgement, a leap of faith.

## "Why are they using Amazon instead of our multi-billion dollar suite?"

You know the story. Many of the folks in the IT vendor world have had a great, multi-decade run in selling infrastructure (hardware and software). All the sudden (well, starting about ten years ago), this cloud stuff comes along, and then things look weird. Why aren't they just using our products? To cap it off, you have Apple in mobile just screwing the crap out of the analogous incumbents there.

But, in cloud, if you're not the leaders, you're obsessed with appealing to developers and operators. You know you can have a

"go up the elevator" sale (sell to executives who mandate the use of technology), but you also see "down the elevator" people helping or hindering here. People complain about that SOAP interface, for some reason they like Docker before it's even GA'ed, and they keep using these free tools instead of buying yours.

It's not always the case that appealing to the "coal-facers" (developers and operators) is helpful, but chances are high that if you're in the infrastructure part of the IT vendor world, you should think about it.

So, you have The Big Meeting. You lay out some charts, probably reference RedMonk here and there. And then the executive(s) still isn't convinced. "Meh," as one systems management vendor exec said to me most recently, "everyone knows developers don't pay for anything." And then, that's the end.

## There is no smoking gun

If you can't use Microsoft, IBM, Apple, and open source itself (developers like it not just because it's free, but because they actually like the tools!) as historic proof, you're sort of lost. Perhaps someone has worked out a good, management consultant strategy-toned "lessons learned" from those companies, but I've never seen it. And believe me, I've spent months looking when I was at Dell working on strategy. Stephen O'Grady's The New Kingmakers is great and has all the material, but it's not in that much needed management consulting tone/style. (I'm ashamed to admit I haven't read his most recent book yet, maybe there's some in there.)

Of course, if Microsoft and Apple don't work out as examples of "leaders," don't even think of deploying all the whacky consumer-space folks out like Twitter and Facebook, or something as detailed as Hudson/Jenkins or Oracle DB/MySQL/MariaDB.

I think SolarWinds might be an interesting example, and if Dell can figure out applying that model to their Software Group, it'd make

a good case study. Both of these are not "developer" stories, but "operator" ones; same structural strategy.

## Eventually, they just have to "get it"

All of this has lead me to believe that, eventually, the executives have to just take a leap of faith and "get it." There's only so much work you can do—slides and meetings—before you're wasting your time if that epiphany doesn't happen.

*Originally published* March 31st, 2016.

# A Note on This Text

Who has time to truly finish a book? Not me. This one container typos, misspellings, and words spelled in that strange form of English, British.

If you are kind enough to do so, please send me any errors you find: digitalwtf@cote.wtf.

Coté

*Amsterdam, June, 2019*

# Change Log

- June 5th, 2019 - declared done, changed cover to detail "The Tower of Babel."
- April 12th, 2019 - first version on Leanpub.
- May 6th, 2019 - removed chapter 4, "BigCo" stuff. Should be its own booklet. Also fixed British spelling of "organisation." That's madening.

# About the Author

Michael Coté currently works in marketing at Pivotal. He's been an industry analyst at RedMonk and 451 Research, worked in corporate strategy and M&A at Dell in software and cloud, and was a programmer for a decade before all that.

Coté does several technology podcasts (such as Software Defined Talk), writes frequently on how large organizations struggle and succeed with agile development and DevOps. His book Monolithic Transformation collects home-spun tales of digital transformation. He blogs at cote.io and is @cote in Twitter.

Texas Forever!