DevOps in Telecoms – is it possible?

A recent article¹ on CA takes a look at how DevOps is making strides into many industries to improve the speed in how we deploy our solutions in Agile development environments. What surprised me was that it’s the Telecoms industry that stands out as the biggest adopter of DevOps and seems most willing to extending their existing agile methodologies. CA says:

“The clear leader here is telecoms. They are almost two-times more likely to be using DevOps (68%) than the aggregate (39%). In fact, a total of 88% of telecoms either are using it or plan to, versus an aggregate of 66%. It is clear that the telecoms industry is one of the most competitive, and the pressures to continuously deliver new products and services are enormous. The benefits of DevOps in terms of accelerating the pace of delivery provides a pretty compelling reason for telecoms to move forward aggressively with DevOps.”

This seems accurate for ISP’s and mobile operators who deploy webRTC, web services, HTTP, can run in a browser or is easily virtualized in an OpenStack² container. DevOps comes from IT (as did Agile). It is used primarily by technology platforms in industries that converge with Internet platforms and web-portals.

---

But this pure IT stack, responsible to deliver content to the device or implement middle-ware, at every operator and ISP is only a subset of “telecoms”, and doesn't reflect the largest share of what the Telecoms industry “actually is”: Namely equipment providers building the hardware.

Apart from the bigger vendors such as NSN, Ericsson, Alcatel-Lucent & Huawei, etc., there are thousands of medium sized firms writing a lot of software that is deployed using Agile principles. They moved from iterative-waterfall to Agile years ago, and albeit their huge size (compared to IT teams) are no strangers to Continuous Integration, -and up to a point- even the Continuous Deployment of their software packages.

But unlike IT and Internet platforms, what they create isn't a virtual service that is deployed somewhere in the cloud and can be continuously patched. They deliver hardware that may cost millions to commission and is maintained over years with strict SLA's. So on a technical level, by using OpenStack, Puppet\(^3\), Chef\(^4\), Salt\(^5\) or other technologies, DevOps isn't going to do anything for the Telco guys.

When I first asked around about what DevOps actually was, and to learn how to improve processes in Telecoms or Automotive, the confusing answer by advocates was always:

"To understand DevOps you need to remember that it isn't a framework of tools, but a mindset".

Aha! This made me curious, because nobody even in IT could provide a proper definition other than: "It will make you faster by improving the communication between development and integration/deployment departments".

**Whether Telecoms, Automotive or another hardware intensive industry, anyone forced to take a conservative route, will be unlikely to embrace DevOps by-the-book, even if Agile processes are already in place.**

Those of you familiar with some of the funny images over at devopsreactions\(^6\), might get a good chuckle out of the idea of "DevOps in a Nuclear Power Plant", or deploying "live upgrades to Medical Surgery Equipment".

But don't dismiss it just yet. Its ideas (or "mindset"), can still make you quicker than your competition! Before we go all philosophical, let's look at how these bigger firms outside IT usually develop/deploy their products:

ACME Corp is a multi-billion dollar business and develops network components for mobile operators. One of their R&D projects currently keeps more than 1000 engineers busy developing away to make this world a better place.

---

3  http://puppetlabs.com/puppet/puppet-enterprise
4  http://www.getchef.com/solutions/devops/
5  http://www.saltstack.com/enterprise/
6  http://devopsreactions.tumblr.com/
In the last 10 years they have moved from "iterative" (Waterfall) development method to Agile. Their engineers now organize themselves as micro-teams which are assembled/disassembled in an ad-hoc manner working on various chunks of the product. Not everyone touches everything within the code-repository but they look at code & ownership in the context of its features and impacts on the system.

Competences are spread across the different geographic sites with a good mix and balance of their skills. Some teams are closer to the hardware others work higher on their stack. Strict quality guidelines ensure that whenever they submit a change to the software-repository, immediate automated feedback is given by their continuous integration (CI) system to inform when they’ve broken existing functionality. When that happens they either deliver a correction immediately or roll-back that change. This way the overall content always has a very recent working and testable version of the product or it's sub-components.

Once they delivered changes or new features this way, and provided nothing breaks, their code is "promoted" to downstream integration and test stages.

These guys (in other departments) then pick up some versions and integrate them further and run more complex tests. The integration guys have their own CI system and repository to version and store their test-cases, and then in turn promote whatever came from upstream after their tests passed, to the next teams.

You might find 4 or more of such test/integration instances, all organized by their own management layers. Eventually the developers code and final package reaches the real hardware where end-to-end system tests and inter-operability (IOT) tests can be conducted.

Everyone is happy! That is until faults come cascading back up from downstream, and analysis will eat up a lot of the resources.
Which software package version has the fault? How many branches have we already created in the meantime which now have the same faults? Where do we have to merge the corrections to? Who else will come back from another department reporting the same error? How can we ask developers to deal with all this overhead when they are meant to work on a tight deadline to churn out the next features in the sprint?

Fault Manager to the rescue!

But wait! Now we have an important customer trial (actually we always have one of those) which means we have many of these must-fix-immediately-super-critical-do-not-wait faults. And we need developers to correct it right now.

Yes NOW, ... even we don't know yet who the responsible culprits are, who've committed this crime.

And since us fault managers are overloaded enough as it is, we don't have time to analyse any of this. Only if we would have some dedicated people who could look at the logs and identify which developers must deliver the correction!

Pre-Analysis to the rescue!

And just in the nick of time, our hero managed to analyse the 10 GB of logs and identified the usual suspects, confirmed it wasn't a bug in the actual tests, and forwards the fault to the right guys, who then deliver the required correction tout suite.

Though this is clearly not the end of the story! As by now the fault had propagated into products which have already been frozen and therefore MUST NOT be corrected unless there is an official request from the customer.

Since the official delivery of the faulty code was made 3 weeks ago, this issue will raise it's ugly head a few more times in the coming months: And each time on a different branch. And each time popping up as a new bug in the bug-reporting-tool with a different set of trace files.
So clearly the fault-manager and the pre-analysis guy will be busier than Batman and Robin for years to come!

**What happened? Didn't agile make them faster?**

Having abolished their code ownership and implemented Agile internally within their team, ACME Corp gained tremendous speed over their competitors, churning out well tested changes and new features every few hours.

But by doing so another bottleneck had been made visible: The company had internally committed to Agile methods, but every department after becoming Agile still did it on their own terms, maintaining their status quo and continue to treat other departments like an external collaborator.

Back to their workflow: If you look closely at how the product delivery chain "cascades", often over 5 or more integration & test departments, you still see a waterfall! And that waterfall is one costly and resource hungry administrative nightmare, quickly eating away the developers time, as they have to constantly provide clarification to dedicated coordinators, whose main job consists of steering the communication about the faults into the correct channels. Frustrating.

**And this is where Agile stops and DevOps can help you pierce those remaining silos.**

What DevOps can bring to the table in industries that are "close to the metal", isn't a new tool to solve all your problems. And neither was Agile!  

Thinking in "DevOps terms" means **integrating your downstream test environments / operations bringing them closer to developers.**

Before you dismiss this, take a look at how your downstream integration and test departments are currently still reinventing the wheel! Some of them will not even use Agile principles or automation such as continuous integration or (god forbid) even version their tests.

In fact from their perspective not much has changed: They still get a baseline at regular intervals which is loaded to the test-environment and then they give feedback.

The further you move downstream the more you will find a lack of scripting know-how which comes natural in your upper layers. Integration might select only a few of these baselines randomly or "cherry-pick" whatever looks most promising.

So even your developers churn out features at maximum capacity, downstream continues to live isolated from the rest of the upstream departments.

---

And it gets worse the further you move downstream and away from developers:

- Each department inventing their own test systems because of their unique needs (some justified some not).
- Many writing overlapping test cases and re-test what was already verified. --> Not that double checking is a bad thing, but writing/maintaining the same test-code several times is a waste. My guess is that there will be a massive overlap, and engineers creating and solving similar problems in every department.

Approaching this with a DevOps "mindset" will help you cut down the technical barriers between these cascading departments, so that **automated deployments become possible**. Add API's that **make the whole delivery-chain transparent** from a tech pov - all the way to the target hardware.

The good news is that you don't have to start doing it all at once, as you did when initially migrating to Agile 10 years ago! Instead start with a gap-analysis conducted by either an internal architect -let's call this person a consolidation-engineer- and check every one of your test/integration departments to identify overlap and ways to automate your deployments API's/interfaces.

Once you have a clear picture, figure out how you can break your internal silos and improve intra-department communication. How would you do that? Simple: you already did this once when you introduced Agile in your development teams!

1. Now take the next step and have some of your developers spend one or two days a week in what was until now the next downstream department. And move some of these guys into development for few days a week.
2. You might want to think about incentives for doing so and reward people willing to work across these borders. Consider them as ambassadors in your company who pierce through your silos. Before you know it, boundaries will have become blurry and that silo is cracking! Natural cross-pollination of ideas will start happen - from bottom up!
3. Developers now see the effects of their code-changes within just a few hours and on the actual target hardware. This makes them a more relevant part of your big "organization-machine", and also helps them to identify themselves deeper with the product you sell.

If you have thoughts or ideas about how to make use of DevOps in industries that are close to the hardware please share with us! I'd also be interested on thoughts on how this could tie in or benefit existing DFMEA strategies.

---

This post was written by Joachim⁸. You can hire him, or one of his consultants, to help you improve your R&D processes, Agile-Development techniques, Continuous Deployment.

---

⁸ http://fr.linkedin.com/in/jbauernberger/