puppet

Hiscox Reduces the Cost of Release by 97% with DevOps and Puppet

CASE STUDY

HISCOX

Industry

Insurance

Background

Hiscox is a publicly traded property and casualty insurance company based in the UK.

Challenge

Hiscox needed to increase the pace of change to release new products quickly into a highly competitive and fast-moving market.

Solution

Introduced DevOps practices and automated with Puppet Enterprise.

Results

- Reduced cost per release on a major application by 97%.
- Release cycle improved from 10 weeks per deployment to 50 deployments per week.
- Reduced time per release by 89%.
- Reduced staff required to release by 75%.
- Now builds VMware virtual machines twice as fast.

I joined Hiscox as a solution architect, having worked at a number of different companies and wearing lots of different hats (developer, support, infrastructure, architect, etc.). Although I had some interesting offers on the table elsewhere, it was clear that Hiscox was where I wanted to be. There is something intangible about the culture — nice people who are also courageous, entrepreneurial and ambitious.

The journey of DevOps adoption at Hiscox was inspired by a need to increase the pace of change. The business is growing fast, the market changes rapidly and it's important as an IT function that we can respond to those demands. We have to work smarter, not just harder. It really is a journey; there are wrong turnings, ups and downs, and while we've made real progress, we still have a way to go.

There are some exciting ambitions for the company for 2020, and to meet those we've been assessing our cultural, people, process and technology capabilities in order to meet (and hopefully surpass) where the business wants to get to.

Jonathan Fletcher *is an enterprise architect, and the lead for technology, platform and DevOps at Hiscox.*

@FletcherJofanon

DevOps is key in our strategy. It's helping us to do things faster at higher quality, and increasingly to work a lot closer with the business. Our ambition is really that it should not be "IT" and "the business" — we should all be working together to accomplish the same goals. IT should understand where the business wants to go and have a very close understanding of how the business works. The business should have empathy for the challenges of IT delivery and work with us to mitigate them.

We have had some great results from our DevOps initiatives so far:

- **We reduced our cost per release on one application by 97 percent.** That rolls up from:
  - Reducing time per release by 89 percent.
  - Reducing staff required to release by 75 percent.
- By automating our testing, we've reduced multiple man days of effort down to an overnight hands-free process.

These are fantastic results, but there's a huge amount more to do. I'm going to walk you through how we started on this DevOps journey at Hiscox.

First steps

The early days and weeks of the DevOps journey were spent bombarding (or probably more like boring) the CIO and CTO with PowerPoint presentations. First I presented the problem statement:

- The need for a faster pace of change (which obviously they were acutely aware of)
- The silos we had
- The fact that people were being pulled in different directions, with conflicting goals
- Expensive and slow repetitive manual tasks
- Inefficient technical processes

DevOps is about aligning goals, but IT at Hiscox wasn't set up to achieve that. For example, we had development teams that threw code over the wall to the release team that deployed it, and then a separate team supported the code once it went to production. In scenarios like this, there is very little empathy between all those different teams. It's a bit like a relay race and passing the baton — "I've done my job, now it's your problem." We are now moving to product-centric teams that have cradle-to-grave capabilities — development, support, the business, etc. This is really new and really exciting.

I talked a lot about cultural change in the beginning of our DevOps journey, talking to our teams and to executives about the challenges of micromanagement, lack of empathy and how to restructure our teams and the technologies that will help pull all this together.

While DevOps is commonly associated with increasing the pace of change, the principles and practices can also be applied when you're trying to increase quality or revenue, or when you want to make sure you're taking care of customers using a sunset product (one slated for end of life in three to four years).

Once the IT executives were on board, we started with a project called "Maximize IT." We moved people into roles that were more aligned with the business unit — "product focused" is probably the right term, as I mentioned earlier. Each product team has its own business analysts, developers, testers and other functions as needed. These teams are the masters of their own destiny, rather than trying to beg and borrow capacity from a big group function. That's why I believe the term "DevOps" is wrong. DevOps shouldn't be "Dev" and "Ops." Making change fast, at high quality and in line with business goals is everyone's responsibility, so it should be called DevOpsBizTestThingy.

Another reason we reassigned people to individual units instead of having one large development team is because when you try to meet the competing needs of different business units, you'll probably have different goals and incentives. Trying to deliver on different goals is really hard; you might have one business unit where there's a high demand for pace of change and sufficient budget, another that's dealing with a sunset application and there's not enough budget. Compromises get made when you're trying to appease all these different groups, making it pretty difficult to truly meet the needs of even one group.

We started with the U.K. business unit under a program called "Da Vinci," which recently bought a new policy administration system called Idit and paired it with a great web portal technology called Backbase, in order to increase the pace of change. We established a new product team around this system — devs, ops, business analysts, business product owner, business user acceptance testers and automation testers.

This is probably the first time we've consciously thought about how we are going to develop an application after it leaves project mode. The project worries about getting the application live, but then what happens on Day 2? How are we going to develop and support an application as part of BAU (business as usual)? We have to think about the change function as well — everything the application will need to make it successful after it's live in production, and being used by its customers.

A few learnings

Starting our DevOps journey was largely delayed by the sheer scope and size of the challenge ahead. Trying to effect process, people, technology and cultural changes across the entire application portfolio, in a globally dispersed team and with a lot of associated technical debt, is an epic challenge. If you think about deployment, testing, moving to Agile, spinning up new environments, instituting new version control strategies, etc., it's just too much to do in one hit.

We decided to limit DevOpsifying to our crown jewels: the five applications we care about most.

With this level of change and relatively little experience with DevOps in the team, mistakes inevitably happen. The important thing is to recognize those mistakes and learn from them. You need patience and belief from the top dog (CIO/CTO), because you aren't going to get it right the first, second or third time — but the results are well worth it.

We made the mistake of telling people what the problem was and what the solution would be.

People can see that only as a criticism. Now our approach is to work with people, have them help define the problem statement and then lead them to the same conclusion. That helps overcome the resistance people feel and helps accelerate the change curve. People feel bought in to the change instead of feeling, "Who's this telling me how to do my job?"

No doubt, the hardest part of DevOps is the people aspect. DevOps is a cultural idea, so it's very people-centric. It has been a learning exercise for me to watch people go through this change of mindset...and it has also been exhausting. If I'm honest, we are only partially there, although it's better than it was. I still hear "I'm a developer, why do I need to know about deployment?" from time to time. I'm not a hugely patient person, so I've found that frustrating.

We gave recommendations to the owners of the crown jewels for what needed to happen next in order to improve their delivery capabilities using some of the DevOps methodologies.

Once we rolled off and left the teams with our smug "look at how clever we are" slide decks, the things we recommended just didn't get done because other business priorities got in the way. When you're starting out with DevOps, or any big process change for that matter, you have to dedicate time to it. You are changing a lot; you can't just say, "go do DevOps" and expect people to drop everything else and get it all done at once.

What helped here was showing a proof of concept, demonstrating how DevOps created value in another area of the company. Instead of hearing, "I'm trying to tell you the right thing to do," people can see success, and then they think, "Oh, I want that service" and pull in the change themselves. They find ways to make the change a priority amid their other work.

People need to have the confidence that what you've proposed is the right thing to do. When they can see you've done it before and achieved success, they can trust that it is the right thing to do. That's the value of evangelism: When you show people you've done it and had a good outcome, it creates a different dynamic.

We needed a team to evangelize and support the adoption of DevOps.

It's all very well saying to people, "go do continuous integration," but if a team doesn't have a budget for a continuous integration server and other tools, you won't succeed.

We created a platform services team (which I run). That team learned continuous delivery and other DevOps practices by working with the DaVinci programmers. The team now owns Puppet and other tools, and the best practices for DevOps ideas and continuous delivery. We can tell you, "We've got the tools, we know the best practices, and people have already done this." The barriers to entry are lowered; other teams can get a quick start and begin realizing the benefits of a changed approach, whether it be faster delivery, better quality or a happier workforce (hopefully all of the above!).

The platform services team is five core professionals, and a number of others have a dotted line into the team. That's compared to an IT team of roughly 200 at Hiscox. It's actually hard to say just how many there are, though, because IT is mixed in with business analysts and others. It's not IT over here and the business over there — we are the business. We just happen to do a technical role.

Overcoming the test bottleneck

One of the main bottlenecks in delivering systems is verifying and testing. We looked for a new way to do this — and not just throw it to the tester after code is completed — because you can't test quality into a product. My QA architect says it's like trying to lose weight by weighing yourself more often. That little nugget got him the job!

We now have testers working with developers and business analysts from the early requirements stages. There has been a big change, from testing being something we throw offshore to testing being a first-class citizen. We've invested in test automation and funky concepts like BDD (business-driven development) and TDD (test-driven development). The test code is as important as the application code. Great testing sets your development free.

Results

The results I cited as I opened this article — time per release reduced by 89 percent, staff time required to release reduced by 75 percent, and the resulting reduction by 97 percent of the financial cost of a release — are dramatic. These results largely arise from automation, but it would be interesting to do a study on the softer side of DevOps. How much is un-siloed thinking actually worth? It's probably unmeasurable, but it might be fun to try and put some metrics together.

Before we went live with DaVinci, we executed 47 releases, and automating saved us 17.5 man-days on just that one application, in a single week! Automating means the same team can do much more, because there's less firefighting, and infrastructure is far more manageable.

We're saving time with our VMware virtualized infrastructure, too. We can now spin up servers in minutes rather than hours, and the handoffs between the different infrastructure disciplines have been reduced (i.e., VM guys talking to the Active Directory team, the storage team, etc.).

In general, DevOps has meant big improvements in reliability and visibility, and those lead directly to improved productivity and speed of releases. With more reliability in your delivery process, you get a faster pace of change. If you understand that your environments are in an expected state, that helps, too; the questions about the state of dev, or of test, go away. DevOps helps solve that frustrating question "why does it work in systest but not UAT?"

It's hard to quantify just how much reliability increases the pace of change, but it's going to continue to improve. Now our servers are all configured the same, so the bottleneck moves to other places and we'll have to figure out how to get databases configured the same, then load balancers...and so it goes. Eventually, I'd love to configure everything with Puppet and define the entire infrastructure with Puppet. Then we'll have great traceability between environments, from the top of the app stack all the way down to the VMs and network devices. We're still some way from achieving that, but as we move to cloud — which is on the roadmap — our starting point will be to define as much as possible in Puppet.

Visibility has opened up a lot for us. We still have some silos, but getting everything defined in code means the line between what's application and what's infrastructure is starting to blur. We want to get to a point where an application guy has visibility into what the network is doing, so if there's a problem, he can say, "Okay, my app is fine; let's look at the network." If we're all using Puppet, we have a common language for defining and making change, so it no longer matters if you're talking about the application or the infrastructure. It doesn't matter if you are talking about Dev or Ops — it's all just DevOps (or rather, DevOpsBizTestThingy!).

In fact, it's gone further than that. We've got to a point of maturity where rolling out a deployment is no longer an IT task. Guys in user acceptance testing, or the systest guys, can push a button and deploy. The knowledge of how to deploy is much easier; it's no longer just one guy who knows the magic potion. Potentially, anyone can do a release. You still need to have your approval systems in place, but it's no longer the case that if the one guy who knows everything is sick, you can't release.

Tools and technologies at Hiscox

Hiscox runs Windows primarily, and our Puppet master runs on Linux. Our database servers are physical, and for the rest, we're running between 2,000 and 2,500 VMware virtual machines. We had people building VMs manually, but now we automate that with VMware orchestrator; that saved us 50 to 60 percent of the time it took to build a VM. Now we want to think about using Puppet to automate what goes on top of the operating system. For example, we could use Puppet to get antivirus software onto every machine.

- We use Puppet Enterprise for middleware configuration management plus UrbanCode Deploy, which is more app-centric. For us, Puppet is about converging state and about stability and alignment, keeping us maintained at a certain level. UrbanCode changes state.
- For continuous integration, we use both Jenkins and TeamCity. It depends on the user: The .NET guys like TeamCity, and the Java guys like Jenkins — it's a more natural fit for them.
- We use Subversion for version control, and the Microsoft guys use Team Foundation Server, but we're moving everyone over to Git (hopefully).
- We pull thousands of metrics and use Splunk to aggregate the different metrics from different tools, so we can infer relationships between things. For example, we try to see if we can relate story points from Jira to the number of errors that appear on our websites.

Next steps

DevOps is a journey, not a destination. The highest-profile teams get it first; then as they get onboard, the rest of IT will get the tools and start applying DevOps principles to their work.

We'll be utilizing public cloud soon, and that steps it all up a notch. One of the reasons we need to move to cloud is to spin up environments and applications quickly and with confidence. It's a challenge to get the full benefits of cloud, and to do that, we have to keep moving on this DevOps journey.

Learn more

DevOps can boost overall organizational performance, and depends strongly on IT leadership for success. Read about the state of DevOps today in the 2017 report:
puppet.com/devops-report

Get more tips for managing organizational change and measuring the impact of your DevOps initiative in "Get Started with DevOps: A Guide for IT Managers":
puppet.com/devops-guide-it

Top outcomes of using Puppet Enterprise

- Lorem ipsum dolor sit amet, consectetur adipiscing elit.
- Duis semper vitae ex vitae malesuada.
- Sed porta, nisi non rutrum iaculis, turpis eros.
- Efficitur mi, eget varius velit mi eget quam.