DevOps for Software Engineering

Khondoker Yasin Ahnaf Prio
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
yasin011@morris.umn.edu

ABSTRACT

DevOps is a combination of practices and tools in software *development* (Dev) and *operations* (Ops). DevOps focuses on: collaboration, automation, continuous integration/delivery, continuous testing, and continuous Monitoring. In this paper, we are going discuss why DevOps is important and the problems it tries to solve. We will then dive deeper into continuous monitoring and automation by looking at the design and implementation of an "FG" (Filling-the-Gap) tool. The paper then concludes with a summary of research which tries to evaluate DevOps and how it can affect software engineering.

Keywords

DevOps, Automation, Continuous Integration, Continuous Delivery, Collaboration, Continuous Feedback, Continuous Monitoring

1. INTRODUCTION

DevOps, the combination of *development* and *operations* according to Jabbari, et al [1], is a development methodology in software engineering that aims to improve software quality, the software development design cycle, and, the maintenance of software after deployment [4]. When there is a gap between the development and operations team, it can lead to a lot of conflict and friction in software companies. For example, development teams often focus on the constant addition of new features to the product, where operation teams focus on stability and maintenance of the software instead. As more work is being done on features, and then tested and staged, the development environment is evolving in ways that operations might not be aware of. The operations team might utilize more tools to implement smoother maintenance that the developers might not know about either. It is fairly difficult to ensure proper software engineering when different teams are responsible for different environments and live in their own worlds. The practice of DevOps aims at closing this gap between operations and development [5]. By promptly using DevOps, developers can identify bugs in the code immediately and correct them way earlier in the software development life cycle tackling the problem of high costs for problems and defects that are found later in the cycle. DevOps helps to guide effective business decisions by using metrics driven monitoring to identify product features that need more attention. DevOps also provides opportunities for open communication and collaboration. This helps guarantee the sharing of knowledge between teammates and across teams. To summarize, the development and operations teams can now work together more efficiently to reach their common goals by the practice of DevOps.

In Section 2, we get familiarized with some fundamental concepts and terms in Software; followed by section 3 where we study the five key aspects of DevOps. In Section 4, we learn about the architecture of a DevOps tool that aims to enhance software engineering. In Section 5, we are then introduced to a study that evaluates DevOps and its impact on the success of software development followed by a brief summary of key findings from all the sections combined to conclude the paper.

2. BACKGROUND

To understand DevOps and how it affects software engineering, we need to learn more about some key terms in the software development design cycle.

Many software development methodologies organize the process of developing software into small iterations to reach the complete final software prod. This includes the act of writing code, analyzing requirements, writing tests, documenting and such. Throughout the entire development cycle, developers collaborate by storing work in a *repository.* In software development, a repository is a central file storage location where developers can use to collaborate. A *Build* is the process of creating the application program through compilation of source code to object code.

Software systems often involve multiple services, which might even be interlinked sometimes, where each of the services has its own goal to ensure for the successful delivery of the software. Each service is labeled as *Service-level objective* (SLO): the objective defined to be accomplished by a particular service for the end user. A popular metric used to measure how effective software components are at solving a problem domain is Quality of Service (QoS). QoS is the measurement of completion of the service-level objective. It will consist of attributes and metrics used to measure the SLO. For example, an application that was made to send message reminders hourly to remind their clients to stay hydrated: the message service component would have the SLO to provide messages on time. An example of an attribute of the QoS for this specific SLO would be the time it takes for

UMM CSci Senior Seminar Conference, April 2019 Morris, MN.

the server to process and send data packets to the client in the form of a notification.

To understand the field of software operations better, we need to be introduced to some other key terms. We need to understand the client-server infrastructure that most software systems utilize. The client (perhaps the customer's computer) sends requests to the server (such as pressing a submit button on a survey or asking to view a specific URL). The server then provides the response based on the request. Sometimes there might be multiple requests from different clients to the server at the same time. A client that arrives to find all servers busy generally joins one or more queues. The server responds to clients according to their relative priority or position in the queuing system.

To understand how the system works operations team often uses a *Monitoring Platform*. Monitoring platforms are used as a tool for testing and verifying that end-users interact with the software as expected. A *server log* is a log file generated by a server consisting of a list of activities performed. For example, web server logs maintain a history of page requests, where one could track interactions taking place in the queuing system. Software monitoring is often used by businesses to ensure uptime, performance, and functionality is as expected. They mostly use the server log file for analysis to achieve this.

3. FIVE KEY ASPECTS OF DEVOPS

In this section, we are going to go into more depth about the five key aspects of DevOps and how they result in more effective software quality and development.

3.1 Collaboration

Collaboration practices in DevOps aim to connect the development and operations team by sharing knowledge. Effective communication can help teams during requirements gathering, development, testing, and deployment. In the DevOps methodology it is proposed that every team member is kept informed about the software product throughout its life cycle [2]. Product development conflicts may arise as a result of poor communication such as information that is received unclear or late [8]. Team members can discuss and seek insights on what they are working on during meetings [8] and can be a platform to keep everyone up to date with the progress made [2] and to prevent duplication of work. Stand-up meetings provide common physical space for a chance of interactions which can be very helpful. [8]

3.2 Automation

Automation refers to the technique of initiating or controlling processes by automatic methods to keep human involvement to a minimum. Automation standardizes how the software is built, how each team responds to change made to the software, and how the software is tested and reviewed. This ensures the processes are executed the same every time, reducing the chances that defects or bugs slip through because of human error. Automation can also improve communications between the development and operations teams by generating automatic feedback at key moments in the development process.

3.3 Continuous integration and deployment

Continuous integration according to Shanin et. al refers to the active sharing of code inside a central repository [7]. If a set of developers merge and share their code with one another frequently, there might be less conflicts when they put it together. In this process, every time code merges together in a repository, automated builds are run to test the software that it aligns correctly with the work already done. In summary, continuous integration refer to the frequent connecting of smaller chunks of code into the main code base to ensure smoother delivery of software.

DevOps supports frequently and reliably releasing new features and products which includes continuous deployment. The continuous deployment practice includes and extends on continuous integration practices. It tries to incorporate new changes into production environments by providing as much automation support as possible [7] and bridge discontinuities between development and deployment teams. In other words, continuous deployment refers to not only frequent integration of new code into the development cycle but also rapid deployment of it on the production environment.

3.4 Continuous Testing

Software testing is the process of evaluating different components in a software to make sure they are functioning as expected. Continuous testing refers to the active flow of incorporating testing inside the development cycle. Developers write tests and code simultaneously. Continuous testing is a constant balance between good test quality and test speed. The goal is have tests that are effective enough to reduce the cost of later testing, where finding bugs and defective code becomes more difficult, while maintaining fast testing that doesn't that much time. Continuous testing does not ensure no bugs whatsoever but increases the confidence significantly that the software deployed will not include them. Continuous testing practices include having automated tests continuously running as the developer writes code, providing rapid feedback about failures the code is being developed [6]. In a study by Saff et. all, "participants using continuous testing were three times more likely to complete the task before the deadline than without" [6].

3.5 Continuous Monitoring

Continuous monitoring refers to the collection of metrics of the deployed software and the sharing of it with developers. Monitoring is often done with the usage of monitoring platforms by operations team in companies but is rarely shared with the developers. The sharing of the reports of the application to the developers aims to provide easy-to-use feedback allowing better understanding of the performance and availability of the application. Continuous feedback from end uses provide visual evidence and full context for analyzing behavior to locate points of work the software that is receiving most action. DevOps promotes the usage of this metrics to drive business as well as development features and design decisions. An example of continuous monitoring is explained furthermore in the description of the FG tool.

4. DEVOPS TOOL: FILLING-THE-GAP (FG)

In this section, we shall learn more about a filling-the-gap (FG) tool [4], that connects the software performance metrics to developers. We will look at the key objectives of the FG tool followed by the architecture of the tool: the FG Design-Time component, the FG Runtime component, and the Monitoring History Database. We will look at the estimation techniques and finally how to analyze data collected

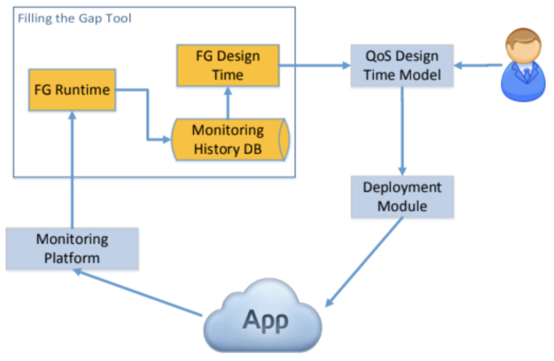


Figure 1: FG Tool Architecture

to improve performance. We will also define a working example of a hypothetical application and try to apply the FG tool in certain contexts and try to evaluate how it might help the software engineering cycle.

For the sake of better understanding of how the FG tool works and how we may apply it, we are going to define the goals and objectives of a hypothetical application as a running example. The application only has one SLO for simplicity, and that is to get notifications on time as reminders to end users who sign up for these notifications. The user will interact with the client and set down the specific timings when they would like to get notified and set their notification message content.

4.1 Objectives

The key objective of this FG tool according to Perez, et al are as follows [4]:

- To parametrize software performance in the design-time and the run-time.
- To provide the developer with a report of the run-time application behavior.

The run-time here refers to our application that is deployed and users are interacting with. In the context of our running example of the notification application, this would benefit us in trying to measure the QoS of notification SLO in run-time. We would be able to know if users are getting delayed messages, times of peak demand, different categories of reminder messages etc. The design-time is where the developers are working on addition of new or refining of already in place features. By measuring them and providing them with the developers, they can now constantly measure success and work more efficiently.

4.2 Architecture of the FG Tool

In this Section, we will learm more in depth about the FG runtime, the database where we store all the monitoring data and then finally the FG designtime.

4.2.1 FG Runtime

The FG Runtime is connected to an external monitoring platform to collect metrics relevant for the performance of the application to collect data for further FG analysis, and this component needs to be configured with the queries necessary for the full extraction of all data relevant for FG analysis.

In our example application, this might include the setting up of a monitoring platform to see the interactions of end users with the input form to set categories of reminder they would like to get. We would measure metrics such as thinking time spent by taking time stamps of these interactions at the start and the end. The FG runtime, would then include setting up queries to the historic database to store these interactions data for further FG analysis.

For the queries to be configured the component requires input are:

- Frequency (F): This parameter determines how frequent the FG analysis will be run. One could make sure during the testing stage, this is increased to gain rapid knowledge about the application behavior. After the testing stage has been analyzed enough for broader understanding the parameter could then be lowered down when running on the production server to keep resources consumption down.

- Horizon of analysis (H): This parameter determines which time period input data we would like to base our assumptions on for the FG Analysis. For instance, x hours means that only the data collected during the last x hours are considered.

  In our running application example, one could focus on a particular peak demand time as H specifically to understand resource consumption during this time and prioritize goals on how to handle it.

- Monitoring Intensity (MI): This parameter determines how intensive the data collection must be, in the range (0, 100). In this scenario, the value y indicates that y percentage of the total available samples were collected.

- Maximum Collection Window (MCW): FG Analysis requires the need to keep count of calls to the application methods inside the system. The MCW parameter enables the user to set maximum length of the accumulation time frame for which it is still keeping count. The FG tool will not collect more data after reading the MCW amount.

  This is useful as a noteworthy number of data collectors may bring about an undesired overhead. For our running example applicationt test environment, the MCW parameter can be set to a maximum. For production, the MCW can be set to low, which together with a little MI as well will ensure minimal overhead.

4.2.2 FG Monitoring History DB

The FG Runtime component filters the data received and relays it to the Monitoring History DB. This database guarantees that a backup copy is kept of all application interaction for future bug identifications or cross-feature development.

If in our notification example, one would like to go back a previous analysis to build on top of, they may choose to do by querying previously collected data in the Monitoring History DB.

4.2.3 *FG Design-Time*

The monitoring system retrieves, processes the data and then relays into the system for the estimation methods. Estimation routines are run to update the metrics of measurement in the QoS model by the design time component.

The types of metrics that can be collected by a monitoring platform depend on the system that the FG estimation tool is tracking. A list of the different parameters that can be observed for different kinds of systems is listed in Figure 2.

For our application example, the monitoring platform metrics required would be the total number of requests to estimate how many users are actually using our services. Throughput, queue length, response times and queue length (arrival time) would give us the ability to observe resource consumption. To understand more about end user think time, we would need parameters such as throughput, total number of requests, and mean number requests.

Population is the estimation of the total user clients. This is achieved by counting the number of active requests executed to each main application method.

Resource consumption is the estimation of the resource (CPU) consumption. The several alternative methods, based on different sets of information are:

- Utilization and throughput: A few methods utilize two quantities to understand the CPU utilization for each type of request. For the throughput, the number of calls to each application method within the QoS is tracked. They also configure an application-level data collector to enlist the calls to internal methods in the program.

- Response time and Queue Length: This refers to the number of requests waiting to be executed in each application method in each resource. To collect this information, they set up a monitoring platform that counts the number of call and reaction time. They at that point utilize statistical methods to assess the queue length at a particular time period.

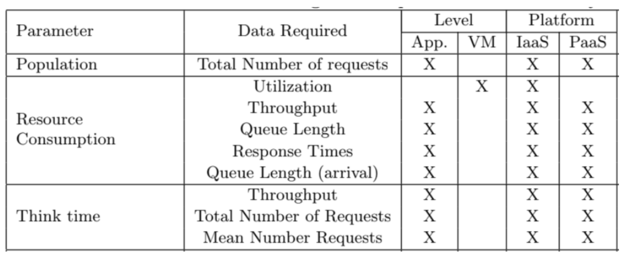
The consumption of different resources (CPUs, disks) is very difficult to access as typical server logs specifically track only generalized client requests. A way to get these metrics would be to configure in-depth monitoring tools. The FG run-time joined with the monitoring platform aims to measure these resources consumption by doing that.

In our notification application example, it is crucial for us to be able to capture resource consumption to try to fine tune different server resources dynamically at demand peaks to ensure better QoS for the notifications to reach end users with no delay. The FG run-time is where all this information will be captured.

4.3 Estimation techniques for FG Tool

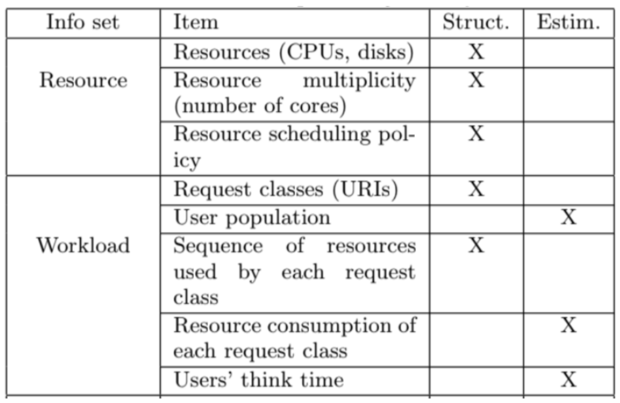
The FG Estimation techniques aim to measure the consumption experienced when multiple clients attempt to access hardware and software resources. These are parameters that will be part of the QoS model, could not be directly captured through monitoring platform and therefore have to be estimated.

The first of the two estimation techniques for the FG tool is based on looking at the queue for the number of requests that are still outstanding and have not been processed. The queue can be found by looking at server logs as we defined



Parameter	Data Required	Level		Platform	
		App.	VM	IaaS	PaaS
Population	Total Number of requests	X		X	X
Resource Consumption	Utilization		X	X	
	Throughput	X		X	X
	Queue Length	X		X	X
	Response Times	X		X	X
	Queue Length (arrival)	X		X	X
Think time	Throughput	X		X	X
	Total Number of Requests	X		X	X
	Mean Number Requests	X		X	X

Figure 2: Monitoring data required for the FG analysis



Info set	Item	Struct.	Estim.
Resource	Resources (CPUs, disks)	X	
	Resource multiplicity (number of cores)	X	
	Resource scheduling policy	X	
Workload	Request classes (URIs)	X	
	User population		X
	Sequence of resources used by each request class	X	
	Resource consumption of each request class		X
	Users' think time		X

Figure 3: Information Gained by QoS Model

them in the background. The count is taken from the logs in the server for requests on application methods and their specific time stamp for that method. The second estimation technique depends on response time, which can be acquired by dynamic testing or by injection of timers in the application code.

Three main sets of information were identified. They then further identified six more parameter sets as well. For the scope of this paper, we are not going to address the third information set: *environment stages* which aims to measure more details run-time metrics of PaaS and IaaS product models and so does not affect our running application example.

The parameters which are directly a result of FG analysis in the QoS model are: User population, resource consumption of each request class, users' think time. This is shown in the Figure 3.

The Qos model's resource information set is derived from the resource consumption parameter set from the monitoring platform. For our application, this could mean a report of how much of CPU power we might be utilizing on run time just for querying on the database, or how much CPU power we are spending on just sending texts after we have a result from running a query.

Other examples of parameters on the QoS model include the workload information set estimated from the population parameter. Data such as user's think time is based on estimations from the think time parameter of the monitoring platform data.

4.4 Summary of FG Tool

In this section, we talked about a tool to fill the gap between development and operations. The focus has been on the design of the tool, its architecture and how one would implement this tool. We did this by identifying how it would fit in with our working application example of a text notification software.

The first key objective of the FG tool uses analysis to estimate the QoS of a software application. Estimation routines are run on software repeatedly to figure out how close the software is to meeting its basic requirements. Different parameters can be tweaked to customize data collected from monitoring platform. After data has been collected we can run FG Analysis to provide the developer with reports, so they can get constant feedback on how their software is operating. This will allow the identification of services that need more attention.

5. DEVOPS RESULTS

In this Section, we will present the research of Perera, et al [3] which studied organizations that practice DevOps to evaluate if it impacts good software development metrics. In the following subsections, we are going to discuss the research methods of the study followed by their results and analysis.

5.1 Methods

Perera, et al [3] identified key variables that they believed had an effect on the success of software development. The three key concept they identified are reflected in Figure 5.

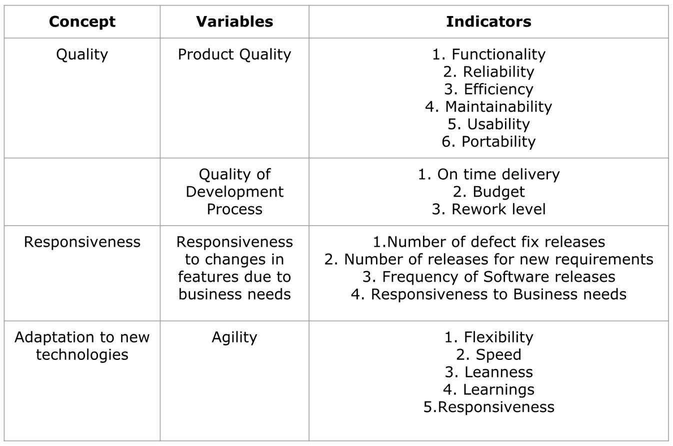
The quality concept had two separate variables: product quality and the quality of the development. Product quality referred to the quality of the software written and had indicators such as functionality, reliability. The quality of development referred to the process and not the functionality of the end product and included proper development practices such as documentation for maintenance etc. It was interesting to see how maintainability was also listed as an indicator, which reflects on the fact that good quality code also provides support for maintenance besides being functional and efficient. The portability indicator reflects on the ability to move systems with ease into new platforms.

The responsiveness concept aims to address the need for a company to quickly incorporate changes in the software product due to certain decision changes made by the business or because of bug detection. It is the measurement of how fast the development team implements changes after a new requirement is set. The indicators includes the number of new releases of the software that had the new requirements as requested by the business needs. Number of bug fixes or releases in general was chosen as a key indicator as well.

The last concept was the ability of a company to quickly adapt to new technologies. It measures the agility of the company with indicators such as the speed, leanness and flexibility.

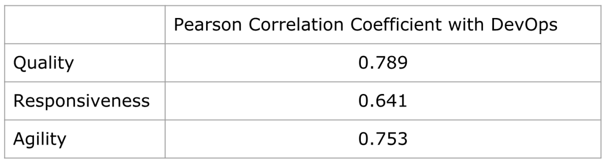
Interviews were then conducted with DevOps experts at 15 different organizations to go over a questionnaire based on DevOps metrics and key indicators. 150 organizations of different sizes all over Sri Lanka were sent then a questionnaire to gather data on the impact of DevOps in their organization. This included information on how much different DevOps practices they implemented affected the quality, responsiveness and the agility to adapt to new technologies.

According to Perera, et al [3], the key practices of DevOps are: culture, automation, monitoring and sharing.If we would like to tie their DevOps practices with our Five key aspects of DevOps from Section 2, we could map Culture and Sharing as DevOps: Collaboration, Measuring as a part of DevOps: Monitoring, Automation remaining constant and lastly Continuous Deployment which is built on top of DevOps: Continuous Integration.



Concept	Variables	Indicators
Quality	Product Quality	1. Functionality 2. Reliability 3. Efficiency 4. Maintainability 5. Usability 6. Portability
	Quality of Development Process	1. On time delivery 2. Budget 3. Rework level
Responsiveness	Responsiveness to changes in features due to business needs	1.Number of defect fix releases 2. Number of releases for new requirements 3. Frequency of Software releases 4. Responsiveness to Business needs
Adaptation to new technologies	Agility	1. Flexibility 2. Speed 3. Leanness 4. Learnings 5.Responsiveness

Figure 4: Variables with corresponding indicators



	Pearson Correlation Coefficient with DevOps
Quality	0.789
Responsiveness	0.641
Agility	0.753

Figure 5: Correlation

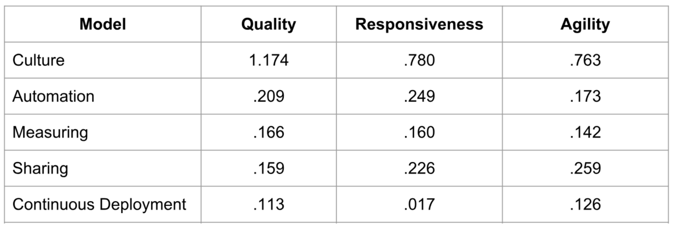
Afterward, three main hypothesis were derived to evaluate if DevOps had an impact on these key success concepts.

- Hypothesis 1: Implementation of DevOps would positively impact the quality of software, i.e., there is a positive relationship between the implementation of DevOps and the quality of the software being developed.
- Hypothesis 2: Implementation of DevOps would positively impact the responsiveness to business needs: there is a positive relationship between DevOps and appropriate responsiveness to business needs.
- Hypothesis 3: DevOps would positively impact the adaptation to new technologies: there is a positive relationship between DevOps and the ability to adapt to new technologies.

5.2 Analysis

After the results from the questionnaire were collected and combined, statistical analysis was performed to see if there is a relationship between the usage of DevOps and key success metrics.

It should be noted that these results were self reported and so might have validity problems. The answers the developers put down in the questionnaire may be exaggerated



Model	Quality	Responsiveness	Agility
Culture	1.174	.780	.763
Automation	.209	.249	.173
Measuring	.166	.160	.142
Sharing	.159	.226	.259
Continuous Deployment	.113	.017	.126

Figure 6: Linear Regression Model Coefficients

and various biases may affect the results. They might not remember everything with details to input data that truly represent their DevOps practices or they might be told to answer in a specific way by their supervisor. There is no way accurate way to validate them and thus draw research observations and relationships with full certainty. But because they had such a large number of participants from different companies, we can assume that the data collected is not completely biased and invalid, and so study them to understand the nature of DevOps in these companies.

The Pearson product moment correlation coefficient was calculated between quality of software and DevOps, responsiveness to business needs and DevOps and adaptation to new technologies and DevOps. The Pearson correlation coefficient has a range from +1 to -1. A value of 0 indicates that there is no specific relationship between the two variables that we are analyzing. If the value is greater than zero than that would depict a positive linear relations. One could think of it as a directly proportional relationship i.e. as the value of one increases it so does the other. The larger the value and closer to +1 the more dependant they are. If the value is less than zero than that would depict a negative linear relationship. One could think of it as a indirectly proportional relationship i.e. as the value of one increases the other decreases.

The results of these statistical analysis is shown in the Figure 5. The Pearson correlation coefficient in all of our cases was positive. These depicts that most of these companies according to their response on the questionnaire believed that the usage of DevOps had a positive impact in the quality, the responsiveness and the adaptation to new technologies. The highest coefficient was with the quality of software. Agility to adapt to new technologies was the second closest and was large enough to deduce that DevOps also had a positive effect on this success metric. The responsiveness to business needs of a company also had a positive relationship with the practice of DevOps but not as much as the other two.

After all of the three hypothesis listed were evaluated to discover there was a positive relationship, Perera, et al [3] did additional statistical analysis to examine how much each DevOps practice impacted each of the success concepts. A linear regression model was setup to achieve this through predictive analysis. It aimed to examine how the different DevOps practice variables impact the dependant success concept. By doing this, we could now rank which practices in particular are significant to ensure quality, responsiveness and agility. The larger the coefficient in the linear regression equation, the more of an impact in had. This can be seen in the Figure 6.

This data suggest that the quality of software was most impacted by the cultural practices of DevOps, followed by automation, measuring, sharing and finally continuous deployment. Responsiveness to business needs was most impacted by culture followed by automation, and then sharing, measuring and finally continuous deployment. The adaptation to business needs was mostly impacted by culture, followed by sharing, and then automation, measuring and finally continuous deployment.

Overall, the cultural aspect of DevOps had the most impact on quality, with continuous deployment having the least impact.

These provide key insight as one could observe the problem domain and prioritize certain DevOps accordingly. If a particular company is having a hard time ensuring good quality of software they might like to implement DevOps since it has shown results to improve that more than it improves responsiveness. Focus should be put to implement the cultural practices first, followed by Automation as they have the higest impact on the success concepts. Continuous Deployment should be the least significant practice they implement after the other ones have been set forth.

The sharing of knowledge has shown to have a higher impact on the ability to adapt to newer technologies compared to its impact on the other two key successes.

Continuous Deployment has the least impact on responsiveness to business needs which is surprising as the indicator of this metrics was defined as the number of fix releases. This is very contradictory. If the total number of releases is closely related to responsiveness to business needs, continuous deployment should have a major impact on this as it ensures continuous delivery of new code into the development pipeline and then in production. This study reflected that it is not so and developers should prioritize automation, culture and sharing more than continuous deployment to ensure better responsiveness.

6. CONCLUSIONS

In this paper we learnt about what DevOps constitutes and the key practices of DevOps. We defined them as Collaboration, Automation, Continuous Integration, Continuous Testing and Continuous Monitoring. We learned about how each of these practices aim to solve challenges faced in the development cycle and operations cycles which relate to maintenance after deployment. We looked at a DevOps tools that incorporated two of these key aspects. We leart about the architecture of this tool and how to implement it followed by a study that evaluated the usage of DevOps and its effect on success in a software company.

7. ACKNOWLEDGMENTS

I would like to thank my professors: Nic Mcphee and Elena Machkasova for their invaluable feedback and guidance. I would also like to thank Emma Sax for her feedback.

8. REFERENCES

[1] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer. What is DevOps?: A systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*, XP '16 Workshops, pages 12:1–12:11, New York, NY, USA, 2016. ACM.

[2] T. Masombuka and E. Mnkandla. A DevOps collaboration culture acceptance model. In *Proceedings of the Annual Conference of the South African Institute*

of Computer Scientists and Information Technologists, SAICSIT '18, pages 279–285, New York, NY, USA, 2018. ACM.

[3] P. Perera, M. Bandara, and I. Perera. Evaluating the impact of DevOps practice in Sri Lankan software development organizations. In *2016 Sixteenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 281–287. IEEE, 2016.

[4] J. F. Perez, W. Wang, and G. Casale. Towards a DevOps approach for software quality engineering. In *Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development*, WOSP '15, pages 5–10, New York, NY, USA, 2015. ACM.

[5] J. Roche. Adopting DevOps practices in quality assurance. *Commun. ACM*, 56(11):38–43, 2013.

[6] D. Saff and M. D. Ernst. An experimental evaluation of continuous testing during development. *SIGSOFT Softw. Eng. Notes*, 29(4):76–85, July 2004.

[7] M. Shahin, M. A. Babar, and L. Zhu. The intersection of continuous deployment and architecting process: practitioners' perspectives. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 44. ACM, 2016.

[8] M. Walls. *Building a DevOps culture.* " O'Reilly Media, Inc.", 2013.