xmatters
AN EVERBRIDGE COMPANY

Seven Crazy Goals to Start You on Your ITSM Journey to DevOps

You won't believe this one weird trick to make service management agile

By Rob England

Introduction

2018 is the year for service management in DevOps. The echoes of “Ding! Dong! The wicked ITIL is dead!“ are just fading in the DevOps and Agile community. But the DevOps world is increasingly rediscovering the value of service management techniques, just as ITSM teams are adopting DevOps principles and practices.

DevOps experts like Gene Kim have consistently defended the need for ITSM in a DevOps world, and DevOps is coming around. When Google’s Site Reliability Engineering (SRE) deduced a metric for reliability based on first principles, and they came up with... Unplanned Downtime. (Everybody in ITSM rolls their eyes. Well, duh.)

And ITSM is accepting DevOps. The new ITIL Practitioner book and course by Axelos embrace Agile concepts, and there is a rumored rewrite of ITIL coming. There is also a rising awareness in ITSM of the Cynefin framework and how the world can’t all be standardized.

At last, ITSM is coming in from the cold. It‘s important to deploy products faster and better. It‘s just as important to deliver its value over time. Here is a roadmap for one way to encourage that crossover in your ITSM organization.

Set a crazy goal. It's OK to fail

Create a culture where people feel safe to fail, where they understand that experimentation is essential in order to move forward. Not a major catastrophic failure, of course. It means to fail fast, fail small, fail early, fail often—to gain quick feedback and change course if necessary.

One way to experiment is to set stretch goals that might seem a little crazy at first. Tell your service management friends that DevOps is a new world where they can be adventurous and curious, with the right environment and controls. Startle people out of their day-to-day life and make them cry, "Impossible!" And answer back, "Unless….?"

Make these goals SMART: specific, measurable, achievable, relevant, and time-bound. Write them as a user story with defined acceptance criteria and a milestone date.

Stretch goals rally and energize early adopters. As success draws closer, even skeptics want to get onboard.

Here are seven suggested crazy goals:

1 Make performance testing a parallel control instead of a serial one

Set up a parallel performance environment so that developers can test code at will instead of waiting for a single monolithic performance test at the end of their work when any feedback is too late.

Create automation so they can submit packages of code for execution and get immediate feedback on performance metrics. Extra points if you give them comparative metrics from the previous run. This encourages developers to iteratively tune the performance of code instead of taking a blind guess.

Organizations have set this up with astonishing speed even in complex legacy environments, with automated tools for developers to submit tests and see results. Integrate testing and monitoring tools to enable the flow of information.

The role of test engineers then shifts from working through a backlog of testing requests to building and maintaining automation and watching the results to identify teams who need special help or issues with infrastructure performance.

2 Collect data to support problem management

Some documentation is only "documentation theatre." Nobody ever uses it again. Find out if and when it is ever opened.

Often it's just a glorified checklist of items to be completed at a critical stage gate. Why not just use an actual checklist? Checklists are the low-hanging fruit in terms of reducing levels of ceremony and optimizing controls.

Other documentation is indeed essential but you should look for a way to store the information in the machine instead of transcribing it manually. In other words, your automated systems of source control or migration or testing include all the necessary information. For example, do we really need release notes when all the information is in Jira? If we move to test-driven design, can we eliminate many written requirements and use the test scripts themselves?

One major target for reducing documentation is the CMDB. In ITSM parlance, a CMDB is a static repository to which information needs to be manually and automatically written. Its ROI is often debatable. In a DevOps context, the configuration database becomes the active definition of the environments, so data flows from it, not to it, and it is by definition correct. It is used to drive our system, not to record it after the fact.

3 Kill a CAB (Change Advisory Board)

Some organizations have multiple CABs at the technical, business, and operational levels. Many CABs can only be described as dysfunctional: reading out a litany of 60 changes to 30 bored disengaged people. If it is just "theatre" to reassure management, if the very idea of going to a CAB makes everybody lose the will to live, then get rid of the CAB. Find sufficient other ways to ensure that risk is managed so that the bizarre ritual is no longer required. Identify what value the CAB delivers, if any.

- Identifying risk. Apply the Shift Left principle and find better ways to identify risks earlier in the process. Turn to a trusted peer instead of 30 strangers and a five-minute discussion. It is pure theatre.
- Assuring quality. Again, Shift Left to build quality, security, and compliance into the process. For example, Continuous Testing based on extensive test automation eventually eliminates many external assurance functions
- Communication mechanism. Look at tools such as social media, and especially the ability to subscribe to configuration items through ChatOps
- Approval. This is dysfunctional at the core: the "A" is for "Advisory." Usually, it comes to this because the team creating the change refuses accountability, so the CAB provides a way to distribute accountability

4 Enable developers to create a test environment at will, cloned from production

If it takes weeks to handcraft a test environment, it is a major bottleneck to development. Again, it is low-hanging fruit to provide automated request mechanisms for developers to be provisioned immediately with the server, database, storage, and network resources they need to get started.

If a test environment is not identical to production, then what is even the point of testing in it? The primary purpose of testing is not to find defects: the purpose of testing is to reassure us that it will work in production. Dynamically create copies of production on demand for testing. Tools exist to obfuscate data and to create virtual images of production that do not require massive amounts of storage.

For many organizations, their initial entry point to the cloud is to provide transient resources for test environments in order to accommodate on-demand requests for large production-like systems. When they experience the low cost, responsiveness, and integrity of those systems, then the use of the cloud starts to spill into other practices such as development and eventually production. It is the thin end of the wedge.

5 Automate change approval of a product

For every person who needs to approve the change, ask what controls she would need to see in order to no longer approve, then find a way to automate those or render them obsolete.

Any change manager should be able to provide such a checklist, although it is often an interesting exercise for them to do it.

Likewise, it is equally enlightening to ask people why they are approving. It is not unusual to find that they never wanted to. They are happy to be relieved of accountability which they never felt they owned.

Shift Left as many approvals as possible. Peer approval of change is one of the strongest predictors of IT performance. Similarly, business approval of change should happen before it is done, not after. If we agreed to make the change at the start of a sprint, we shouldn't need approval again at the end. The need for final business approval of change stems from the days when we went away for a year or two.

For the remaining approvals. we try to automate the rule base for approval criteria. If we can't automate the approvals, we can at least semi-automate them by providing single-click emails or similar messages, or review queues in ticketing tools.

Organizations talk about "virtual CABs" which never need to actually meet.

6 Set a product's deployment window for 2 pm on a weekday

Why do we deploy at 4 a.m. on a Sunday morning when it would be so much safer to deploy during work hours when all the essential people are there watching the consoles? Impossible unless... What would it take to be able to deploy on a weekday? What level of automation and quality would be necessary?

You would need to find a way to decouple deployment from release so that code could be live but not yet provisioned to users. Then you can deploy frequently and incrementally while the business is free to pull functionality in a business release on any scale or cadence they desire.

It doesn't mean you have to be Netflix, but you don't have to white-knuckle it every time either. Deployments should be so automated, tested, and reliable that they are a mundane, boring event. This is classic DevOps Continuous Delivery practice.

7 Automate provisioning of users to a service

In order to have such decoupling between deployment and release, you need to make it as easy as possible to provision users to a new deployment.

The business should be able to release deployed code to groups of users at will, i.e. they pull, we don't push.

To do this, provisioning users to a release should be automated in such a way that we can push and they can pull. The ideal approach is user profile configuration parameters, but we can achieve limited versions of the same result with feature flags, blue/green environments, and even clunky staged releases from pre-production to production. Good provisioning enables canary releases, beta testing, A/B testing, releasing by timezone or business unit, and incrementally increasing the user numbers.

If you don't think these goals are crazy, you're a unicorn. Happy flying. We horses salute you. For the rest of us, these goals should be sufficiently crazy to get the attention of everybody and to act as proof points of the effectiveness of DevOps if and when we succeed at them.

Understand the implications

Unpack the implications of the goal: “Impossible, unless…?”. Work your way back from it until you arrive at some requirements which you can begin to work on now.

In general, in order to Shift Left, you need to **improve the rest of the system** until the control looks ridiculous/ superfluous because it is no longer necessary. In other words, we should build in earlier practices, tools, and controls to improve quality to the point where the control is no longer performing any useful functions, is no longer “trapping” poor quality or preventing risks. For example, look at Continuous Integration as a development discipline, or movements such as DevSecOps and Rugged Software.

Providing new capabilities such as copies of production often raises strange political objections. For example, “developers are not allowed to see production data for testing,” and yet at least one has a production sign-on for debugging purposes. They all signed some form of confidentiality agreement when they joined the organization, so treat them like they are on your side and get out of the way: IT must move from “default access none” to “default access all.”

Just do it

Once we identify some immediate requirements, and once we put sufficient controls in place to protect the integrity of the systems, then we must avoid analysis paralysis. The IT obsession with perfection creates the “define once, execute perfectly” fallacy, which stems from our belief that we are creating simple systems. We aren’t; we are creating complex systems. The only way to move forward in a complex system is to **iterate, increment, explore, and experiment**. You must generate observational data to understand what to do next.

And so we embark on continual iterative experimentation and exploration as we incrementally improve our way to our initial requirements and eventually our crazy goal. Plan only enough to start, then get going because your plans will always change. Planning is essential but plans are expendable.

Success factors

DevOps makes you better and faster, but nobody said it was easier.

Here are the success factors to enable you to do crazy things. As prerequisites, it would be useful to have:

- Executive support or at least curiosity
- Management understanding of Agile principles at all levels up to ownership of the artifacts you are changing (so they don't think you are mad)
- An appetite for change amongst those doing the work
- A culture that empowers those doing the work to design the work
- Governors who are beginning to understand the different attitudes to risk in an Agile enterprise, or who at least accept that there are alternate ways to deliver the same risk outcomes
- An Agile approach to changing practices. Agile Service Management is a thing

Perhaps the most critical success factor is expectation setting:

- Zero risk means zero experiment, and zero experiment means zero innovation. Everyone in the organization up to the highest levels must understand that if we are ever to move forward we will occasionally go wrong, and that this is the cost of that movement. After all, it is demonstrably true that locking everything down does not result in zero errors. This is the fallacy of robust systems
- You may never reach these crazy goals you set yourself. The agile way of working is to iterate towards a goal, deploying incremental success along the way, sometimes abandoning the idea completely or pivoting to an alternative idea
- The real value is not what you achieve. The value is in the journey of attempting it, and what you learn about new ways of working: new attitudes to working which embrace experimentation, failure, and curiosity. You may not get where you were going, but you will be in a better place than you are today. If you can say that it will be better in a year—that you will achieve better results and have a better time doing it—then you know you work for a learning organization with a culture of continual improvement

Technical Notes

You may have noticed an emphasis on automation. Automate testing. Automate sharing data. Automate cloning production environments.

You may have also noticed an emphasis on **Shift Left**. Approve processes earlier. Test earlier. Fix problems earlier.

Finally, you may have noticed an emphasis on **sharing data**. Share testing data. Share process status. Share monitoring Alerts.

These are the linchpins of modern IT operations. They require business alignment first and foremost, of course, and then they require deep system integrations to make data available when and where people need it. From monitoring tools to service desk systems, and from ChatOps tools to issue management solutions, collaboration relies on data sharing as much as it does on traditional forms of communication.

About the author

Rob England B.Sc., MIITP, CITP is an independent IT management consultant, trainer, and commentator based in Wellington, New Zealand. Rob is an internationally-recognized thought leader in IT Service Management (ITSM) and a published author of seven books and many articles, best known for his controversial blog and alter-ego, the IT Skeptic. He speaks regularly at international conferences. In 2005 Rob founded his company, Two Hills, with the motto "Sensible Business Practices."

Read Rob's books: http://www.twohills.co.nz/books

Visit Rob's blog: http://www.itskeptic.org/

About xMatters

xMatters is a service reliability platform that helps DevOps, SREs, and operations teams rapidly deliver products at scale by automating workflows and ensuring infrastructure and applications are always working. The xMatters code-free workflow builder, adaptive approach to incident management, and real-time performance analytics all support a single goal: deliver customer happiness. Over 2.5 million users trust xMatters daily at global companies and innovative challengers including BMC Software, Credit Suisse, Danske Bank, DXC Technology, Experian, NVIDIA, ViaSat and Vodafone. xMatters is headquartered in San Ramon, California and has offices worldwide.

xmatters.com