DevOps Perspectives 6

Actionable Insights from Leading DevOps Practitioners

ca technologies

DevOps and the Art of Maturity

Whisper it gently, but DevOps isn't an automatic choice for developers. In fact, it's entirely possible that attempts to introduce a DevOps culture into an organization might be met with resistance from the very community that would reasonably be expected to welcome it.

It's perhaps a sign of the growing maturity of DevOps that such a scenario is actually feasible and that it's a common-enough phenomenon for serious consideration to have taken place around tactics and techniques to overcome it.

That growing maturity is a common theme across this edition. As well as looking at that sort of counterintuitive reaction to DevOps, we look at how other technology areas are touching on the subject—for example, how to bring DevOps into the realms of big data, data science and analytics.

Or consider security, every organization's top priority until events come along that prove it wasn't really after all. And by that time it's too late. So how do we build security into DevOps? Step forward DevSecOps, a concept more discovered than invented and as such carrying with it real-world credibility rather than academic or technical theory.

Of course, there's no better way to consider DevOps maturity than to tap into real-world use cases, and there are few better examples than online grocery firm Ocado, which shares some of its learnings on growing a DevOps culture with us.

Contents

Nudge Theory: An Introduction

Sarah Wells

"You build it, you run it" is a fine principle, but it means you need to let your teams make their own choices. No one wants to support a system running on (insert inappropriate or flaky technology here) just because that's the company's recommended queue technology or data store.

But what's the implication for ongoing support of your services when you end up with multiple content delivery networks (CDNs), data stores, queuing technologies, issue-tracking systems, communication tools, build and deployment tools, and languages? It's fine when a big team is working on the shiny new thing, but what happens when they leave and you have five people supporting all the legacy stuff?

And how can the technology leadership make sure program teams still pay attention to department goals that may not match their short-term incentives? You want to save costs on Amazon Web Services (AWS); they want to get stuff out there and optimize virtual machine (VM) size later.

At the Financial Times, where I work on building a semantic publishing platform, teams are pretty empowered to make the right decisions for themselves, but this means they're resistant to top-down diktats. As a result, company leadership has to find other ways to influence people to do the right thing.

Luckily, there's a fair amount of information out there on how to influence people rather than force them to do things, particularly in the realm of government.

In this article, I'm going to describe nudge theory and talk about why I think it's relevant for software development.

"If you want someone to do something, you should make it easy—both to understand what the benefit is and to take the appropriate action."

What Is Nudge Theory?

"A 'nudge' is essentially a means of encouraging or guiding behavior."
–David Halpern, Inside the Nudge Unit

Nudges try to influence you rather than force you: putting healthy food on display at eye level rather than banning sales of junk food; making sure there are litter bins available and signs explaining that most people throw litter in bins rather than imposing fines on the spot.

Nudge theory was named and popularized in a book by Richard Thaler and Cass Sunstein, "Nudge: Improving Decisions About Health, Wealth, and Happiness." It's proven attractive to governments because it's about small changes, avoiding legislation (which is costly) or financial incentives (again, costly).

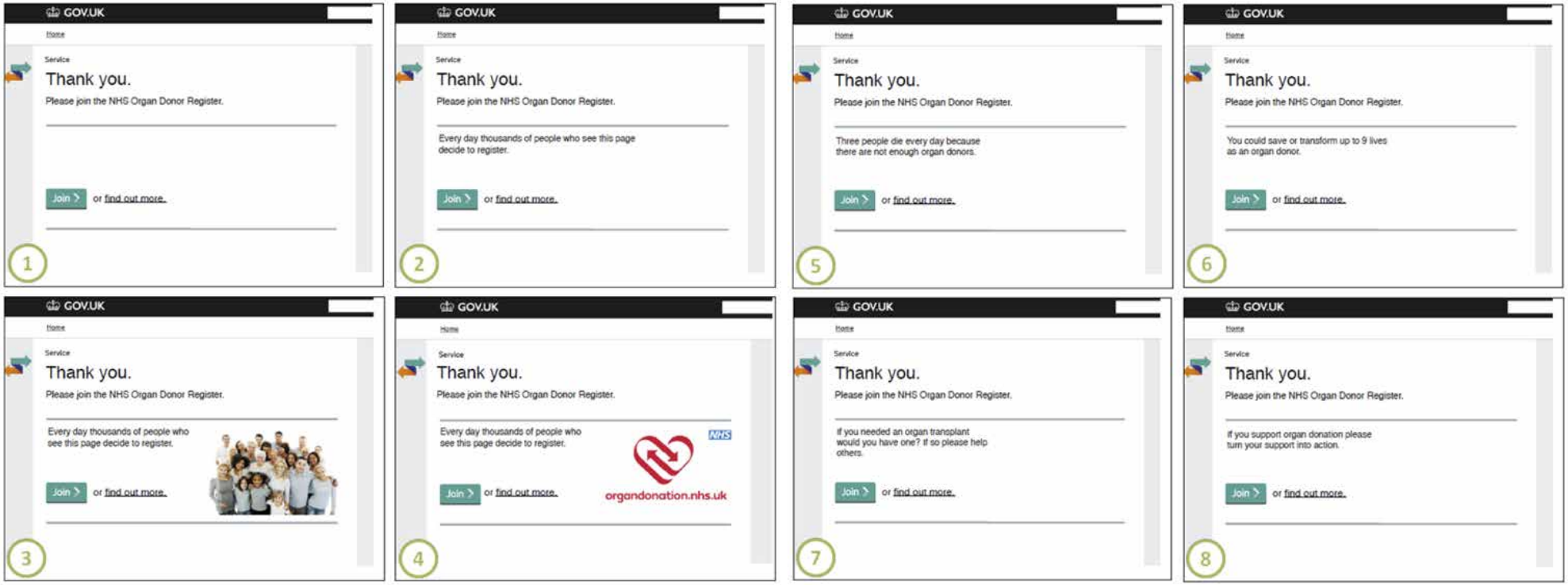
Schiphol Airport Urinals

The canonical example is in Schiphol Airport. The black fleck in the picture is a painting of a fly. It turns out that men like to aim at something, and the fly is in the best place to avoid splashback. This simple change has reduced cleaning costs by 20 percent.

U.K. Organ Donation Register

Another example relates to the U.K. organ donation register. This is an opt-in register, and although 90 percent of people support the idea of organ donation, only 30 percent of people have signed up to the register.

When you renew your car tax online, you're prompted to sign up. The Nudge Unit ran a randomized controlled trial, with eight alternative versions of the sign-up screen.



The variants were:

1. The control

2. States what other people do in the same situation (there's evidence that we're affected by social norms)

3. and 4. Same thing as 2, but with pictures added (previous research suggested adding relevant pictures increases the chance of someone donating to charity)

5. Framed in terms of negative consequences

6. Framed in terms of positive consequences

7. Framed in terms of reciprocity: Do to others what you want done to you

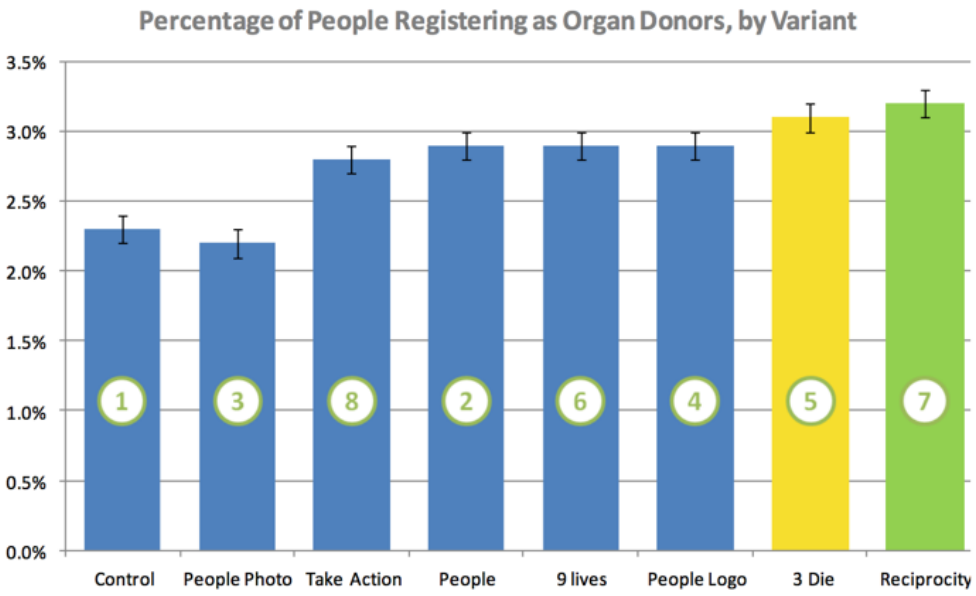
8. Pointing out the gap between the 90 percent who support organ donations and the 30 percent who actually register

The results were statistically significant:

Reciprocity does the best. Notable is that adding a picture of people to the social norms option was worse than the control, and framing in terms of negative outcome worked better than framing in terms of positive outcome.

What does this mean in terms of numbers? Well, in a year the difference between control and the best alternative would be around 96,000 extra registrations. Given that there are about 21.8 million people registered, that's about a 0.4 percent increase in the total number of organ donors for the sake of a few changes to a webpage.

How Can We Apply This to Software



Development?

The Nudge Unit came up with an acronym, EAST. They suggest that effective nudges are:

- Easy
- Attractive
- Social
- Timely

I think we can use each of these aspects to more effectively influence teams to do the things we care about as a company.

Easy

If you want someone to do something, you should make it easy—both to understand what the benefit is and to take the appropriate action.

Clear, simple messages and a good choice of defaults go a long way. We have a strong tendency to stick with the default, which is why it matters whether something is opt-in or opt-out.

Things we can do to make stuff easy:

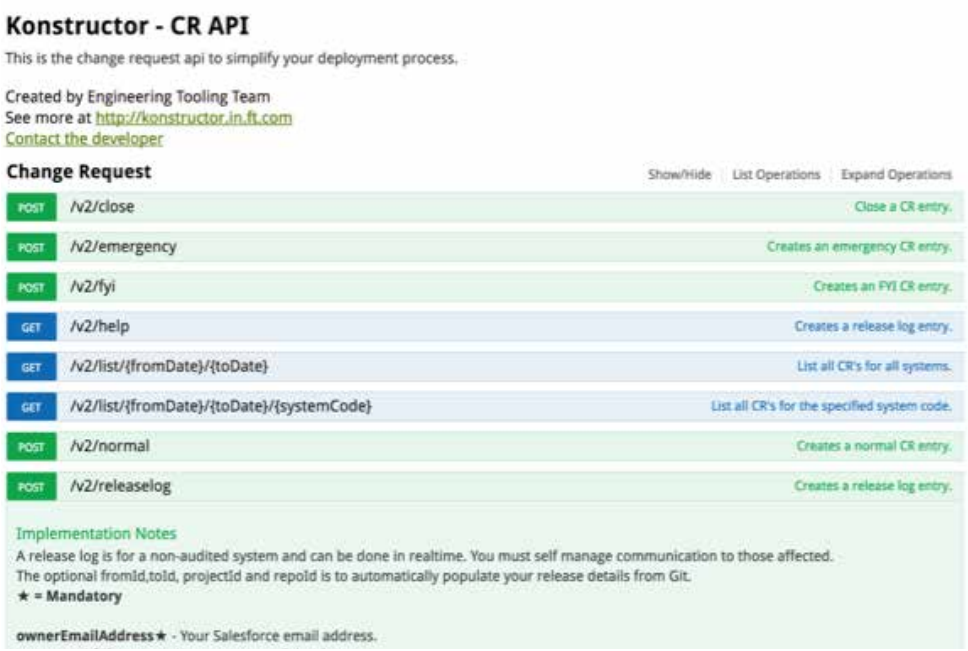
- Supply checklists, APIs, example code, libraries.
- Allow people to try your stuff out without having to wait for someone to allocate a key or set up a user account.
- Be customer-focused: Make sure people know whom to contact. Anytime I see a tumbleweed icon in a team's slack channel, I wince.

At the Financial Times, we have APIs for creating change requests, which have replaced Salesforce forms.

However, the team has made things even easier by also supplying shell scripts and GitHub webhooks. I integrated the change requests for our systems in minutes thanks to this.

We use the power of defaults with our AWS instances, which default in our staging environments to being shut down overnight and on weekends. Teams have to actively choose to keep them running outside normal working hours rather than actively choosing to turn them off.

Attractive



There are two senses in which you need to make something attractive: First, you need to attract people's attention so they know what you're asking them to do. Then you need to make that thing attractive by explaining why they should want to do this.

An example at the Financial Times comes from our security team. They want us to use WhiteSource, a tool for scanning libraries in various languages to look for known vulnerabilities.

They've documented it comprehensively to explain what it is and why we should use it. They also have a one-pager for getting started that tells you exactly what you need to do and shows the languages that can use this. I'd assumed there wouldn't yet be support for Go, but I quickly realized from this one-pager that my assumption was wrong.

Social

Humans are social animals. We're influenced by what other people do: Sending someone a letter telling them 95 percent of people pay their tax on time is proven to make it more likely that they will do that too.

In software development, we can show how other people are doing: For example, wherever possible, we want our teams at the Financial Times to migrate to Amazon Linux, because it saves us money—and we have a website that shows each team how much they would save:

We also encourage people to talk about things that have worked for them, in lightning talks or blog posts. When we share information we can show how easy it is to do something and show what people can gain from it. My team adopted Graphite and Grafana because of a really good lightning talk that showed how useful it would be and how easy it would be to integrate it with our systems.

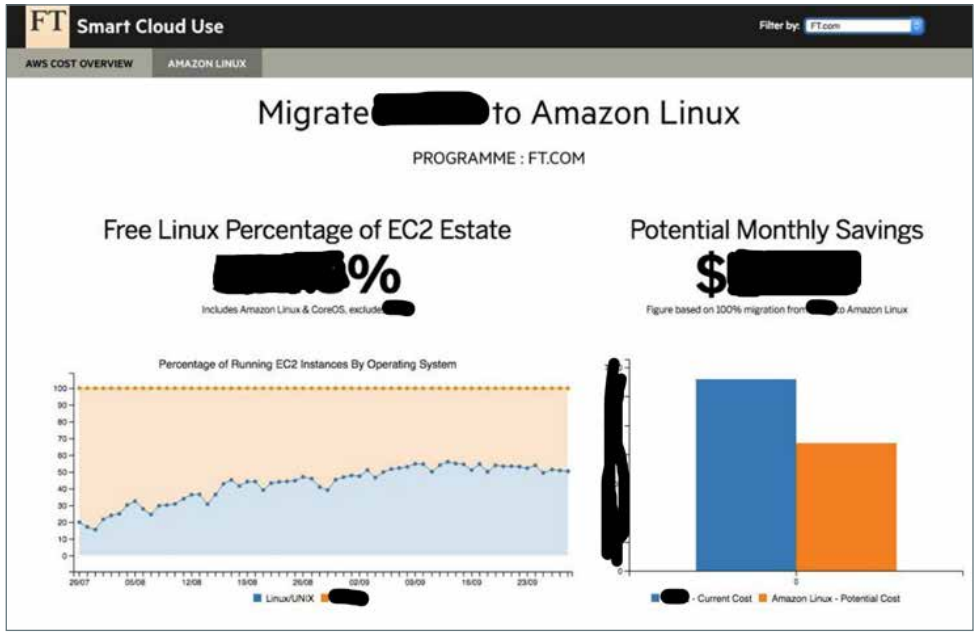
Timely

It's important to pick the right time to ask people to do things. Usually, that's when they are just starting to think about how they are going to approach their problem.

If you can provide a solution that already provides most of what the team wants, people are very likely to opt for that.

We have an engineering checklist at the Financial Times that covers the things we expect teams to do.

Here's part of it:



For each item on the checklist, we link to documentation, libraries, code examples, etc. When teams need to get something done, they're given all the relevant information right then and there.

Having a checklist also makes it easy to do the right thing and applies some social pressure on teams to do what the other teams are also doing, so it works across several of the levers.

Conclusion

My colleague Matt Chadburn has a great blog post about how a free market economy can be a great thing within a company: Let teams pick the best tools, either internal or external. This encourages internal teams to behave like service providers: They need to build great tools, easy to use, well documented and fit for purpose.

Internal teams don't have a captive market, but they should have a massive advantage. They have their customers right there, and they only need to build tools for a single specific situation.

I think our tooling teams now understand this. They changed the way they worked. They started creating small individual tools and APIs. This means as a client, I can pick and choose. It's a bit like the UNIX® philosophy in that it favors composability over monolithic design. We now have a set of tools that each does one thing, and does it well. The teams can then compose those however they wish.

How can an internal team persuade other teams to pick their tools rather than going externally? By making sure those teams know how easy it is to use the tools, the benefits they can get from them and that other people are using them successfully, and making sure they get this information at a time that's relevant to them. This means nudge theory and EAST can help.

How to Fail at **DevSecOps** (and How Not To)

Greg Bledsoe

Security is everyone's concern. That, of course, ought to be one of the basic mantras baked into every organization on the planet by now. Looking at the impact of security breaches on companies such as Target or Yahoo should provide cautionary affirmation of it.

In reality security is a fluid concept, one whose requirements change and evolve over time. One thing that can be agreed upon, however, is that traditional perimeter security is no longer an option, particularly in a culture of DevOps-enabled iterative systems development.

Enter DevSecOps, an idea that has been discovered more than invented, says Greg Bledsoe, managing consultant at Accenture. At the heart of this is collaboration. When there are global development teams, operations teams and other cross-functional teams, there needs also to be a way of making security part of the structure.

"It's about making security everybody's problem," says Bledsoe. "You're re-defining everyone's definition of 'done.' Nothing is 'done' until is it is secure. That's what matters. If something is not secure, then nothing else matters."

With that in mind, Bledsoe predicates that there are five ways for organizations to get this wrong—and, equally, five ways to get it right.

The first way to fail is simply to do nothing. This is unfortunately commonplace, with people and organizations seemingly content to continue to repeat the errors of the past, even when they know something is not working. "Why is SQL injection still a thing?" asks Bledsoe by way of illustration. "SQL injection has been a well-known problem for 15 years, and we've known for at least that long how to avoid it, but every day people still release new applications with SQL injection flaws."

A solution here is to cultivate more of a shared perspective within the organization, casting new light on old problems. Bring in the security people to talk

to the business owners and answer some questions jointly, suggests Bledsoe. For example, is the data going to need to be encrypted when at rest or in transit or both? Answer such questions and adapt accordingly.

"The flaws are going to be in the coding," notes Bledsoe. "That's where they need to be addressed. The way we do security traditionally doesn't work. It has to be baked in. Doing nothing is not an option."

Even if you avoid doing nothing, the next danger is to wait too long to do something. Specifically, the danger lies in waiting until every i has been dotted and every t has been crossed before deciding to take action. That's everything that's wrong with the traditional waterfall approach to development, Bledsoe points out.

"It's the idea that you don't do anything until you understand every possible impact of every possible action," he says. "That doesn't work; you can't know all of the things all of the time. You could be getting started and learning early on what works and what doesn't. You literally cannot wait around. Wherever you are, whatever your position, you need to get started. You need to work out what you don't know, at least. But do something. It's the idea of the journey of a thousand miles beginning with a single step. You have to take a step!"

The third way to ensure failure is to avoid engaging the security people at the project planning stage. The temptation might well be to wait until code has been written before involving them, but that's too late. Their contribution needs to be considered and factored in from the very start, and certainly at the project scoping and planning stages.

Fourth up is not paying enough attention to the basics and failing there as a result. Bledsoe cites the example of making a basic error such as not encrypting data. "There's often a fundamental level of negligence," he cautions. "People don't do the most basic things correctly. So you have situations where there is no password policy in place, for example, or people are still using 12345 as their password!"

> "It's about making security everybody's problem. You're re-defining everyone's definition of done.' Nothing is 'done' until is it is secure... If something is not secure, then nothing else matters."

The fifth and final way to fail, according to Bledsoe, is to have ineffective or non-existent policies in place. This can be caused by going from one extreme to another. Clearly you don't want a situation where there is no policy in place. Equally, there's no point in having a security policy that runs to 800 pages, as no one is ever likely to read that from cover to cover.

"There's a danger of making something too secure, with the result that it becomes less secure in the process," he warns. "If you lock things down too much, it can have the reverse effect. People will disregard things."

Getting It Right

With those cautionary notes in mind, Bledsoe offers up some positive advice on how to ensure that a DevSecOps culture can succeed. It starts from the top, he says—the very, very top. "Security has to be built in at the top level of an organization and then roll down. A recent study found that 60 percent of CEOs said that security was someone else's problem. That is the problem right there—if security is not seen to be important to the CEO, then it's not going to be important to anyone else."

So get security on the CEO boardroom agenda. Second, make sure that security awareness among the development team is built in from day one. Where Bledsoe was the VP of operations, new developers were trained in secure coding practices starting on the first day on the job.

Third, keep policies clear and simple, but forceful. Less is more, and is more likely to deliver beneficial results. "There has to be a password policy, there has to be a device policy, but keep them to a page," advises

Bledsoe. “It’s important to let people know what the standards are, but keep those simple and enforceable. And update and communicate those policies regularly.”

Fourth, think about the 80/20 rule. There’s a law of diminishing returns in play here. Hit the biggest things first, then address the rest as and when you need to.

Finally, test everything. Leave nothing off the table. From day one, there needs to be testing going on. Also at Bledsoe’s last VP job, internal staffers were trained as penetration (or pen) testers, and the company also hired external third parties to run their own tests from the outside. “When you know that there’s pen testing going on all the time, you focus constantly on thinking about what holes there are in the systems and how to address them as you go along,” Bledsoe says.

So that’s the theory. How it works in practice will obviously vary from organization to organization. At the end of the day, this is about a cultural mind-set as much as anything else, moving away from traditional ideas about security. But, says Bledsoe, “It’s getting there.”

> “The flaws are going to be in the coding.” “That’s where they need to be addressed. The way we do security traditionally doesn’t work. It has to be baked in. Doing nothing is not an option.”

Testing Times at The Guardian

Amy Hughes

Here's the dilemma: As a DevOps team, you've moved to continuous delivery, and you're pumping out new features on a fast and furious basis. But there's one problem, and that is, how do you gauge which of these new features is having the most impact? And if you don't understand that, then how can you effectively prioritize the next iterations of the code that you should pursue?

That was the situation that The Guardian newspaper found itself in. It was a position made all the more difficult by available data being potentially compromised by other external factors. As a news organization, The Guardian will inevitably see spikes in usage that mirror major events in the news cycle. If those spikes coincide with a new piece of code being deployed, is the increased usage attributable to the new software or to the latest pronouncement from Donald Trump?

For The Guardian, the answer lay in A/B testing. This enables the DevOps team to treat feature releases almost as scientific experiments, withholding changes from a portion of the audience to understand the impact of the new code based on the different reactions of the main and the control groups.

It's an approach that is paying dividends, says Amy Hughes, part of the 150-strong team working on digital projects at the newspaper. "When I joined three years ago, we were already committed to continuous delivery, and we were releasing changes to code hundreds of times a day," she says. "That was all very well, producing new features, but we needed to understand which ones were working. We were tracking and analyzing all the data that we had to do with the features, but our team really hadn't been optimized to analyze data."

"It's hard as a news site not to be impacted by the news cycle. If you released 14 changes on … the day Donald Trump got elected, then that is going to have an impact on your traffic for the day."

The problem The Guardian had can be summed up by the project to release a new feature on the main website, called The Minute. This was intended to encourage people to revisit the site more often—by presenting content in an attractive and digestible manner—and encourage greater engagement with stories.

"We looked at users who had seen The Minute and those who hadn't and used a metric of frequency of visits to judge whether it was working," explains Hughes. "We found that visit frequency had gone up, but it was launched to coincide with the U.S. presidential election. We couldn't be sure it wasn't interest in the topic that brought people back rather than the new feature itself."

This was a basic problem for The Guardian. "It's hard as a news site not to be impacted by the news cycle," notes Hughes. "If you released 14 changes on a particular day and that day was the ninth of November, the day Donald Trump got elected, then that is going to have an impact on your traffic for the day."

The challenge then is, how do you isolate data specific to the new features themselves and not have the results influenced by macro-events? What The Guardian team found, after consulting with their data scientists, was that A/B testing was the only method that would work and not slow down their continuous delivery culture.

The theory here is simple enough. "When you release a new feature, you split your test audience into two groups," says Hughes. "You have a variant who get to see the changed code feature, and you have a control group who do not. You release the feature, old and new, at the same time. Any changes you then see in their behavior are most likely down to the feature itself."

What's important here is to ensure that the test results will have statistical significance. That leads to a three-stage approach to A/B testing, beginning with the creation of the segmented test audience and then the collection of data about their behaviors. Finally, look for differences in those behaviors and analyze those for statistical significance.

To begin with, The Guardian used an off-the-shelf tool, but as the cost of such tools scales according to the size of the audience, this became untenable. The solution was to develop an in-house A/B testing framework. For the first two stages, this was rapidly achieved using open source solutions. But for the third—and most crucial—stage, it wasn't as straightforward.

"We tried to do analysis on results manually, but this was tricky," says Hughes. "Statistical significance is not a trivial thing to understand. It needs a knowledge of statistics that the DevOps team didn't have. So we'd end up serving up the tests to the segmented audience, collecting the data and then taking the results to Harry, our data scientist."

Having to be dependent on a third party for analysis, no matter how cooperative that party is, inevitably resulted in the process hitting blockages. "It would take time to come back with results, and that was a bottleneck," says Hughes. "We needed to get things out at speed. So we needed to automate the analysis stage."

That was a nontrivial task, and one that was dependent on a team of six developers working in tandem with Harry, the data scientist, but the end result three months later was a tool called Abacus. "Harry was a really large part of the process," says Hughes, even though it might have been suspected that the data scientists were being 'automated' out of their role. "The people who were being asked to do this analysis work for us were the ones who were most keen on getting it automated."

Among the learnings that The Guardian gained during this process was that features with lots of traffic are quickest to test. "We can A/B test our headlines. They get a lot of traffic, 10 million visitors a day. So we can put one version out and then another, and we can see which gets the best click-through in about 15 minutes. So we can make decisions without slowing us down particularly."

Other tests aren't as quick. The Minute is again a case in point. The editorial team at The Guardian was enthused by this feature and wanted it to be used again on other major events, but the DevOps team felt there wasn't enough information to know whether it had had the effect on the audience's behavior that it was intended to.

When an A/B test was run using The Minute as a new experience variant and the normal news format as the control, the result of the subsequent analysis was that there was no statistically significant change in reader behavior. It took two or three weeks to reach that statistically significant tipping point, but it was time well spent. If the DevOps team had committed to doing more iterations of The Minute for other events, it would have tied up lots of people over a long period of time, at some cost but with no particular beneficial impact.

For The Guardian, A/B testing has been an evident success. It may not always be applicable for all organizations or all features—if a new feature is not going to have enough of a user audience or traffic, then getting to that statistically significant point may well take too long.

But Hughes concludes: "We are at an advantage because our products do get a lot of traffic, so A/B testing is a pretty easy choice for us."

Site Reliability **Engineering**—It's a Kind of Magic

Pete Waterhouse

Hagrid: You're a wizard, Harry.

Harry Potter: I'm a what?

Hagrid; A wizard. And a thumping good one at that, I'd wager…once you train up a little.

The best bit from the first Harry Potter movie aptly describes how it must feel to be a site reliability engineer. And what's a site reliability engineer (SRE), I hear you ask? Well, it's the IT equivalent of a wizard, or as Andrew Widdowson, an SRE at Google, described it, "Like being part of the world's most intense pit crew…changing the tires of a race car as it's going 100 miles an hour."

So how is an SRE any different from traditional IT operations, and can a discipline originating from the world of web-scale, cloud-native unicorns ever apply to the steady-as-she-goes state of enterprise IT?

It can, because the notion of enterprise IT and a technology function being confined behind closed walls doesn't exist anymore. Now, the only way to create and conduct business at scale is through mobile and cloud—meaning the operational focus has shifted from keeping the "technology lights on" toward engineering reliability at levels never before imagined.

This is different from traditional IT operations because of its emphasis on engineering. Like any feature, reliability isn't something that's retrofitted after deployment; it's established and enhanced as software is developed, tested and released. That means establishing a new discipline, which Ben Treynor, Google's original SRE lead, describes as "what happens when a software engineer is tasked with what used to be called operations."

A Sobering Reality

It's easy to throw out yet another three-letter acronym and claim it's a magical elixir for all the problems involved with running complex IT systems. In reality, engineering reliability into distributed systems with thousands of containerized applications and microservices is a tough gig. Not least because of all the moving parts, but also because any preconceived notions about predictable system behavior no longer apply.

Take, for example, keeping watch over a modern software application. This might consist of business logic written in polyglot languages and linked to the legacy enterprise resource planning (ERP) system (custom-built, packaged or both). There will

"[Being an SRE is] like being part of the world's most intense pit crew...changing the tires of a race car as it's going 100 miles an hour."

also be a raft of databases (traditional relational for transactional support, yes, but more likely a smorgasbord of NoSQL data stores), be they in-memory, graphing or document—perhaps fronted by recently adopted Node.js. Some of this componentry will be on-premises; some will be containerized and moved to the public cloud. That might mean Docker and Kubernetes on Amazon Web Services (AWS), but maybe Microsoft® AZURE™ and Mesos—heck, why not both, for some hybrid-style resilience?

But like the old Monty Python sketch, "you'll be lucky" if this is all you ever have to manage. Depending on the nature of the business, there will also be a glut of third-party services, including payment processing and reconciliation. That's not to mention all the new Web and mobile apps interacting with core business systems through an API gateway, and possibly some analytics horsepower delivered by the likes of Hadoop and ElasticSearch.

It'll take a lot of operational wizardry to keep all that performing efficiently.

Fortune Favors the Bold

In a wonderful talk at SREcon earlier this year, Julia Evans of Stripe described the realities of managing today's complex distributed systems. What was refreshing about her presentation was the open admission that she often finds the work difficult, and how there's always a ton of new stuff to learn. As she says in her abstract, maybe just a tad like Harry Potter, she doesn't always feel like a wizard.

application (let's say some latency issue is causing an increasing number of mobile app users to abandon a booking service), how would teams address the issue? Problems like this might go unnoticed for some time, or there could be a deluge of alarms. Even when a problem is identified, where do teams find the root cause? Is it a problem with a new code release or at the API gateway? Is it a down to some weird microservice auto-scaling issue, and was that earlier CPU increase we thought was OK actually really bad?

But beyond exposing new normal application weirdness and "unknown unknowns," modern tools also encourage and stimulate more of the SRE detective work—the real valuable stuff. These tools won't just detect anomalies and then leave teams scrambling to find the needle in a haystack of needles. No sir: They'll analytically gather all the evidence and lead teams in fact-based fashion toward a solution. Like, for example, using an SRE-inspired monitoring service to detect a performance anomaly introduced

> "Like any feature, reliability isn't something that's retrofitted after deployment; it's established and enhanced as software is developed, tested and released."

This honesty illustrates what's exciting about being an SRE. With systems like those described above causing any number of thorny problems, it'll be the inquisitive and brave that keep business on track. Being an SRE isn't for the faint of heart or those happy with a fire-fighting status quo. It's for those within our ranks who get bored easily—those super sleuths who keep asking reliability questions, crafting improvements and learning as they go.

So if we consider a typical business-critical problem that could impact our aforementioned omni-channel

With an SRE-style approach, business critical problems are never addressed in knee-jerk fashion. Using modern tooling in areas such as application performance management and app analytics, SREs can observe the real-time behavior of applications, with systems collecting and correlating information from all related components. Rather than react after the fact, these solutions continuously identify anomalous patterns (like those mobile app abandonments) and compare them to historical trends—meaning SREs are alerted well before the business is impacted.

with a new software build and then tracing to the actual code causing the problem.

Like Potter, operations professionals might have a hard time accepting that they're wizards. But ask yourself this: Do you want to remain a silly muggle getting burnt out by constant fire-fighting? Of course not; it's career limiting, and it sucks. Time, then, for some SRE magic— gaining the skills and tools needed to adopt new tech like containers and microservices—becoming an essential part of future-proofing your business.

You're a wizard, right?

Making Delivery **Continuous**

Leena N

Continuous delivery is built upon the premise of ensuring that code is always in a deployable state, even when you have large teams of developers iterating and making changes on a daily basis. The objective is to get changes—from new features to bug fixes to configuration adjustments—into the hands of the user quickly and securely, eliminating the notion of post-dev-complete integration, testing and hardening phases.

That sounds like an objective that every organization can get behind, but as with any change of approach, introducing continuous delivery to a development culture can be a challenge. There are ways to overcome resistance to change, says Leena N, cofounder and head of engineering at Multunus Software. These are often relatively simple practices, but ones that can help to introduce continuous delivery core elements such as monitoring, build automation and test automation gradually.

First up is the need to recognize the problems that continuous delivery is intended to address. Leena cites the fictitious example of Bob. Bob is an entrepreneur with lots of business plans and some big ambitions. To deliver on these and build the products that will make him successful, Bob hires a group of highly talented individuals, who come together at the start of the project full of enthusiasm and a spirit of cooperation.

Fast-forward a few months, and they're fighting like cats and dogs. "What's happened is that they all went off into silos to build their own things," Leena explains. "Then they came out and started to try to integrate everything and found that bugs started cropping up. Those bugs then started breaking other things. Then Bob realized that what he asked for and what's now built are very different things. So the team starts talking about things not beings bugs, but rather being features, because they need to release this somehow. But when they do release it, it's not going to get the users it needs to get, because it's not the product they set out to build."

A large part of the problem here is that the development team set out to build the complete product all at once. "That's not a good position to be in," Leena says. "Experimentation is the key to a lean start-up, and you need good technical support for that.

The other problem is finding yourself stuck in the last mile. How much time does it take for a one-line change in the code? Maybe it's just a color change for a button, for example, but it's going to delay you for days or weeks."

This is where continuous delivery's basic premise kicks in: being confident that code is releasable to production at any time. "You need a development pipeline," says Leena. "You need to know how the code travels from developer to production—the build, test and deploy stages."

It's continuous delivery when all developer working copies are merged to a shared mainline several times a day. So the DevOps team needs to set up a mainline development trunk, or master, against which every new build and addition can be tested and compared to ensure the changes are OK.

"You need a development pipeline. You need to know how the code travels from developer to production—the build, test and deploy stages."

Some features, such as security compliance, for example, will need more time and attention than others. To avoid these slowing down the overall project progress, feature toggles should be introduced as a key enabler of continuous delivery. These are small setups in the configuration that allow developers to turn on or off specific features depending on conditions. They instruct the code to do one set of actions if a certain condition is true and another set if another condition is true.

There are different types of feature toggles. Release toggles allow for the separation of deployment and release so that developers can continuously deploy without actually releasing code. This allows them to quickly roll back on any ticks that emerge.

Experimental toggles are used to test multiple versions of the same app or page with different implementations or features. "From experimenting with different versions, you can learn things and implement accordingly," says Leena. "Certain things in certain versions may be better than in others, so you can choose which is which and use the best. You experiment and see what works the best for the user. Because you're doing a gradual rollout, you can see how users are reacting as you go along and get feedback on design and architecture. Importantly, if there is a problem, you can release again."

Meanwhile, operations toggles are designed for failure scenarios and are short-lived. These are toggles created for a certain time and set of circumstances that are then deleted. "If you have part of your application that is dependent on a third party API and there's a problem with that API, then it will affect your application," notes Leena. "So to implement a circuit breaker in this, you use operations toggles. If an API then doesn't complete in time, the application can use a cached version, or it can shut off the feature for a time. There will be a degraded experience for the user, but that's better than bringing the entire system down. It's a very good way of controlling dependencies that are not under your direct control."

Leena cites a specific example from her own experience of a client who needed a system to support its custom T-shirt business. To create a design, customers would access a library of images. "After a certain time, we realized that the library we were using didn't support a certain feature very well. We wanted to move on to another implementation, so we introduced an abstraction layer. The current library worked well in certain conditions for certain features, so sometimes the system could use that. Otherwise it would use the abstraction layer to ensure that the library choice was the correct one for the specific conditions."

There's no such thing as a perfect solution, though, Leena cautions, and feature toggles do have downsides, most notably in terms of introducing a degree of complexity.

But the main benefit lies in creating more business impact by delivering secure code and product as quickly and cost-effectively as possible. "In my opinion, the only way to create a business-supportive engineering team is to have continuous delivery in place," Leena says. "You need to deploy continuously, so you need that framework in place that is going to allow you to make changes as quickly and non-disruptively as possible."

Why **DevOps** Delivers at Ocado

Kevin McCormack and Alex Howard Whittaker

In the increasingly competitive market for online grocery services, the most celebrated example in the U.K. remains Ocado. While offline supermarkets such as Tesco and Sainsbury's have added digital extensions, Ocado was built from the ground up as an online retailer, powered by its own technology division, Ocado Technology.

Since its founding in 2000 with three people in a room in London, Ocado has become the world's largest online-only grocery store. In the U.K., it boasts a reach that covers over 70 percent of households nationwide, shipping out more than 230,000 orders per week.

At the heart of this are Ocado Technology and the power of online retail and fulfillment systems that have been built for this purpose. As the firm notes on its website, "We wanted to start from scratch and build a unique online shopping and fulfilment solution that would revolutionise the way people buy their groceries.

"Nobody had done this successfully before anywhere in the world, there was no blueprint to follow and off-the-shelf solution we could buy. We had to build it ourselves, and that's why almost all of the software that powers Ocado is developed in-house by Ocado Technology, including a lot of highly specialised systems that you wouldn't expect us to be using, let alone building. From the optimisation algorithms that fine tune our daily delivery routes in the 500 milliseconds of a mouse click, to the machine learning techniques that drive our consumer demand forecasting, to the real time control systems that operate our vast Customer Fulfilment Centres, there's nothing run-of-the-mill or business-as-usual here."

That's why Ocado execs talk in terms of the company being a grocery operation that has the look, feel and culture of a technology start-up. Close to 1,000 employees at Ocado are technologists of some description.

"Lots of people will tell you that we are a grocery supermarket and talk about the vans and the website

and so on. But behind the website and the colorful brands is a lot of technology that makes everything possible," says Alex Howard Whitaker, a cloud services engineer on Ocado's cloud infrastructure team. "The people who create all that technology are Ocado Technology. We have about 1,000 developers and engineers who are split between several teams, handling various parts of the retail process. These range from mobile apps and websites up to the highly automated warehouses where we process all our customer orders, pick the product and pack up the deliveries. From the warehouses, we load up the vans and send the orders out to the customers. Ocado Technology provides everything needed in terms of software, whether it's to control our automation and robotics systems or to implement some clever piece of machine learning in an area of our business."

And in the mix is a growing DevOps culture, although it's not always easy to define what that is, says Kevin McCormack, team leader CFC (Customer Fulfilment Center) DevOps. "I think it's very hard to define what a DevOps culture is," he says. "My ideas on DevOps culture is not having to set up continuous integration servers and persuading people to use them. It's about encouraging individual teams to have individual responsibility for their own applications and making them self-sufficient.

"As a development organization, there's a lot of interest in self-service and autonomy. Continuous delivery is part of the culture. There is an interest in exploring what the latest tools are around deployment and coding practice. There has always been an interest in following good development practice. So yes, Ocado really ticks a lot of the boxes of DevOps characteristics."

For his part, Howard Whitaker sees the use of DevOps techniques and practices as critical for meeting Ocado's ambitious goals, adding that the adoption of DevOps has grown almost organically, rather than being a predetermined strategic decision.

"We started moving toward architecting a microservices architecture, rather than from having a clear intention of becoming a DevOps organization," he explains. "There was a strong push from within to become an independent technology company. A lot of DevOps practices came out of necessity, as a result of other things that we wanted to pursue, things that we found ourselves shifting toward. We had complicated and monolithic systems, and we needed to get away from that, so we fell into what are now regarded as fairly standard DevOps practices.

"On the cloud side, we use Amazon Web Services (AWS) for microservices and Google Cloud for data analytics," he adds. "We use Elastic Beanstalk. What we commonly try to do is to use a set of internal self-service tools for developers to use to provision new application environments. There are common security certifications so that the environments are consistent across all the apps. We also use common deployment tools across the Amazon APIs. AWS is very flexible around what you can do in terms of customization. There are lots of people who use the Amazon AWS platform, so there's a lot of good community knowledge to be had, and it's easier to recruit good developers."

Having built its own systems successfully, one of the big projects at Ocado today is to export that expertise to third-party retailers in the form of the Ocado Smart Platform (OSP). This has involved creating a whole new Ocado end-to-end e-commerce solution, including cloud-hosted software and swarm robotics-based hardware. "That takes all the lessons that we've learned from building our own website and packing it into a white-label service that we can offer to other retailers around the world," says Howard Whitaker.

The development of OSP was helped by the use of DevOps practices, including ongoing testing and feedback loops as development progressed. This wasn't as straightforward as it might have been. "For some time, we didn't have real retailers to get feedback from," explains Howard Whitaker. "We had

> "We wanted to start from scratch and build a unique online shopping and fulfillment solution that would revolutionize the way people buy their groceries."

a team internally who were acting in the role of retailers from a product ownership point of view. Then we moved on to running an internal shop that was doing deliveries to people's desks. So we had someone acting as the picker, someone as retailer, someone as HR manager and so on. It was a way to get a feedback loop going."

The use of DevOps techniques has had another beneficial impact in that it has allowed greater transparency within the organization. "A move to continuous delivery has allowed the business to observe and manage the pace of change better," says Howard Whitaker. "From an operational perspective, Ocado must keep a close eye on the website and the warehouses. They have to keep working. Making development changes smaller and more visible is better. Large changes can result in large disruptions, and that's not good. With a move to smaller but more frequent changes, the potential for minor disruptions is less."

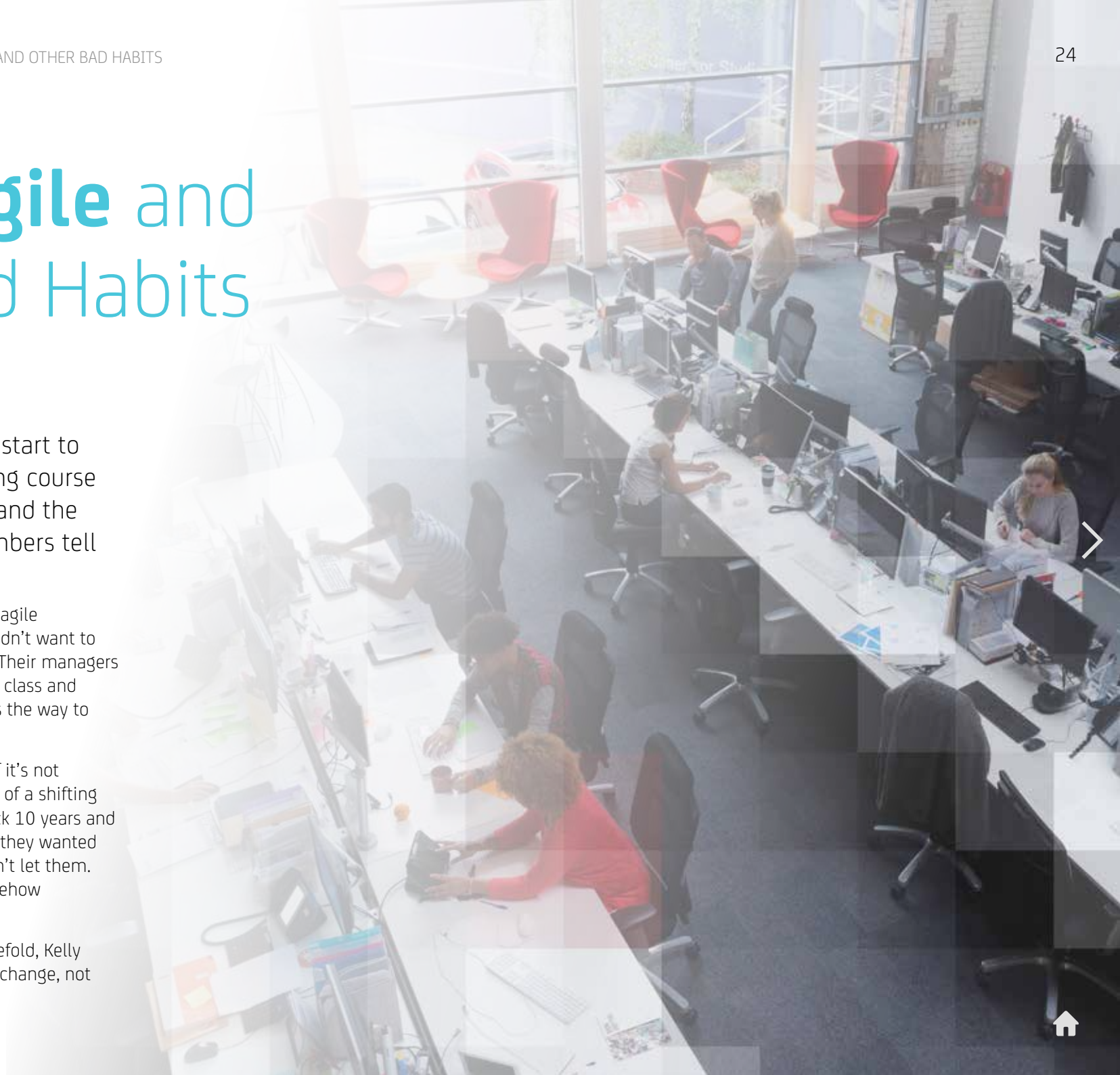
As for specific learnings gained in Ocado's journey, McCormack points to the challenges of transitioning. "The main learnings have come from the transition from using traditional infrastructure or virtual machines hosted on premises to moving to a private cloud or public cloud with containers," he says. "That brought a lot of challenges at an organizational level as some of the relationships with the infrastructure side of the business changed. If you're using something like Docker, you're not limited to what OS or what version of Java® the infrastructure team wants you to use because they don't want to have to support 20 versions. This pushes that onto the development team. It has been a bigger upheaval than most people imagined."

Howard Whitaker concurs. "It's definitely been challenging shifting to the new practices. We have to keep the lights on and keep operating at full scale," he says, but adds that it is worth the effort. "DevOps is the only way that could scale in terms of the number of people who work in the tech organization. When I joined, we had one office in the U.K."

"A move to continuous delivery has allowed the business to observe and manage the pace of change better."

Hating **Agile** and Other Bad Habits

Allan Kelly



It's not the most auspicious start to an agile development training course when you walk in the room and the first thing the audience members tell you is, "We hate agile!"

That's what happened to Allan Kelly of agile consultancy Software Strategy. "They didn't want to be in that training course," he recalls. "Their managers had gone off and done a Scrum master class and come back with the belief that this was the way to go and that was that."

It's not an uncommon problem, even if it's not always talked about. It's also indicative of a shifting balance in attitudes, says Kelly. "Go back 10 years and it was the developers who were saying they wanted to use agile, but their managers wouldn't let them. Now the managers get it but have somehow alienated the developers."

The reasons for this alienation are threefold, Kelly reckons. "First, it's seen as an imposed change, not

"Go back 10 years and it was the developers who were saying they wanted to use agile, but their managers wouldn't let them. Now the managers get it but have somehow alienated the developers."

something that you as a developer have control over. It's seen as a case of 'Thou shalt be agile,' and that leads to resentment."

Then there's the danger of the technical aspects of agile development being overlooked by management in pursuit of quicker and quicker coding. "When agile is interpreted by non-coders, you can find that the tech aspects are either underplayed or ignored altogether," says Kelly. "When you ignore the tech aspects, that just makes life harder for the developer. You know, you can crank out more stuff in half the time, but if you do, the quality is going to go down. You've got to keep top-notch quality a priority."

Kelly also warns of the associated dangers of building up technical liabilities. "The original foundation of agile said that you must have a technical element. You must keep the team real clean and effective," he says. "But you build up more and more liabilities, robbing Peter to pay Paul, and sooner or later Peter gets upset! You can run the team faster and faster, but if they don't get the chance to improve quality, then their lives are just going to get harder and harder. So you can't blame them for complaining.

"At the heart of agile is the need to keep really high levels of agility so that you can respond to change. If you have less than two-week iterations, then it should be all about highest quality. When agile is imposed on people, the technical quality is forgotten."

Finally, there's the double-edged sword nature of agile as tool of choice, that in the wrong hands it can encourage unhelpful behavior. "The one that gets the most attention is the idea that agile opens the door to a culture of micromanagement," says Kelly. "Agile tools were intended to help teams self-organize and to make individuals more responsible and more collective. But if you put those same tools into the hands of micromanagers, you find that they are amazingly good micromanagement tools as well.

"So you end up with the 9 a.m. team get-together meeting to celebrate achievements, talk about what you all are doing, make plans for the day, talk about what's making life difficult and so on. If the team is cooperating, then that's brilliant. But if you've got a micromanager there, then he or she is ticking things off as you go. It's the same tools but the wrong mind-set."

Agile techniques can also lead to people falling back into bad habits. "People in authority have that authority because they have had some past perceived success," says Kelly. "So when times get tough, they run back and try to do the same sort of stuff that made them successful in the first place. Maybe in the past, they feel that they succeeded by making people work hard.

"In the traditional project environment, the assumption is that everyone is lazy and has to be whipped into action. But in an agile community, where you want to be trying to build community and cooperation, those behaviors are the opposite of what you want. The micromanagers haven't learned that there are quite different approaches. You won't get far in the DevOps world if you make your team sweat bucket loads."

"When agile is interpreted by non-coders, you can find that the tech aspects are either underplayed or ignored altogether."

So with those barriers, the question becomes how to overcome them and get buy-in from both developers and management. "Start by appealing to self-interest and their own sense of professionalism," suggests Kelly. "You've got to say to developers that you want them to be part of a journey. This should be about managers asking how everyone can work better and saying, 'I've heard agile might be an answer, and we're prepared to spend money on this, but how do you as a team think we can improve?' "

And money will have to be spent, says Kelly. "This generation of programmers weren't taught all this at college, so you need to invest in training and coaching or they can't be effective with the newest techniques. Spending the money on training and coaching does two things. It directly trains people in the necessary skills, but more importantly, it says to your developers that this organization is prepared to invest in them. It says that this organization considers enabling you to do a good engineering job important, and that it will allow you the time and space to do things well."

It's also important to remember that you're all on the same side here—or you should be. "None of this is malevolent," says Kelly. "Agile and traditional development projects have the same goal in mind, which is to deliver the best product to the business. Whether you're working with agile or not, there's still huge pressure to deliver."

That means both managers and developers need to up their game. "When you talk through this, most management teams get it," says Kelly. "You have to feel a bit sorry for managers. If they try to impose this, they're micromanagers, and if they don't, then they're not going to get the benefits of agile. But developers also need to put their hands up and say they really want to do this. There's a cynicism on both sides that needs to be overcome."

Spotting the Future of **Anomaly** Detection

Pieter Buteneers

At its most basic, an anomaly is something in a data set that doesn't look like all the other things. Anomaly detection is the process of spotting that. It's a challenge that can scale frighteningly when you're dealing with larger and more complex systems and balancing metrics that may be highly fluid in nature. Container and microservices environments are a prime example of this.

"You need to be able to detect the unexpected. That's the best definition, I think, for anomalies—they are unexpected events," says Pieter Buteneers, data strategist and machine-learning consultant at CoScale. "What's really important is that whatever you detect has to be important to the user. If it isn't, then what you detect might not actually be an anomaly, but a smaller glitch.

"It's going to vary from case to case. There is no specific definition. What can be anomalous for one piece of data may not be anomalous, with that same data, for a different user. If Google has an error rate of 0.0001 percent, that might be really relevant to them, but for others it might be too small. It depends on the customer and on the type of data involved.

"It's a matter of perspective. It can be quite difficult to figure out what is relevant and what is not. From our domain knowledge for container environments, we have a general idea, but that doesn't mean that we're never wrong. We'll use all the information we have around the systems to work out what is normal and what isn't, but the perception might be different between business and developers."

That said, there are four possible outcomes of anomaly detection, one of which is a major problem for developers.

The ideal outcome, of course, is a true positive. If anomaly detection is working as it's supposed to, then the anomaly has been spotted and can now be dealt with in an appropriate manner.

Equally valuable is a true negative outcome. Despite sounding bad, a true negative's upside is that you haven't fallen for a false alarm. You don't have a problem, and you're not about to assign developers to fix an issue that doesn't exist.

A false positive outcome risks doing exactly that—setting the development team off on a wild goose chase for a problem that has been wrongly identified. This wastes time and resources and, as a possible side effect, may undermine confidence in the monitoring tools and techniques, both among the development team and across the wider business.

The worst possible outcome is the false negative. In this case, you do have a problem, but the anomaly detection tools haven't spotted it, so nothing is done to address the issue. Development work continues in less-than-blissful ignorance while problems lurk further down the track.

"You can speak about false positives and false negatives, those are the ones that people are interested in," says Buteneers. "There can be as many positives as there are, and people don't really care about the true negatives. It's the false positives and false negatives that are important. Those are really hard to measure.

"Those are the things that we look for. For every false positive we find, we would hope to have more true positives in the same time period. If you have one false positive and 200 true positives in the same time period, that's not too bad. But if you have one false positive for every true positive, that's alarming."

The technology to track down anomalies will typically begin with relatively simple dashboards that are able to monitor and report on basic metrics. As system development becomes more complex, the need for more sophisticated and functionally richer monitoring tools grows. Increased automation and machine learning capabilities become features of the most sophisticated detection systems once the complexity of the system precludes human visual detection.

"Detection itself is not that difficult," says Buteneers. "If it is, then it's because you're making it difficult! If you use the right tools to detect anomalies, then it shouldn't be that hard. Every now and then you read about Uber (see box) or Google, who build their own tailor-made systems, but there are tools out there. It depends on which techniques you're going to use and how much you know about those."

The challenge is to make sure that the approach to anomaly detection and the tools used are appropriate to the needs of the end organization. "With smaller companies, data is much more noisy. With larger companies, the data is a lot more stable," notes Buteneers. "If you look at the anomaly detection field, many of the systems out there generate so many false alerts that I can see why so many developers don't look at them. It should be up to developers to require of their vendors that the anomalies detected are going to be valuable."

> "What's really important is that whatever you detect has to be important to the user. If it isn't, then what you detect might not actually be an anomaly, but a smaller glitch."

Sarah Wells

Principal Engineer,
Financial Times

Sarah Wells is currently leading work at the Financial Times on building a semantic publishing platform, making it easy to discover and access all the Financial Times' published content and metadata in a common and flexible format, via APIs. Sarah has been a developer for 15 years, working across consultancy, financial services and media. She is more dev than ops, but definitely shifting. Her recent focus has been on Go, microservices, containerization, DevOps and how to influence teams to do the right things.

Greg Bledsoe

Managing Consultant,
Accenture

Greg Bledsoe is a managing consultant at Accenture in the DevOps architecture practice and regularly advises and leads the implementation of DevOps principles and practices at the Fortune 100. A regularly published author, Greg has spent a career being responsible for keeping platforms secure and reliable. He brings twenty years of experience building scalable, secure and cost effective environments. Greg is a certified ethical hacker and certified penetration tester and led a talented team with the same certifications at Personal, Inc.

Greg has been implementing DevOps principles since before the term DevOps existed and was recognized by Jax DevOps as the third on its list of the Top 20 Social Influencers in DevOps.

Amy Hughes

Software Developer,
The Guardian

Amy currently works with the Data Technology team providing real-time analytics for the Guardian's digital products.

She has been working at digital at the Guardian for three years, and she recently led a project to help teams understand whether the code they are writing is in fact changing the behavior of their audience.

Pete Waterhouse

Senior Director,
DevOps Strategy and Marketing

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

Leena N

Cofounder and Head of Engineering,
Multunus Software

Leena N is a pragmatic and passionate programmer, lean thinker and extreme programming evangelist hooked into continuous delivery.

With more than 15 years of experience in the industry, she has seen many failures in software product development either because of the team lacking the discipline in delivering software or because of it creating something that nobody wants.

Her journey of building a team at Multunus gave her a lot of opportunities to concentrate on the first problem: i.e., bringing in sustainability and predictability in software delivery using continuous delivery and extreme programming practices.

Kevin McCormack

DevOps Team Leader,
Ocado Technology

Kevin McCormack is a DevOps team leader with a Java Development background. Kevin has promoted the use of container public/private cloud solutions at Ocado Technology using Docker and CoreOS.

Alex Howard Whitaker

Cloud Infrastructure Engineer, Ocado Technology

Alex Howard Whitaker is a Cloud Infrastructure Engineer and Python developer working at Ocado Technology. When not building Ocado's AWS environment, his chief interests are automating away the boring tasks, creating a frictionless platform for developers and building software that thrives under volatility.

Allan Kelly

Agile Specialist

Allan Kelly makes digital development teams more effective and enhances delivery with agile approaches to reduce delay and risk while enhancing value. He has been called a thought leader but clients have a habit of calling him Agile Allan. Once upon a time he was a programmer; now he keeps his programming hand in but spends most of his time with teams and managers, looking at processes and helping decide what the right thing is.

Right now he's probably best known for his work on Continuous Digital: The Agile Alternative to Projects—formerly known as #NoProjects.

He works with a number of companies through his company Software Strategy Ltd., providing agile training, agile coaching/consultancy and more general business advice for digital companies.

Pieter Buteneers

Data Strategist and Machine Learning Consultant, CoScale

Pieter Buteneers started his career in academia, first as a PhD student and later as a post-doc, where he did research on machine learning, deep learning, brain computer interfaces and epilepsy. He won the first prize in the biggest Deep Learning competition of 2015 together with a team machine learners from Ghent University: the National Data Science Bowl hosted on kaggle.com. In 2016, he finished his MBA at Flanders Business School and now he works as a data strategist and machine learning consultant for CoScale and other companies.

Next Steps

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

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

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

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

#modernsoftwarefactory

CS200-289403_0717

ca technologies