What created DevOps, what is it and where is it going?

Eyrún Thrastardóttir

May 2020

1 Introduction

Software development is still a relatively young profession, taking off properly in the mid 19th century from the written theory around the turn of the century all the way to practical applications. By feeding instructions to machines the first computers were created and the instructions were their first programs. As a relatively young field it is still finding its footing between engineering, applied sciences and pure science. But the benefits of computers in most if not all fields can not be denied and it has been instrumental in driving the digital revolution that we now live in (Shallit, 1995).

For that reason a lot of people work as developers to create applications for practical usage or entertainment and for a while the way these developers worked mostly mirrored traditional engineering design. But the software development process is very collaborative by nature. People need to work together to get a project from the initial ideation stages all the way to a finished product that is deployed and maintained, with improvements or additional features added as it grows. The traditional engineering design is not the best framework to allow for this sort of collaboration. This has led to different methodologies emerging for software development. Although not much time has passed since the early computers there have already been shifts in how software is approached and one of the most significant one is a paradigm shift from traditional software engineering to a more agile collaborative environment.

In this article we will talk about what DevOps is, where it came from, what makes a good DevOps team and eventually pondering what the future of software development processes could look like.

2 Background

A paradigm shift was defined by Thomas Kuhn in the sixties as a way to explain scientific revolutions, or when a set of scientific practices undergo a change and new practices become prevalent (Bird, 2018). Software design has been no

exception when it comes to developing its own set of applied practices. Traditional software development is rational by nature. Its processes are split up into linear sequential phases where each step is finished before embarking on the next one. A classic example of this is the waterfall model that is the cornerstone of traditional development. But Software development is a creative process and is knowledge based so methods that suit other industries such as manufacturing or construction, where the waterfall model has its origins, might not be as well suited for software. There was a call for a change, a more flexible and iterative methodology that eventually led to the agile methodologies (Awad, 2005). Agile practices have eventually become the industry standard. It is a shift in paradigms from the traditional rational models to the more empirical human centred agile models and this shift certainly resembles a paradigm shift for applied sciences (Ralph, 2018).

A traditional software development team follows a linear role based approach where labour is divided between groups defined by their roles. These roles include the programmers, business analysts, operations, architects, testers, designers. And the labour will be defined into sequential phases, where each phase must be completed before the next one begins in incremental and iterative steps (Chau, 2004).

Teams will work on different parts of the process and often enter at different stages of development. That means when one stage of the development has been finished it is handed over to the team responsible for the next steps in the process. This fits in well with the sequential nature of the waterfall model. It can lead to a long communication channel where some roles are in direct contact but others are not, which in turn leaves a lot for room for miscommunication to happen. Product cycle would be long and with a few production releases with some defined interval. Shifting to agile means that teams work collaboratively and iteratively through all phases of a project. The process is more empirical as opposed to rational. Teams are cross-functional, self-organising and adaptive to changes in requirements throughout the process. They experiment by developing quickly and getting early feedback.

Figure 1: Development process

However, this is not necessarily true for the whole project development cycle as programmers would be separated from those who maintained and deployed the code, they would be a part of two different organisations.This means that they could have entirely different objectives and that's where the DevOps methodology comes in. Let's dive a little deeper into what that is (Wikipedia contributors, 2019).

3 What are developers and operations?

A software development team, or the programmers, is the team that is responsible for delivering code. They implement features decided upon in the requirements gathering and design phase usually using agile methodologies with various involvement in the prior stages depending on the framework and team culture. As their goal is to deliver software, they focus on writing code and delivering fast for agility and flexibility. Using agile frameworks they should handle changing requirements and additional features coming in during the development phase. At the end of the development cycle they will hand over the code to the next phase of the project, basically signing it off and remaining uninvolved in the rest of the process. Therefore they are partly oblivious to how the service will be run, deployed and maintained after handover only being involved when they are needed to fix particular functions or add new features.

An operations team is the team that is responsible for the operational management of a system. That includes getting the system up and running into production, deploying new releases, maintaining the system, responding to tickets from external and internal stakeholders and fixing bugs. They will be responsible for incident management and often need to be on-call to quickly respond to system outages or broken features, relaying tickets back to the development team only when feature developers are needed to fix an issue. In simple terms, the operations team will keep the lights on and fight the fires after the development team has left the building. Operations will more often than not also be in direct contact with the systems users, building a wall between the development team and the users themselves, so the team will see direct value in keeping customers happy. They are also trained in infrastructure services and depending on the situation could even be the ones to maintain and monitor servers.

4 DevOps

In the 2000s a shift was happening in the development cycle. Things were shifting left, the operations side was moving closer to the development side. The previously thick wall separating Dev from Ops was breaking down as the need for even more agility became necessary and the empirical thinking with small incremental and frequent changes needed to be supported by the operations side as well with progressive delivery. Separation between the two teams could result in infrequent production releases and long cycles of getting code into production whilst the teams align. This could mean maintenance windows that could result in down time of a service or the possibility of something breaking when going into production. DevOps combines the processes of the two teams as they work collaboratively to automate and integrate their processes (Buchanan, 2020).

Even if the team fully follows an agile workflow in the planning and development phase of a project, it can still be stalled by trying to get features into production via the operations team, lessening the agile effect of quick feature delivery to customers.

That brings us to DevOps, which is simply Development + Operations = DevOps

Figure 2: DevOps

There is a need for teams to be responsive in system maintenance as well as dealing with potentially rapidly changing requirements. For these reasons one of the main parts of DevOps is to shorten the development cycle as a whole, reducing the time it takes for development to go into production. It promises continuous delivery without removing software quality. DevOps increases innovation whilst retaining operational quality. Development and Operations become a responsibility that is shared by all members of the team, increasing agility and flexibility which is very important in the rapidly changing technological environment of modern times. The DevOps team also has to prioritise customer requirements and relations, there is now no barrier between the developers and the stakeholders.

Devops is in part writing code to automate operational tasks, using infrastructure as code (IaC). It is a programmatic description explaining how an application should be deployed and run. Instead of a trained expert doing it manually, possibly taking a long time setting up new services or adding features, running a script would be enough, with the developer not having to have the operational expertise. But of course someone with that skill set is also needed, to help plan it from the start whilst writing the software

Going with pure agile might mean that feature related requests might get prioritised over infrastructure related tasks, performance or security. Make sure that non-functional requirements like reliability, security and performance are included in the backlog. Get the operations team thinking like developers when it comes to planning iterations. DevOps takes agile processes out of just the development team and applies it to operations things as well. Continuous delivery really helps with the agility, that is welcoming to new or changed feature requests (Buchanan, 2020).

The operations team already has expertise that it can teach the software development team to make sure that the operational side is not an afterthought but organised into the agile software development phase. Operational tasks need to be testable, maintainable and scalable just like the software solutions, the developers should be experts in this. This has significantly shortened the time it takes for a feature to be developed and released to production. The QA cycles are a lot shorter with less communication lag and continuous integration removes a lot of the time overhead and things can be released to production even multiple times a day, even in larger scale projects.

5 What makes a good DevOps team?

Having an office with employment perks such as restaurants, yoga classes and table tennis, might seem like it started with the internet companies of the nineties, but companies like IBM have practised the same since the early twentieth century. They recognised that a happy workforce could build a stronger corporate culture and that a strong culture would make the staff more devoted. This was enforced using common goals and slogans to support them. Many other successful companies have followed suit like Google and Facebook (Braswell, n.d).

The key to good DevOps teams is also culture, using the same methodology to create common goals for the team is essential for the team to succeed. As DevOps relies heavily on collaboration and knowledge sharing between two parts that historically were separate, good communication skills are essential for members of the team. DevOps is essentially a cultural shift. The talent and knowledge of developer and operations teams is already there, shifting them to work together on common goals is the key. Knowledge sharing and transparency is important as the teams teach each other valuable skills that enable them to think about DevOps from the very start, successfully shifting the whole process to the left from operations to development (Walls, 2015). DevOps teams end up with full ownerships of their product, meaning it's developed from the start in such a way that the team expects to be the sole maintainer of it for the foreseeable future instead of handing it off after development finishes. By creating a culture of pride where the team takes pride in their product and cares about it you can achieve this goal. Rebranding your operations and infrastructure teams as DevOps is not enough to create a great DevOps team.

6 Why not DevOps?

Unfortunately, although it does all sound very good, DevOps is not the answer to every single development process problem out there. There are scenarios where it might not be the optimal solution.

You don't need to go all in with DevOps either and can cherry pick parts that are suited for your business case or move towards it gradually when possible. If your projects don't call for frequent releases investing time in automating release processes could be unnecessarily costly. Some highly regulated industries can also benefit from the strict process of traditional rational methods where things like formal methods might be needed to prove success and there is no room to rollback of failures. Think space rockets or healthcare. In some organisations working with different methods might have proven very successful and very rooted in the team culture and an unnecessary battle to try to get everyone on board with changing to DevOps. According to Kuhn, a new paradigm would eventually take over as older generations leave the workforce and younger ones follow the dominant newer paradigm leading to almost everyone following it eventually.

Lastly, if DevOps was the answer to all our problems, what would push us to explore new and exciting things?

7 Future challenge: Culture in distributed teams

2020 saw a sudden shift to most of the tech industry needing to work from home calling for a quick adaptation to distributed working globally. It could be argued we're a part of the biggest working from home experiment ever conducted and the results from it will be interesting. Does it affect developer productivity, will it have side effects such as re-building some of the organisational barriers that had been knocked down by combining developers and operations? There are many more questions and we will get answers to some of them in the next few years.

And as previously addressed, culture is the drive for DevOps, but how do you create a distributed working culture? It violates some of the pillars of the agile manifesto (and agile and DevOps are obviously friends) such as co-location (Manifesto for Agile Software Development, 2019). By having teams co-located their most likely to adhere to local cultural practices for that place, this provides a challenge for distributed teams as not only do they have to fit in with company culture but that culture has to be highly inclusive of global cultural differences. People have to be open, accepting and willing to learn new perspectives that could challenge their own. This is certainly an opportunity to develop with much needed diversity in technology for more accessible and fair products.

Time Zones are definitely also a hurdle and opportunity. You will have to change the type of communication used as most will likely be asynchronous when teams are spread over the world. But it can also provide the teams with near 24h cover of shifts depending on location. Incident management can then be done on the spot by the distributed workers meaning that the on-call schedule DevOps teams often have to follow could be redundant.

What effect working in distributed teams will have on the worker is also an open question, some prefer working from home whilst others prefer an office setting. Balancing the two to make everyone happy could pose a challenge. The way people work could change with this shift as their working day could become more flexible. This is a big advantage for people with mobility issues that are not well accommodated by an office environment (Distributed Agile Software Development, 2020).

What stands for sure is that it's more than ever that the team needs to agree on objectives and goals, come up with plans to achieve them and make sure that collaboration, knowledge sharing and teaching is done as we don't want to rebuild the wall between operations and developers.

This drastic change could very well lead us to the next paradigm shift in software development.

8 Conclusion

The trend of automation and agility doesn't seem to be on it's way out anytime soon and it has made DevOps the new industry norm. We're still far from creating perfect software and our methodologies can probably be improved indefinitely as demonstrated by the world changes. Maybe we will shift even further left, merging together more of the business analyst side into the development cycle. As data analysis and machine learning grows, we will need frameworks that suit big data as well. We could also experiment with an even more experience based and artistic approach where iterations could shuffle back and forth even more.

The options are endless and as long as we stay willing to change we will learn more about what type of software practices suit specific problems. If we stay agile with a DevOps mindset it should be an easy transition when we change our frameworks in frequent incremental steps, not afraid to make mistakes and working together on a common goal without barriers.

References

[1] M. A. Awad. *A Comparison between Agile and Traditional Software Development Methodologies.* The University of Western Australia, 2005.

[2] A. Bird. *Thomas Kuhn.* Stanford Encyclopedia of Philosophy, 2018.

[3] S. Braswell. *From IBM to Google, the birth of company culture.* n.d.

[4] I. Buchanan. *Agile and DevOps - Friends or Foes?* n.d.

[5] Maurer F. Chau, T. *Knowledge Sharing in Agile Software Teams.* Logic versus Approximation Lecture Notes in Computer Science, 173–183. doi: $10.1007/978\text{-}3\text{-}540\text{-}25967\text{-}1_12, 2004$.

[6] Wikipedia Contributors. *DevOps.* https://en.wikipedia.org/wiki/DevOps, 2019, December 1.

[7] Manifesto. *Manifesto for Agile Software Development.* https://agilemanifesto.org/, 2019.

[8] P. Ralph. *The two paradigms of software development research.* Science of Computer Programming, 156, 68–89. doi: 10.1016/j.scico.2018.01.002, 2018.

[9] J. Shallit. *A Very Brief History of Computer Science.* University of Waterloo, 1995.

[10] M. Walls. *Building a DevOps culture.* https://www.oreilly.com/content/building-a-devops-culture/, 2015, September 25.

[11] Wikipedia. *Distributed Agile Software Development.* https://en.wikipedia.org/wiki/Distributed$_Agile_Software_Development, 2020, May27$.