DevOps in practice

How it emerged, what it is and its impact

Benjamin Tellström

Tuesday 30th April, 2019

Introduction

Development and Operations, two words that are often combined in the o so elusive term DevOps. It is a movement, a philosophy and a set of practices that has been in development since 2009, or at least since 2009 with that name. In truth, it is almost impossible to speak of DevOps without also speaking of agile development, something that has itself had that name since 2001. Agile development is perhaps best known for practices such as scrum and extreme programming, both methods from the mid 90s. Another very important concept is that of *Software as a service*, as well as the emergence of design patterns such as microservices. There is also one last concept that must be addressed when speaking of DevOps, that of business metrics, and their role in development and evolution of services and businesses.

Development models

Software development has come a long way since punch cards were used as input to computers, and just as the computers evolved, so did the methods for developing software. For a long time, the waterfall model and similar approaches to software design were the most prevalent. Typical characteristics of these were that one were given specifications and requirements from the customer at the start of the project. From these one would then produce software analysis and design to determine just how this piece of software would be implemented [8].

Following the design of the software, one would implement it in code and finally begin testing and debugging. Once this was achieved to a sufficient level, one would start operating the software and support it.

Note that even at the time of its inception, this model was deemed flawed, however it was still in frequent use, albeit modified in some ways. The United

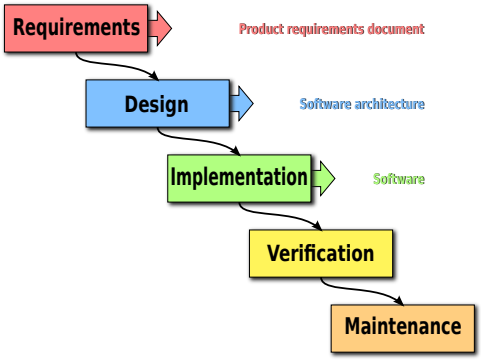

Figure 1: An image of the waterfall model. Source: Peter Kemp / Paul Smith - Adapted from Paul Smith's work at wikipedia, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=10633070

States department of defence even specified that many of these steps should be mandatory when dealing with software contractors [6].

The waterfall model has faced a lot of criticism, frequently noting that it is very static, and once one has begun implementation and reaching the later stages of development, the software will be difficult to change. The model does not allow for any flexibility, or in another word, agility [8].

From this criticism, models that were deemed agile emerged. Focus was now instead on making the development process flexible, allowing for changes during any stage of development. This in an attempt to remove the barrier between customer and developer, attempting to create something tailored to what the customer actually wanted.

The agile movement culminated in 2001, when the *Manifesto for agile software development* was published [1]. It stated four points

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

These points have come to shape the way software is developed ever since.They have often been applied using some of the above listed agile methods of development, and flexibility and adaption became the new focus when developing software.

As these new methods were adopted, the software industry also changed. Internet speeds accelerated and cloud computing emerged. This also changed the way we are able to interact with software, and instead of having to run

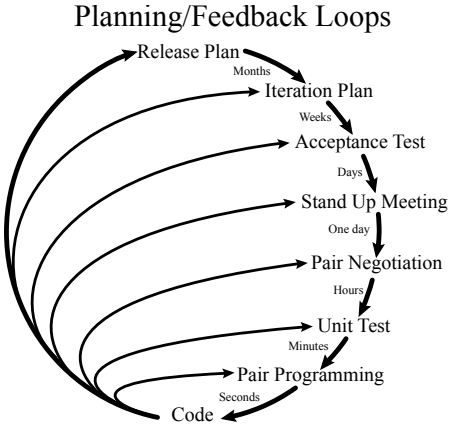

Figure 2: An image describing the principles of extreme programming, one of the typically agile development methods. Source: Don-Wells - https://en.wikipedia.org/wiki/File:XP-feedback.gif, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=27448045

everything on your own machine, it enabled one to work completely remote. No need for your own data centres and powerful development machines.

With this development came the business model of software as a service.

The business model and its impact on development

Traditionally, one has developed software that the user themselves should be able to run on their machine. However, the increased internet speeds made sure that this was no longer the case. Instead, one could develop for a server, which then a user would connect to through the browser and then allow the server to do all of the heavy lifting for it. One did no longer need to sell licenses for the software, instead one could demand fees for the use of the service.

The fact that all your software instead ran on your own servers made compatibility issues and requirements for long term support unnecessary for these companies, since they could themselves update whenever they please and support only until the next version. From this came the need for technologies such as continuous integration and continuous development.

When you no longer need to support old versions, you can focus on releasing new versions, but then also the number of integrations and deployments increase, and since these have historically been tedious and lengthy processes, new methods were needed here as well.

The focus on flexibility, as well as the model of software as a service demanded a new workflow and technology. A developer should not need to wait weeks to get feedback on a change, nor should there have to be weeks or months between deployments. This sparked the idea of DevOps. The aim? Reduce time for feedback and deployment.

The bottleneck in the above process was at which speed one could integrate and deploy software. The people deploying and operating software were not the same, so the new goal? Make sure that they can interact and understand each other. Hell, break the barrier between them completely! The developers become the people responsible for their own code. If it breaks, they fix it. They also became the people responsible for integrating and deploying the code, if something goes wrong live, the developers fix it.

DevOps, the concept

This is the idea behind DevOps, break down the barrier between those operating the software and those creating it.

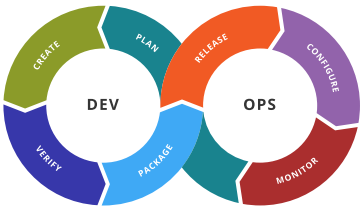

Figure 3: An illustration of the idea behind DevOps. Source: wikimedia under CC by SA 3.0 license.

As one can imagine, this was more than just a technical detail. In general, the development department does not know the same stuff that the operations department does, and vice versa. The way they work, the skills they have and the requirements for their work are all different to that of a developer. This ultimately makes collaboration and breaking this barrier quite difficult, and it is from this difficulty the concept of a silo arises.

A silo is a team or department of a business whose working methods are obscured from most others in the business. There is something in their environment or group, whether it be them sitting in another part of the building, making them physically isolated, or leadership that discourage seeking help from other teams one way or another [2]. Silos are what DevOps ultimately want to avoid, and in fact sharing seems to be one of the main factors for success within DevOps as well [5].

Metrics, the driver of the business

Business metrics are everywhere, the *cold hard facts* they may provide is very desirable for businesses, since they may often show were a business may improve as well as where resources could be spared. DevOps has grown very reliant on such metrics, and also promotes the use of them. The ability to automatically generate such numbers as well as the speed at which the business may respond to them gives a DevOps team quite an edge in focusing their attention correctly.

There are also metrics that are directly relevant to DevOps, such as time to deploy and test suite execution time. These are both metrics that very much describe how well a business attempting to employ DevOps are succeeding at this. A business with test suite that takes to long will often see developers work on something else in the meantime, a context switch that reduces performance. The aim is therefore to keep these numbers as low as possible, so that one may optimise performance and at which rate developers get feedback. Feedback 1 week after a developer has completed a feature is far less effective than that given the next day.

Well, with the above metrics one may ask, how are we to develop? Projects tend to grow in size, and it is not rare to see software that is above one million lines of code in large applications. How would one create tests for such an application that gives rapid feedback and completes in a sufficiently short time? One answer to this question is the design pattern known as microservices.

Microservices is essentially taking the modality to the extreme. You develop a separate program for every part of your service. A business might have a separate team for each service, lending itself well to the agile working methods described previously, since these were generally designed with smaller teams in mind. This also benefits the continuous development and integration part, since each individual program may be tested and deployed much faster than if one were to have one monolithic program in which every part would have to be tested individually [4].

This is not the sole solution, and one may for example divide the test suite into many small parts, or perhaps run only those tests that are dependent on the files or functions that have been changed.

What is the result?

I've described a lot of different methods, concepts and tools associated with DevOps, but is there any proven effect that this increases performance? Are clients in general happier with keeping a constant dialogue with developers?

And last but not least, how does this affect the average employee? Since developer and operations have historically had such distinct roles, would this require an employee to take on several roles, or is this solved in some other way?

First, I'll address the performance aspect.

DevOps definitely succeeds in several aspects. Time to deploy is now a metric that can be measured and improved upon, and deployments can happen several times a day. This is far better than any waterfall model could ever hope to achieve. The time to integrate has also drastically fallen, and both of these can be very well attributed to the introduction to automatic integration and deployment techniques and technologies.

But the speed at which one deploys is not necessarily an indicator of high performance, one may simply deploy many small bug fixes while the software itself remains slow and unchanging.

However, measuring the performance of such a widely defined concept is quite difficult, so for the matter of this essay I will instead focus on the performance changes due to agile development techniques.

Research around agile methods seem to have decreased with the coming and going of 2010, and that is without necessarily coming to any conclusions about agile methods themselves. Studies have been made on individual methods such as pair programming and in general been positive, but no conclusive studies have found that agile methods would increase performance more than other methods. However, there seems to be a positive correlation between project success and the use of agile development methods [9].

Of note is also the fact that most literature state that a hybrid approach is employed rather than a specifically agile one [9]. It is also stated that many of these methods were already employed before the publishing of the agile manifesto and the handbooks in the 90s.

To answer the second question above, the client satisfaction has actually seen an increase with the use of agile methods [3]. This is speculated to be largely due to the focus on individuals present in the methods. Considering that DevOps employ the same methodologies, a long with a larger focus on metrics, especially those focused on program usage, these effects should also be seen when employing a DevOps approach. Interesting is also the fact that stakeholders seem more content when an agile development method is used [9].

Lastly, how does a DevOps approach actually effect the employees that have to work with and implement it? Turns out, that they also tend to be quite satisfied. People that call themselves DevOps specialist and site reliability engineers seem to be some of the most content employees, with below 13

% actively looking for another job [7]. They also seem to earn the highest salaries, so this might be another reason why they are content [7].

Of note is also the fact that DevOps can be implemented very differently in different businesses. Some use it as an extension of the agile methods described above, while some dedicate departments to DevOps, creating a new role rather than removing that of operations. The implementation of it is not unified, and roles are just as in agile development difficult to define. For this reason, the working experience may vary significantly between businesses, and makes the role of DevOps engineer quite difficult to define. Mileage may vary.

Ultimately, what we can say about DevOps is that it is very good at delivering what it was designed to deliver, rapid integration and deployment. However, the development methods associated with DevOps have yet to be proven to increase productivity and performance, but they do for certain deliver higher customer satisfaction. DevOps doesn't just improve customer satisfaction either, it actually also seems to increase employee satisfaction, and salary [7]. How to implement DevOps efficiently is however still an open question, and attempts may result in vastly different workplaces.

References

[1] Beck, Beedle, van Bennekum, Cockburn, Cunningham, Fowler, Greening, Highsmith, Hunt, Jeffries, Kern, Marick, Martin, Mellor, Schwaber, Sutherland, and Thomas. Manifesto for agile software development, 2001.

[2] Audra Bianca. What do silos mean in business culture?, 4 2019.

[3] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10):833–859, August 2008.

[4] M. Fowler and J. Lewis. Microservices, a definition of this new architectural term, 3 2014.

[5] A. Mann, M. Stahnke, A. Brown, and N. Kersten. Pupper - state of devops report 2018, 2018.

[6] US Department of defense. Defense system software development, 6 1985.

[7] Stack Overflow. Developer survey results 2019, 2019.

[8] W. Royce. Managing the development of large software systems. *Proceedings, IEEE Wescon*, pages 1–9, 8 1970.

[9] Pedro Serrador and Jeffrey K. Pinto. Does agile work? — a quantitative analysis of agile project success. *International Journal of Project Management*, 33(5):1040–1051, July 2015.