The importance of DevOps in relation to other methodologies

Henry Luong
Kartik Kandavel Mudaliar

May 2019

1 The classical challenge

Historically, organizations used to clearly define the way of working of the development team and operation team, also known as the Dev team and Ops team. As a result of this clear line between the two, there was very little overlap between the working areas of the two teams, and thus the expression "throwing code over the wall" as depicted in Figure 1, was born to mock this poorly integrated setup. This expression implies the fact that back in the days the Dev team simply create the code and decide it to work after some amount of local testing on their environment and just throw it out as a package over to the Ops team to operate in the production environment. If some problems occurred at the Ops team then the package will be sent back to the Dev team along with a short message "it doesn't work" for fixing in case of bugs. To make this matter worse since the Ops and Dev team might have been working in completely different platform, some bugs found by the Ops might not even be detectable by the Dev, hence this might result to an unending game of volleyball between the Dev and Ops team. To tackle some of these communication problems between the Dev and Ops team, the Quality Assurance team, abbreviated as QA, was introduced as an attempt to oil the gears between the two in the hope that the frustration and friction will lessen to promote a smoother development process. The QA will primarily try to ensure that the software will meet the specified requirements by carrying out extensive tests for quality checking of the software and secondly it will also strive to improve the quality of development overall by for instance communicating as clearly as possible the errors occurred at the Ops back to the Dev or improving the life cycles quality by systematically looking for interesting patterns observed during the interaction of QA with other teams.

While the existence of the QA does, in fact, serve as a patch to more or less fix the current issue, some fundamentals problems like the different platform still remains. By introducing a "plugin" to the current situation does not guarantee that fewer errors will occur, since the QA instead of just fixing the original problem might actually become a part of the problem itself. As an example, to this, the Dev team now sends the code over to the QA and QA send this

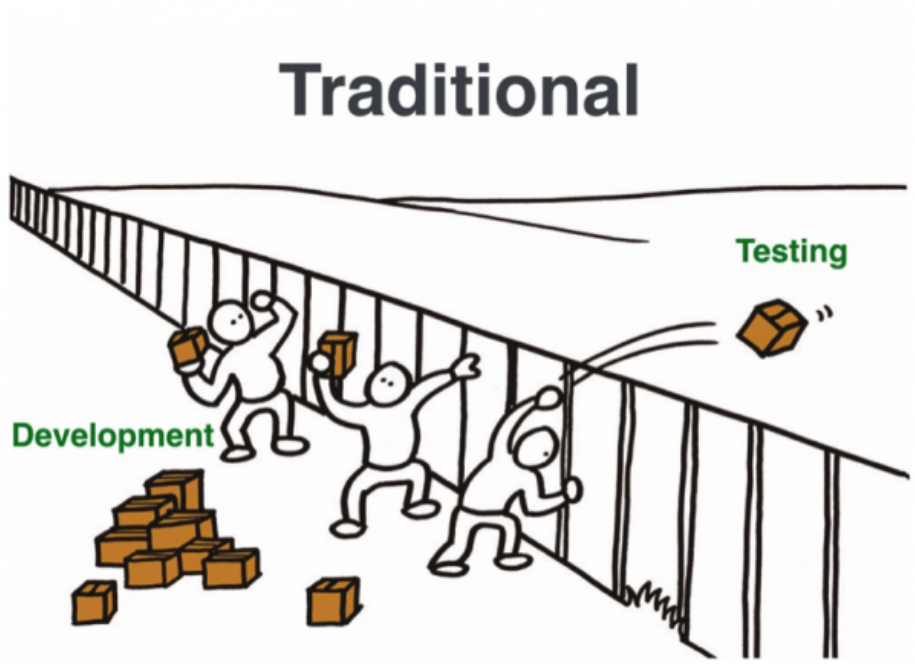

Figure 1: The old days of unending game of volleyball between the Dev and Ops

code back due to errors founded after they ran their tests at their environment. Then out of frustration, the Dev tries to shift the blame to the QA by saying that "they did not test the Dev code properly, therefore, the bugs are not their" or claiming that "the QA refuse to look at a bigger picture and cut them some slacks by ignoring the bugs". After some passing back and forth between Dev and QA, the code finally works at both the QA and Dev environment, but then when this piece of work arrived at the Ops environment it breaks again. This creates an evil cycle of unending tragedy. Instead of breaking down the wall between the Dev and the Ops, it has replaced the original wall with two new walls as illustrated in Figure 2, where one of them is between Dev and QA, while the other is between QA and Ops. This conclusively leaves the problem of improving the organization's productivity through collaboration between the Dev and Ops unresolved.

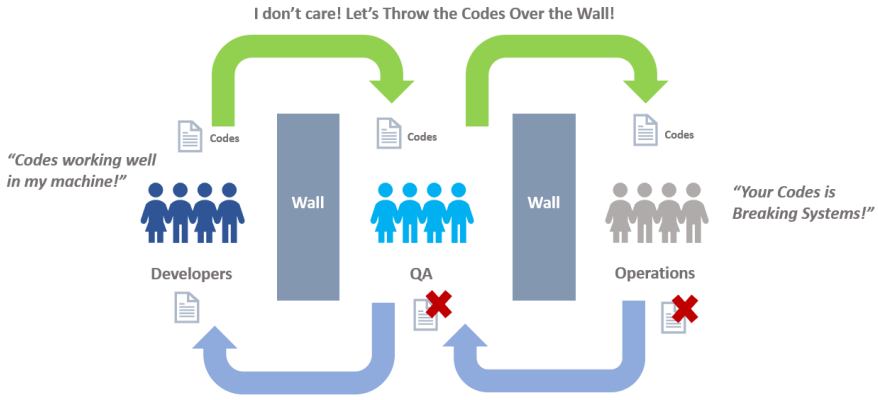

Figure 2: Did the QA solve the problem ?

In short, the legacy code deployment strategy promotes the phenomena so called siloization depicting the fact that the Dev and Ops were functioning quite independently, which lead to current inefficient management and as a result also increase the overall cost for the final product. The concept of siloization is not new since it already existed in the infantile period of software engineering. Back in the day's companies try their best to isolate and prevent knowledge leakages in order to make their competitors' work as inefficient as possible. Due to this mindset, each part of the organization becomes isolated, and subsequently, this makes the overall working structure rigid when there is a need for changes.

2 Popular development methodologies

While the introduction of QA as a mediator between the Dev and Ops has proven to be rather unsuccessful as it attempts to fix one problem and just ends up creating more problems, there are also other methodological approaches such as Waterfall, Agile and Lean during the course of software development history. Although these methodologies carry different names and perspectives, they all in actuality have one big common goal, which is to improve the work efficiency of software development.

2.1 Waterfall, Agile and Lean model

The waterfall model, figure 3, was formally introduced in software engineering as an idea through a paper published by Winston Royce. The model was believed to be a very structured approach as it was inherited from the hardware manufacture and software strategies. The culture of the waterfall model is such that testing only occurs at the end of the project which may be very expensive,

provided that there is a need for modification, this will cause the re-start of the entire process, from the requirements down to the very last phase in the lifecycle. It is apparent that the waterfall does not, in any way, support the continuous delivery which entails the fast and automated feedback on the production of software which, in other words, leads to the deployment of software at the end of each phase in the lifecycle. These processes are clearly not present in this model.

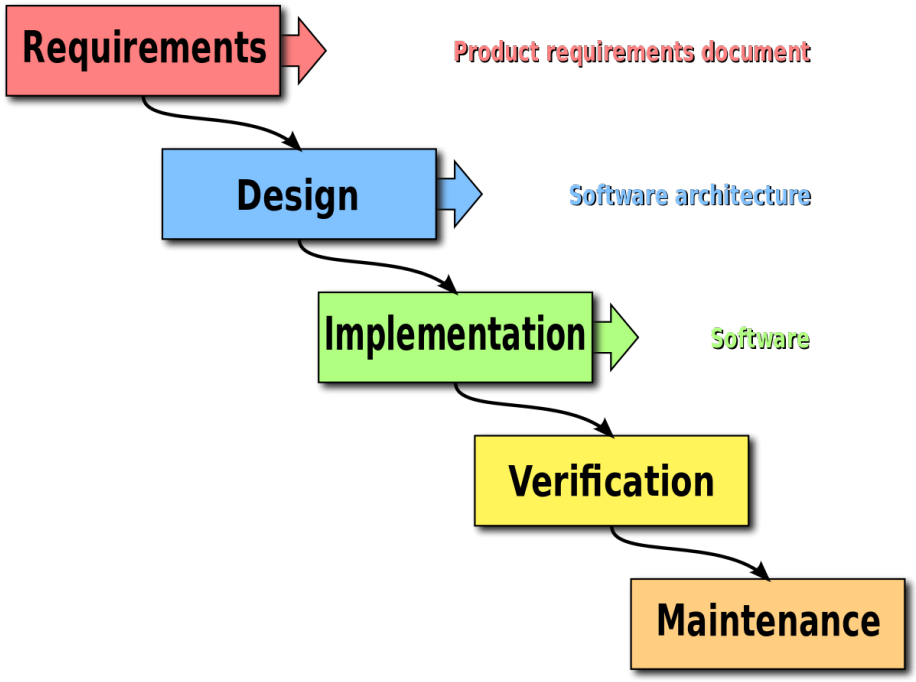

Figure 3: Waterfall Model

Unfortunately, the linear approach being very inflexible does not support a sudden change in the development cycle. This led a set of software developers to break away from the traditional, structured, and bureaucratic approach to software development and ported to a flexible development style hence, the agile methods which were proposed in the 1990s. This, in fact, made agile to focus more on the software than the design and documentation. With the manifesto developed for agile software development, it is crystal clear that agile is not a process, unlike the waterfall model, but it is a culture. This is such that, the mere thoughts of the term agile depicts the rapid delivery of something, which, in our case, is software. Having said that, an inkling thought have been gotten regarding the fact that the culture of agile embraces the aspect of continuous delivery and integration. Continuous delivery simply shows the rapid process of the transfer of coding aspect into the building, testing and deployment phases

through continuous integration, meaning that the binary which is deployed into the code had been committed into a source control which makes it easy to deploy the codes into building and, subsequently, testing, having detected errors and bugs early enough. On the other hand, these beneficial features of Agile do not come without demerits. To accomplish flexibility and fast deployment a high degree of freedom is needed, therefore Agile team tends to suffer unpredictability due to a lack of a clear project definition like Waterfall. Other than that, members in an agile team also need to be "agile" or cross-functional, which means that we need to effectively assimilate people with different sets of skills with each other for certain jobs and this will lead to scaling problem when there is a need for a bigger team. Briefly speaking Agile removes the shackles faced by the Waterfall model shown in Figure 4, by enabling feedback acquisitions not only at the final deployment stage but also in the middle stages to flexibly and quickly adapt to the changes detected during the development in exchange for a less clear development structure.

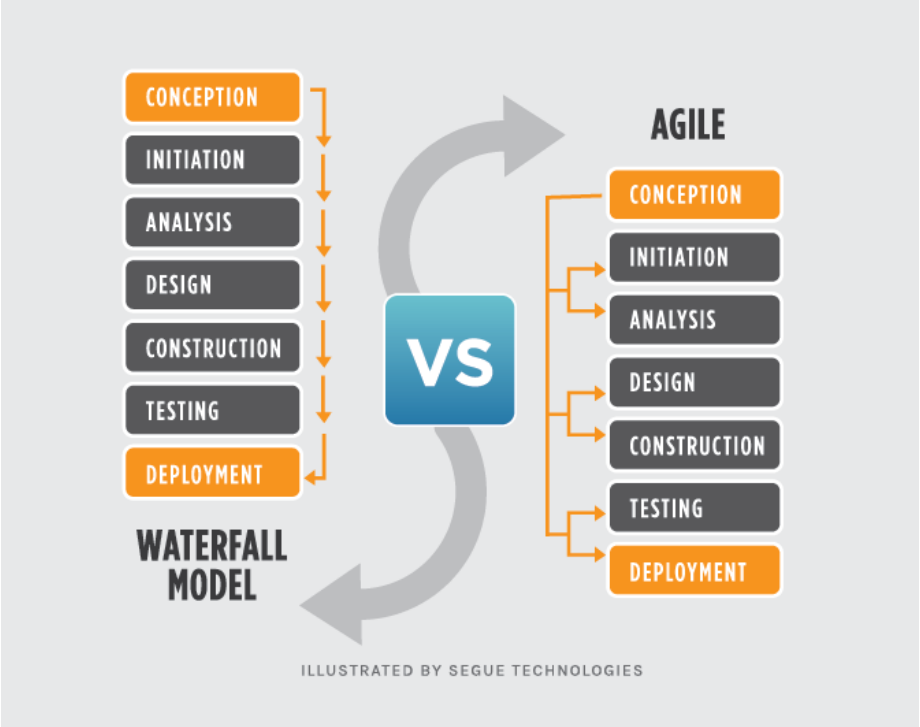

Figure 4: Waterfall vs Agile

In general, the global competition that is present in the community of software development has resulted in an ultimate search for a way in which qualitative software can be developed in a rapid, repeated, and reliable fashion. Other than Waterfall and Agile there is also the infamous Lean which in fact

was originated from the manufacturing industry as far back as 1950, during the Toyota Production System (TPS) [9]. Lean, with the aim of minimizing resource wastage during production, has got its initiatives in the manufacturing, services, and product development which have led to improvements in cost, quality and time of delivery of products. In reference to Figure 5, Lean is a way of thinking in which organizations can specify their values, line up their value-creating actions in a proper sequence, conduct these activities when requested without any form of interruption, and then perform them in an effective way. It is crystal clear after having analyzed the principles and practices of lean culture, it does not accommodate waste and any form of unevenness in the development process of a software. Also due to this core value of "saving as much as possible" of Lean, the organization using this model will tend to be overly frugal, which can lead to understaffed work teams or some small improvement is not worth to make as saving money is more important.

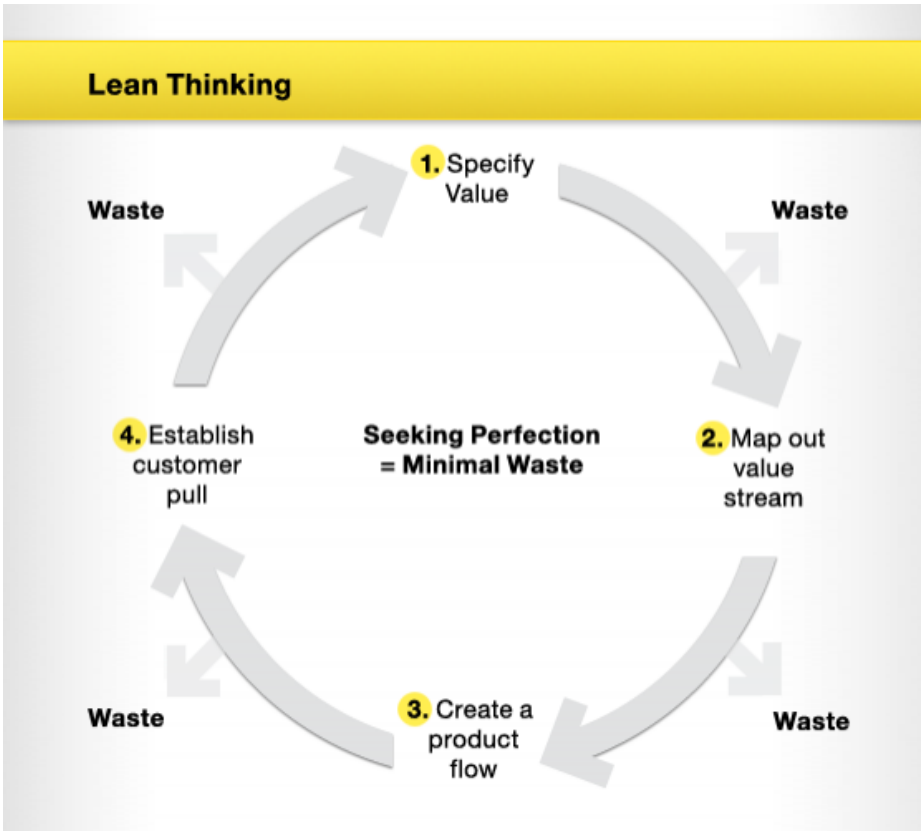

Figure 5: Lean model

2.2 What's Next ?

In short, while Waterfall invested heavily in a very structured life cycle, Agile together with Lean lay more focus on the relation between business and development in their own separate way and despite having their own advantages, these methods still did not quite completely solve the initial challenge as for instance they still overlook, if not outright ignoring the connection between development and operation.

Hence to tackle the problem in a better way, DevOps as being derived from both Lean and Agile, is able to achieve both fast deliveries and minimal wastage by bringing the Development and Operations together through its own culture of collaboration and set of methods to evolve the organization by using highly automated approaches without being inflexible as Waterfall.

3 What is DevOps ?

3.1 History

DevOps began in Belgium, in the year 2007 [1], when Patrick Debois who happens to be a consultant took a project with the government ministry regarding a large data centre migration. In this project, he happened to oversee the testing where he struggled in relating to the development and project team. This is the platform on which the idea emanated thereby formulating DevOps as a bridge between developers and testers. In comparison to Waterfall, Agile, and Lean, DevOps is the proverbial new kid on the block. It is however not only just a methodology or a framework but also a way of thinking behind the usage of these concepts and tools. To point this out clearly, what makes something "DevOps" is not just the characteristics of the object, but rather the manner of their use also matters for achieving top efficiency and with that logic, Docker, for instance, is not DevOps by itself but rather it can be used in a DevOps way.

Like other methodologies, DevOps also has its own set of core fundamentals[12] to follow for effective software development. While these principles can be stated in a different way they contain more or less the same ideas and right below here is a list of some of these important main ideas.

- **Cross-functional team and skills:** DevOps aims to expand the boundaries by integrating the process of operational and functional roles while Agile focuses on reaping the benefits of integrating development and quality assurance roles.
- **Continuous delivery:** Ongoing updates, applications, and every important aspect of the development life-cycle are made more frequently in smaller sizes in order to avoid service disruption.
- **Optimum utilization of tool-sets:** The cross-functional teams i.e. development and operations implement the same tools where necessary.

- **Automated deployment pipeline:** DevOps initiates the creation of automated deployment model pipeline in order to allow for a friction-less deployment in the development life-cycle.
- **Continuous assessment:** The DevOps environment recommends the continuous assessment of application development and release process.

3.2 DevOps Mentality

As a quick recapture of what have been said previously, It is necessary to point out that the integration friction between the Dev and Ops from a perspective of DevOps cannot be tackled by simply introducing a QA part, as because the underlying problems of the design originates from the lack of attention for a collaborative culture within the organization. In DevOps things such as siloization or blame culture are regarded as anti-patterns and not only will these anti-patterns ruin the workflow of the organization, but It will also lead to dire consequences such as drastic increment in production costs or production delay due to siloization, since for instance key persons being absent due to illness cannot easily be replaced.

Metaphorically speaking in order to tear down the wall between the Dev and the Ops, we need a new way of approach and this new approach need to emphasize not only the interaction of the development and operation but also the team culture and individuals behind the scene also need to be considered. Hence, DevOps as a novel approach accommodating the aspects overlooked by other methodologies offers four most important principles to follow shown in figure 6. Firstly, it is the tools which should be chosen with the aim of promoting both individual productivity and the collaboration of the team. However, only having the greatest tools for code writing does not mean that the production process will go smoothly, therefore some attention should be paid to automation as well. Apparently, an automated procedure is usually better than a manual one, not just because it reduces the time spending on less productive activities, but also because it makes the development being more defined, which in turn bridge the gap between the development and operation. Other than the tool and the automation process, the collaboration between team members and a blameless culture do also have important roles to play as they promote the existence and growth of the cross-functional team and skills. Moreover, being in a blameless culture with good collaboration also bolster rational thinking for a continuous assessment, which improves the chances to really catch the real problems instead of the public shaming strategy. This aspect does in fact clearly disagree with the traditional way, where each team individual usually has a fixed set of jobs to do and when something bad happens there will be an extensive search for the culprit. Rather than thinking that errors are caused by malice or incompetence, DevOps instead encourage that the effort should be spent on looking for the faults in the structure in the system since a blame culture or public shaming will only lead to paranoia and fear causing the grown of the organization to be weakened as a result.

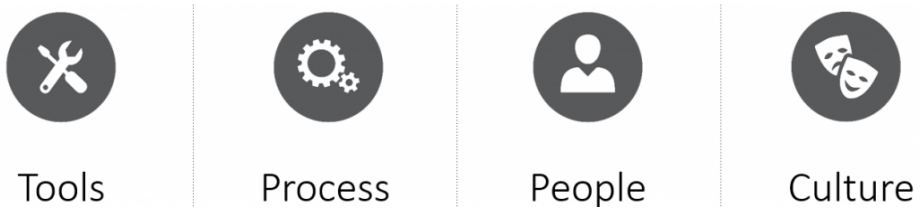

Figure 6: Important principles of effective DevOps

4 Conclusion

Waterfall	**Agile**	**Lean**	**DevOps**
+Clear development structure	+Flexible	+Highly productive	+Great workflow
+Highly methodological	+Fast deployment	+Clear working ethics	+Highly flexible
-Inflexible structure	-Hard to predict	-Stingy tendencies	+Cost minimization
-Expensive	-Hard to scale	-Overlook improvements	-Hard in practice

Table 1: Summary table for different methodologies

In summary, DevOps has proven to be more capable of accomplishing what has been missed out by the other methodologies such as overlooking the connection between culture and the wall between Dev and Ops . Not only DevOps has successfully improved productivity to tackle this problem, but also it has demonstrated that there are more to software developments than just the building, testing and throwing packages out to the customers. As a novel approach, It has shown that collaboration and the culture behind the organization are as essential as the method of software development itself, and this was also the key point behind the existence of the wall between those two parts.

Successful DevOps is however still a great organizational challenge till this day. It is usually easier to change only the work methods rather changing someone's mindset, not to mention that most people are already used to the old way, which makes it difficult to transform an organization into a DevOps one. Moreover, according to DevOps, there is no absolute winning way to develop something not only because of the culture of working as being similar to real cultures varies between organization to organization, but also different methods do have different merits and demerits as shown in table 1. Hence, **despite being fruitful in some area, the way of DevOps should only be regarded as a good principle to be followed rationally and not blindly**.

References

[1] Davis, J., Daniels, R. (2016). *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale.* " O'Reilly Media, Inc.".

[2] Kim, G., Humble, J., Debois, P., Willis, J. (2016). *The DevOps Handbook:: How to Create World-Class Agility, Reliability, and Security in Technology Organizations.* IT Revolution.

[3] Loukides, M. (2012). *What is DevOps?.* " O'Reilly Media, Inc.".

[4] Kim, G. (2011). *Top 11 things you need to know about DevOps.*

[5] Lwakatare, L. E., Kuvaja, P., Oivo, M. (2016, November). *Relationship of DevOps to agile, lean and continuous deployment.* In International Conference on Product-Focused Software Process Improvement (pp. 399-415). Springer, Cham.

[6] Jabbari, R., bin Ali, N., Petersen, K., Tanveer, B. (2016, May). *What is devops?: A systematic mapping study on definitions and practices.* In Proceedings of the Scientific Workshop Proceedings of XP2016 (p. 12). ACM.

[7] Rockwood, B. *The DevOps transformation.* keynote at the USENIX LISA, 11.

[8] Balaji, S., Murugaiyan, M. S. (2012). *Waterfall vs. V-Model vs. Agile: A comparative study on SDLC.* International Journal of Information Technology and Business Management, 2(1), 26-30.

[9] Liker, J. K., Convis, G. L. (2012). *The Toyota way to lean leadership. McGraw-Hill,.*

[10] Schwalbe, K. (2014) *Introduction to IT Project Management*

[11] Sommerville, I. (2012) *Software Engineering*

[12] Httermann, M. (2012). *DevOps for developers.* Apress.