From a Controlled Chaos to Well-oiled Machine: Agile and Devops Complement Each other

Author: Rajasree Talla
Program Manager, New Relic Inc

Abstract

Current thinking in the software industry demands that companies be ready to release a product whenever our customer is ready - not when the company is ready. It's also important to build and deliver software that's designed from the customer's viewpoint. It's no longer about, "Here's a cool feature that makes us look like awesome coders." It's about releasing increments that allow us to deliver features or capabilities that customers find valuable. Incorporating DevOps practices is desirable to deliver products faster, so you can get feedback faster, get to revenue faster, and compete effectively. Hearing from customers and going through transition of practicing DevOps, informed it's not easy and there is no prescribed way. This journey uncovered how closely being agile and practicing DevOps need to work together in order to achieve the true value.

This paper explains a company's journey to become a well-oiled machine -- how internal processes, tooling improvements, incorporating feedback loops supported with agile mindset helped us achieve this transformation providing incredible value to us as an organization and also to end-users. Key focus areas include the approach taken with these tracks:

- Adopting continuous integration, delivery & deployment
- Iterative development and agile mindset
- Incorporating feedback loops: build, measure and learn
- Building the muscle of continuous improvement

Biography

Rajasree Talla has about fifteen years of experience in software industry and has been part of agile transformations and practicing DevOps in various companies. With a passion in customers experience, she is keen in ensuring end user workflow remain the focal point of organizational transformations. Strong advocate of lean methodologies, being agile and continuous improvement. Trained as SAFe Agilist, Scrum master, Project management professional (PMP). She has masters in Engineering and technology management from Portland State University. She is co-author & editor of Planning and Roadmapping Technological Innovations: Cases and Tools . Enjoys speaking about DevOps & agile transformations and is a speaker at DevOpsdays.

Linkedin: https://www.linkedin.com/in/rajasree-talla/ Twitter: @rajasree_talla

Table of Contents

1. Practicing DevOps

1.1 Why DevOps?

Customers expect companies to deliver software faster with a predictable cadence. Organizations expect to get feedback faster, revenue faster and also compete effectively. It is important to try to reduce the gap between development and feedback. This expectation doesn't differ much with various industry sectors like science, health, technology, consumer applications. One common question is 'How to enable organizations to be high performing?' Most of the companies try to align teams to optimize performance.

"In high-performing organizations, everyone within the team shares a common goal—quality, availability, and security aren't the responsibility of individual departments, but are a part of everyone's job, every day." (Kim, Debios, et al. 2016)

Focusing on a common goal and keeping customer experience intact is important. Removing the handoffs and barriers within the organization plays a key role. Not just the product development team, operations team is also part of this equation. That is where DevOps practices play a prominent role.

DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support. (Mueller 2018)

DevOps 'Infinity loop'

(Kulshrestha 2017)

Looking at what are the pieces in software lifecycle workflow in a simplified way in the image above. Communications and collaboration are keys to practice DevOps. Having feedback loops back to development from operations will have this effect of this infinity loop. Planning for a project developing code, testing & release cannot be is isolated from operations - deploying the code, monitoring. Feedback from monitoring the experience should flow back to planning. Tools & internal process changes are required to optimize this flow. Many tools are available in market to optimize & automate this workflow. The benefit of having this working will lead us to not only high performing teams working on a common goal, but also help us with a consistent user experience seeking feedback along the way.

1.2 State of DevOps report

Every year companies analyze what's new in DevOps practices to help them practice DevOps. We chose to review Puppet State of DevOps report 2017 by Puppet to understand challenges of practicing Devops. This is generated in collaboration with DORA (DevOps research and assessment). This analysis is from survey of responses from various IT industries. It's important to understand the current trend while adopting DevOps practices at an organization. This will help you serve as a resource for best practices

and customize to your company needs. Below were key findings with the 2017 DevOps report. (Forsgren, et al. 2017)

- Transformational leaders share five common characteristics that significantly shape an organization's culture and practices, leading to high performance
- High-performing teams continue to achieve both faster throughput and better stability.
- Automation is a huge boon to organizations.
- DevOps applies to all organizations
- Loosely coupled architectures and teams are the strongest predictor of continuous delivery.
- Lean product management drives higher organizational performance

1.2.1 Why DevOps at Puppet?

Puppet is one of the leading configuration management tools used to practice DevOps.While Puppet's portfolio was evolving, the need to respond to customer's feedback also increased. Puppet team was making a significant effort to cut down on release cycles and react to learnings and feedback from field as quickly as possible. Being part of many DevOps transformations, Puppet understands the benefit of shifting the organization to adopt DevOps practices. In this case, 'learning & feedback', 'design for customer needs', ' focus on continuous improvement' , 'reduce time to market' served as key interests for this shift (Kersten 2017). With the support of our leadership Puppet was able to succeed through this transformation (Gazitt 2017)

2. Challenge in hand

2.1 Context - Product

It is important to know the baseline before jumping into how we shift to adopt new practices. At Puppet we were looking at adopting DevOps practices in entire product and engineering organization. Major effort was in shifting the flag ship product "Puppet Enterprise". For context on the problem, this product is a commercial on premise offering of Puppet. Major releases of Puppet Enterprise product are scheduled to be generally available every six months. Let's discuss about some pain points before the transformation.

2.2 Unpredictable releases

Releases were always planned with a fixed scope in hand. Scope here is the features we need release with attempt to detail and plan. There was some predictability with respect to the date of the release. New product iterations were released every six months. As a program manager of Puppet Enterprise, I always experienced a tension in the room about feature completion by this release time. Adjusting the release timeline according to the feature completion was the only solution. This approach led to release date and scope time adjustments at the very end of the release cycle. Not a great start to ideal go to market or customers experience.

2.3 Minimal feedback at development

Puppet already had a few mechanisms to capture feedback at early stages of a feature during discovery, definition, and early design. Technical advisory boards, market research are used during the discovery, definition of the customer problem & Puppet's opportunity. Usability research called test pilots is used during later part of definition, early design where users can provide feedback on prototypes. Not all features were generating prototypes. Once the feature is in development until its completion there was no feedback incorporated. This created a scenario where we had to sometimes to do a new patch release immediately to incorporate feedback and it was not the ideal customer experience.

2.4 Long hardening cycle - 6 to 8 weeks

Hardening cycle was about eight weeks. Hardening here includes integration, performance, exploratory testing. It also includes time taken to get the release bits ready and stages. Risk was always high as we integrate all components work in the first couple of weeks of those eight weeks. There was at least two weeks spent on integrating all the component's code to the main release branch (Johnson 2017) (Talla 2017). Doesn't make it effective because most of the difficult tasks were pushed to end of the release cycle.

2.5 Water -Scrum -fall: Handoffs

Puppet teams delivering this product were organized as scrum teams. They were practicing usual scrum rituals (standups, sprint planning, backlog grooming) and using tools to accommodate the same. Only developers were working from one backlog. Teams were focused on development, testing, documentation tasks separately. Centralized design, quality assurance and documentation team supported the development team (Gazitt 2017). Deliverable at end of a two week sprint was not a value add for user. It was an engineering task that will not make users realize value add directly unless the entire project is complete.

2.6 Continuous integration system always red

Lot of great work went in to making sure existing continuous integration system automated and kept was up to date. These tests were acceptance level tests covering different components. Only throwback was that, it was failing most of the times with breaking code (Johnson 2017). Sometimes with transients that are intermittent failures which cannot be reproduced easily. Bringing it to complete green was an impossible task even at end of release cycle. We had to disable few tests and ignore transients to ensure release happens on time. Basically this system failed to serve the purpose. It helped with finding the problems just not at the right time.

2.7 Code in production only after launch

Puppet's IT operations team uses Puppet to configure, deploy and maintain systems. Not every company has internal customers. They could be a great source of feedback. Pushing through most of the testing to end of the release cycle did not help. IT operations team were not able to install the builds before launch. It was a missed opportunity, but also reflects the quality of builds until right before launch (Talla 2017) (Gazitt 2017).

3. Bright new world!

There were lot of challenges in hand to shift this system at Puppet. Instead of focusing on efficiency, team started to look at what's being effective means at Puppet (Talla 2017). It took about eighteen months to see the complete benefit of adopting the DevOps practices. Puppet Enterprise now delivers a release every sprint (2 weeks) to customers as beta builds. Explained next are the key shifts noticed.

3.1 Continuous Integration green

Getting the continuous integration system to all tests passing were the first tasks. We already had a system and making it work is all we had to do. This required pushing integration testing to further down in the workflow. Code is integrated as soon as it written. Thus reduced the cycle time between developer

check in and integration into the main branch (Johnson 2017).

3.2 Continuous delivery & deployment

Releases are cut every weeks at the same time and day. These releases are installed by site reliability engineering team in to operations staging & production on day# 0. Code was actually in production every two weeks (Talla 2017). Beta program got the builds in to customer hands every two weeks

3.3 Minimum hardening for GA- less than a week

There is no separate time in the release cycle for integration. We integrate the code the mainline, the final destination as we go. Hardening is used for release preparation, allowing our go-to-market team to get things ready to launch day (Talla 2017).

3.4 Small batches of work

This required shift to iterative development. Deliverable at end of sprint are small stories with user value. Slicing of the feature in to such small batched of work became crucial to see value out of this framework. Started embracing agile mind set all along (Gazitt 2017).

3.5 Feedback in entire release cycle

Customers or end users using this product via Puppet enterprise beta program every two weeks. Did not have to wait until the end of the release cycle. We are able incorporate feedback cycles into design, development & early delivery (Talla 2017).

While this transition wasn't easy, there was definitely no secret sauce or standard recipe to adopt DevOps practices. Every organization has unique needs and its own way to make this work.

4. Approach

We were challenged by our leadership to actually adopt the practices recommended by Puppet preaches and promotes to customers on DevOps. With a small group of cross functional team including myself as a program manager were trying to determine how and where should we start this transition.

4.1 Interviews & Findings

It was important to understand the problems from the people that were facing them. We had to understand the pain points. So we started interviewing various disciplines across the organization trying to understand where to focus, their perception of DevOps and also trying to find the low hanging fruits.

4.2 Tracks & approach

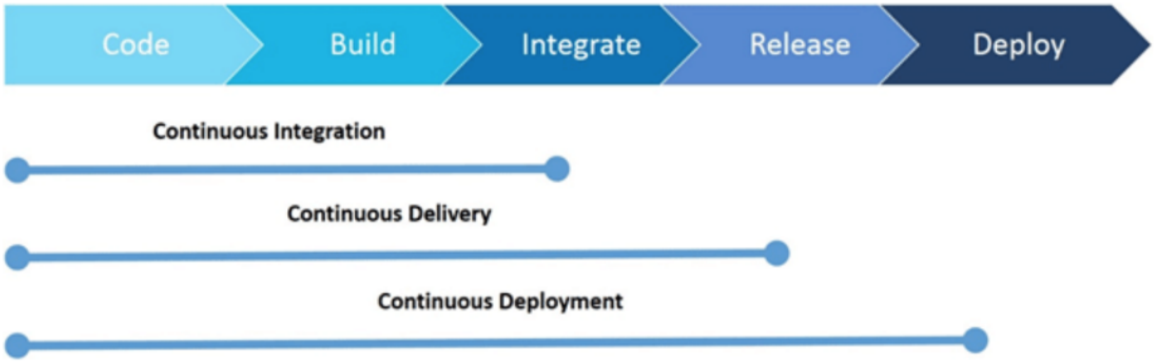
Many good points were brought up and the cross functional group started sifting through the conversations, grouping the pain points. We separated these into three almost parallel tracks for execution

- Adopting continuous integration, delivery & deployment
- Iterative development and agile mindset

- Incorporating feedback loops: build, measure and learn
- Building the muscle of continuous improvement

4.2 Adopting continuous integration, delivery, deployment

To make other tracks work, it was important to get the tooling and automation in place for continuous integration, delivery & deployment work. Here is a view of how these three will impact the release cycle



(Tiwari 2018)

4.2.1 Continuous Integration (CI)

CI system was already in working. We had to make shifts to advance the CI system and internal processes. All developers were asked to promote code early and often. Depending on the component most of the promotions were automated. This reduced the feedback time for developer check-ins. automated testing became a requirement at component level that mimics the integrated environment before merging to mainline. Getting CI green (passing all integration level tests) became everyone's responsibility instead of a single team owning the pipeline (Johnson 2017). This became the measure for quality of code.

4.2.2 Continuous Delivery (CD)

Implementing continuous delivery was major piece to get this adoption work. We had to look at the entire release process and make it as lean as possible. We reduced number of milestones and also heavy water fall stylish milestones. We tag a release every alternate Wednesday at end of sprint cycle. Rule of thumb was, if CI is red due to a code change, revert the change and there is always next release cycle to fix it. Don't hold up the release for that. Also implemented release branching as part of continuous improvement. Releases became predictable with this shift. The quality of builds being delivered as part of continuous delivery are at quality of a general availability build (Talla 2017)

4.2.3 Continuous Deployment

Started using IT operations team as our customer #0. As soon as the release is out this build is deployed in staging environment. Depending on the deployment and basic smoke testing build is pushed production on the day of the release. Every sprint a new build tagged as release is also available on the Beta website for the customer. Customers who signed the beta agreement were able to deploy the build on day#0 of the release. Production deployment was still a manual step. Continuous deployment forced the builds to have the quality of generally available release. Hardening cycle was completed reduced from six weeks to a week.

4.3 Iterative Development: Being agile

This track was the most difficult as it needed a mindset shift on many areas of day to day work Inheriting the agile principles improved the quality of work. Teams were organized into autonomous self-sustaining teams (Broza, The Agile Mind-Set: Making Agile Processes Work 2015).

User design, testing & documentation teams started working from one backlog towards same goal. Centralized team structure for testing, design, documentation was dismantled. All the work items were present in scrum board.

Starting from how we define work to how execute and release software had to change. Product managers and UX designers started breaking the work into stories with user value. With continuous delivery & deployment it was possible to iterate (producing small batches of user value and adjust deliverables with feedback) on the feature before the big splash generally available release. Every team has unique challenges while decomposing a feature into increments of value. Keeping the feature value proposition intact while slicing the user stories was important.

Without disrupting existing work, selected couple of teams to test this shift. Focused on concepts that are biggest bang for the buck. We were already doing sprints and scrum rituals. Key changes at team level include

- Definition of done & acceptance criteria per story
- Getting use interface work into sprints
- Sprint demos & review stories with product owner, field engineers, usability team were significant
- Some teams had a definition of ready before starting development on a story
- Documentation and release notes developed incrementally as we develop user stories.

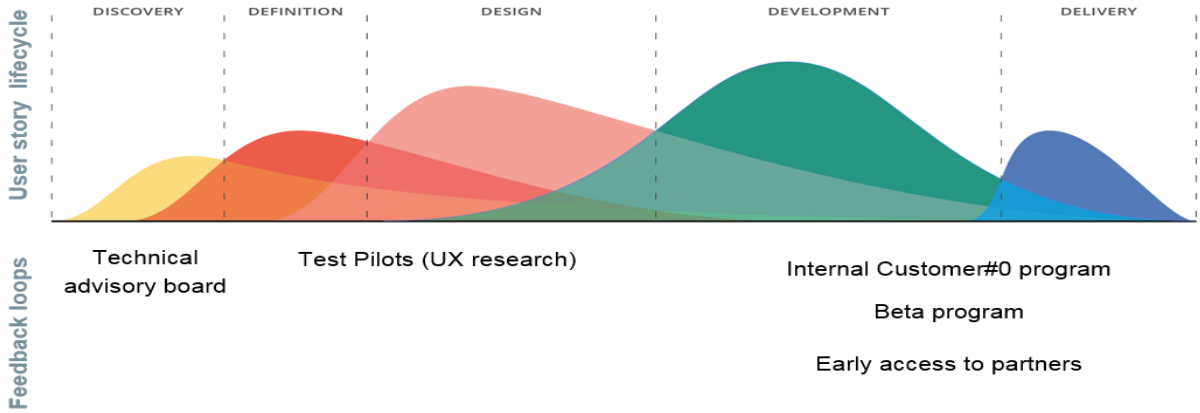
These changes shifted team's mindset. Testing is not just one person's responsibility. Team tested each other's story at component level manual testing. These new tests generated via test planning were automated for future use. It suddenly became a shared game instead of throwing over the wall. These best practices were shared among the teams. (Broza, The Agile Mind-Set: Making Agile Processes Work 2015).

We are able to get flexibility with the scope focusing on objective of learning and feedback After a complete release cycle of six months it was clear that leadership and parts of organization that enable go to market are shifting towards agile mindset and iterative approach (Broza, The Human Side of Agile - How to Help Your Team Deliver 2012).

4.4 Feedback loops: Build, measure & learn

The most important benefit of iterative development is incorporating feedback into product development. This is incorporating feedback loops in a smallest batch of work we can produce. Below is how feedback loops were incorporated in to feature lifecycle

- Discovery of the feature technical advisory board and market research
- Definition & early design : Test pilots conducted by user interface designers
- Development : Customer #0 , beta, early access to partners
- After delivery measuring success, feature adoption and capturing feedback for future iterations.



In this process developers understood that - product teams don't really need the entire feature, perfectly finished, to get customer validation feedback (Kim, Behr and Spafford, The Phoenix Project 2013). Sharing increments of value generated curiosity in our customers, and prompted them to provide helpful feedback, which made it easier for teams to develop software that was most useful to customers.

4.5 Continuous improvement

At end of every sprint teams started doing retrospectives at team level & release level (Broza, The Agile Mind-Set: Making Agile Processes Work 2015). It is not possible to get everything working with guided set of principles for the first time. Internal process changes and molding the adoption to success was only possible with the ideas that came up from the retrospectives. Also fail fast mechanism helped us get to the releases every sprint quickly. This transformation of practicing DevOps was done incrementally and it is still evolving as a well-oiled machine. Building the muscle of continuous improvement is key for any organizations success.

5. Learnings & Conclusion

Many interesting facts were inferred from this transformation of a controlled chaos to a well-oiled machine. As an organization our prominent key learnings include -

- Understand how to be effective before being efficient (Broza, The Human Side of Agile - How to Help Your Team Deliver 2012). This is important while developing any internal process or trying to get to high performing teams
- Business deadlines are unavoidable. Go to market activities are important for a successful product launch and get revenue. Agility may not work in all cases.
- User story definition to delivery in a single sprint is challenging. Depending on the feature this challenge varies. This is an art that can gained with experience.
- Tooling & process is not enough for a transformation to be successful - being Agile is actually a shift in mindset.
- Getting a cross-functional team together doesn't mean agile. Completing a story is shared effort is important to get the value of it.
- Shift to customer lens by incorporating feedback loops helps the organization including developers gain confidence in the code.

In a nutshell, streamlining internal processes support the transformation. Being agile is a mindset shift & practicing DevOps is a cultural shift. It is proven being agile & practicing DevOps complement each other.

References

Broza, Gil. 2015. *The Agile Mind-Set: Making Agile Processes Work.* 3P Vantage Media.

—. 2012. *The Human Side of Agile - How to Help Your Team Deliver.* 3P Vantage Media.

Forsgren, Dr. Nicole, Gene Kim, Jez Humble, Alanna Brown, and Nigel Kersten. 2017. *2017 State of DevOps Report.* Accessed July 23, 2018. https://puppet.com/resources/whitepaper/state-of-devops-report.

Gazitt, Omri. 2017. *Practicing DevOps at Puppet.* 11 December. Accessed July 23, 2018. https://puppet.com/blog/practicing-devops-puppet.

Johnson, AJ. 2017. *How our approach to code integration led to shared responsibility & CD.* 24 October. Accessed July 23, 2018. https://puppet.com/blog/how-our-approach-code-integration-led-shared-responsibility-cd.

Kersten, Nigel. 2017. *What is DevOps.* 30 January. Accessed July 23, 2018. https://puppet.com/blog/what-is-devops.

Kim, Gene, Kevin Behr, and George Spafford. 2013. *The Phoenix Project.* IT Revolution Press.

Kim, Gene, Patrick Debios, John Willis, and Jez Humble. 2016. *The Devops Handbook : How to create World class agility, reliability, security in technology organizations.* IT Revolution Press .

Kulshrestha, Saurabh. 2017. "How do I learn Devops." Edureka. *How do I learn Devops.* Accessed 2018. https://www.quora.com/How-do-I-learn-DevOps.

Mueller, Ernest. 2018. *DevOps Foundations: Lean and Agile.* 23 July. https://theagileadmin.com/tag/devops/.

Talla, Rajasree. 2017. *Continuous delivery at Puppet: from controlled chaos to a well-oiled machine.* 14 September. Accessed July 23, 2018. https://puppet.com/blog/continuous-delivery-puppet-controlled-chaos-to-well-oiled-machine.

Tiwari, Ajay. 2018. "Continuous Integration Vs Continuous Delivery Vs Continuous Deployment." Saviant Integration solutions. *Continuous Integration Vs Continuous Delivery Vs Continuous Deployment.* India. Accessed 2018. http://www.saviantconsulting.com/blog/difference-between-continuous-integration-continuous-delivery-and-continuous-deployment.aspx .