HÖGSKOLAN VÄST
School of Business, Economics and IT
Division of Informatics

FSU600 - Full Stack Developer | Assignment 5 : In-depth study | Fall 2021

Enhancement of Full stack application through DevOps

Kanniga Lakshmi Jagadeesan | kanniga-lakshmi.jagadeesan@student.hv.se

Supervisor : Abdulghafour Mohammad

ABSTRACT

Computers have revolutionized the world over a few decades in almost all sectors.So,is software engineering which led to emerging changes within a short period of time in software development applications.A typical *Full stack developer* skill set combines frontend and backend skills ideally required to support a running application or a website.But the scenario has changed now with a lot of employers expecting more than the classic skillset of a Full stack developer.

This paper focuses on a broad study of *DevOps* and how it can add value to an existing skill set of a full stack developer.This paper also provides some insight about the history and evolution of DevOps practise along with the theory from literature reviews and research studies.Moreover, the benefits of adopting DevOps and comparison between a traditional architecture and Agile culture is presented by outlining the transition that it inculcates during adoption and implementation of the same.This is very important as it enhances the move from *Project based mindset* to *Product based mindset.*

Lastly, it is concluded that there is a great potential for employers as well as for employees to bring a change in an organization that promotes growth and *'Customer satisfaction'* is prioritized.

Keywords

DevOps, Continuous Integration(CI), Continuous Delivery(CD), Software development, Full Stack Developer

1. INTRODUCTION

The term DevOps is a portmanteau of *'Development'* and *'Operations'* in a software development environment.DevOps refers to a set of practises or cultural values which involves *Continuous Integration(CI)* and *Continuous Delivery(CD)* across the entire software development team by following the same protocol.By this way of adopting DevOps, there is an increase in the pace of Software Development process and improved software quality.Due to it's increasing demand and emerging interest among developers in the field, which aims to benefit not only for an individual but for the entire team, a typical Full stack developer is now expected to adopt and possess it as one of the skillset throughout their career.

1.1 Aims

The goal of this study is to provide an empirical framework of embedding DevOps during development of a Full Stack application and how it adds value to the project.Hence, with this research,the author wanted to address the question *"How Devops benefits for a full stack developer and the need for it's adoption."*

1.2 Limitations

DevOps is a well studied phenomenon.Yet, there is a lot that has to be understood,implemented and executed on a full stack application.Although there is much information available in this topic, limitations for this study has been formulated.First of all,a detailed description of DevOps culture, how it works, and it's impact during software development life-cycle is presented since there are many reviews and sources including this information already available. Further, only successful and limited case studies that has already been included in the sources has been considered.This is partly to enable a sufficient analysis, partly since many state-of-the-art digital solutions focus on developmental stages of a product.Thus,this study is focused more on adoption,implementation of DevOps culture in an agile environment irrespective of region,culture pertaining to a specific development environment and any agile framework.

2. STUDY SELECTION PROCESS

In order to fulfill the aims presented in section 1.1 a literature study was performed. Initially, a general search through various scientific databases, namely *IEEE, Science Direct, Google Scholar, Research gate, AIS Electronic Library (AISeL)* and *University West library* was made. In this phase, the used search terms were some combination of *'DevOps'* and at least one of the following keywords: *'Software Development'*, *'DevOps adoption'*, *'Full stack developer'* and/or *'Current IT trends'*. This resulted in a large amount of articles, which were critically analyzed to enable a careful selection of studies to investigate further.All The selected texts provided an introductory understanding of the DevOps and the current state-of-the-art digital solutions in this field.

Moreover, to get a broader understanding of the topic and to further fulfill aim from section 1.1 the knowledge gained from the above procedure was used to make a more specific search through the previously mentioned databases. To further get a deeper understanding of the subject online courses such as *'Pluralsight'* and *'Udemy'* were followed.In this phase, keywords such as *Continuous Integration'*, *Continuous Delivery'*, *'History of Git'*, *'Role of DevOps in IT'* were used.Since DevOps has been a fast developing technology and its' widespread reach among the audience during the last few decades, only more recent (later than 2017) studies were included in the selection of potentially interesting articles. Following the limitation regarding only presenting DevOps culture in an agile environment, as presented in section 1.2,only qualitative studies describing skill-set required for DevOps, adoption and implementation of Devops in the real world were included.

3. HISTORY OF DEVOPS

The origin of DevOps dates back to 1957 with the incubation of first program written in *FORTRAN* thereby giving way for first developer jobs.Later in the year 1967, with the launch of *ARPANET*[1],operation centres were created which gave rise to a network of engineering jobs.Going forward by few decades,to ensure smooth operations on the client side,Ben Treynor hired by Google in 2003, lead the first set of '*site reliability engineers*', who were able to run production environment and development environments separately.This team was highly responsible to maintain high uptime(99.7%) while working along with the developers.

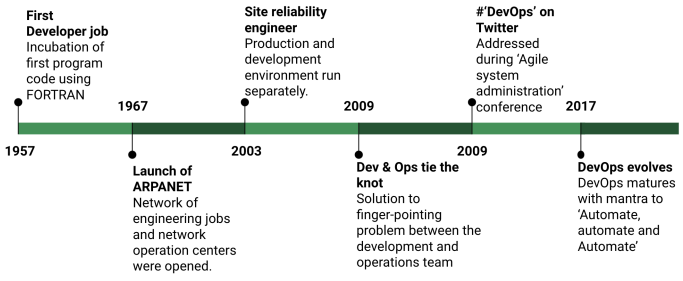

Figure 1: Timeline showing evolution of DevOps

[1]This network became the basis for Internet

Few years down the lane,around 2009, Flickr made an attempt to combine *'Dev'* and *'Ops'* to solve the finger-pointing problem between the developers and operations team.John Allspaw and Paul Hammond proposed a solution at O'Reilly Velocity Conference, where the two arrived at an methodology to integrate "Dev"(Development) and "Ops" (Operations) into an automated infrastructure.IN 2009,Patrick Debois,organized a conference on "agile system administration", where he used Twitter as an active advertising platform for his conference with the hashtag *'DevOps'* thus giving birth to the term and gained it's popularity from social media.As AI gained it's momentum from 2017 permeating almost in every aspects,DevOps can be seen as an evolving phenomenon and matures day-by-day with the mantra to "*automate, automate and automate*".The author agrees that DevOps can be viewed as a global movement and continues to evolve with many forward-thinking computer engineers across the globe contribute their ideas on any open-source platform about testing, automation, adopting DevOps culture to smoothen the friction between dev and ops.

Figure 1 shows the timeline of evolution of DevOps culture.(Online source,2017)

4. THEORY IN ADOPTING DEVOPS

One interesting subject of matter that served as an inspiration to adoption of DevOps is the *Theory of Constraints* explained further(Pluralsight,2021).Figure 2 shows a clear view of how throughput of the system is limited by numerous constraints.Optimizing the product before/after delivery results in consequences which in-turn is a valuable factor to evaluate the quality,efficiency of the product.

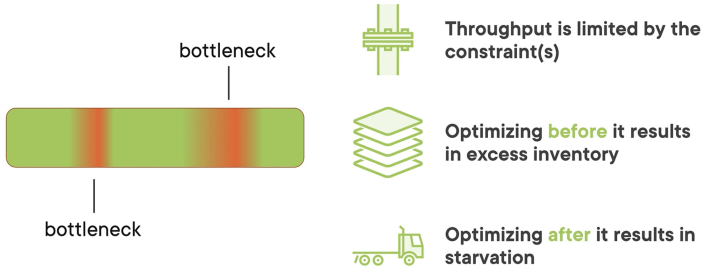

Figure 2: Theory Of Constraints

Imagine a structure of a pipe that shrinks to a small pinhole at one area, which controls the entire flow.The same constraint can be compared as a bottleneck to the product delivery which determines the whole *throughput* of the system.The code needs to be *optimized* before the constraint, which can be done by creation of excess team members, additional processes which in turn leads to overburden of people,physical servers and machines required for sustainable and faster product delivery.These factors act as limiters for throughput and it is not a good practise to blame a certain team/team member for disrupting the whole flow.But,these constraints can be easily alleviated by few policies, technologies or it could be a team member also.To simply frame it in a single term, this gave way to adoption of DevOps practise before product delivery, that doesn't require much time but also improves the throughput of the entire system.

In the following section the author argues with two case studies which mainly focused on the successful adopted model and implementation of DevOps within organizations of different domains, structure and strength of the employees.Both of the case studies involved using *Grounded Theory*,in which the obtained information was compared with the existing knowledge throughout their research.Both the case studies involved a focus group who were given a set of questions to understand the employees perception about adoption of DevOps.Surprisingly,all the organizations involved in the case study gave a positive feedback about their experiences and that very minor errors were faced.It is to be noted that,all the participants had an adequate understanding of Devops which helped to assist other team members/organizations towards the migration.(Luz, Pinto& Bonifácio, 2019; Erich et al,2017).

General set of questions that were used during the case studies is listed below.

- *Why did the organization decide to implement DevOps?*
- *How was DevOps adopted in your company?*
- *What were the results of adopting DevOps?*
- *What problems are encountered when implementing DevOps?*
- *Brief on the terms between development and operations team?*

5. DEVOPS : THE BIG PICTURE

This section gives a deeper understanding of what processes of DevOps are involved in a development environment,typically a full-stack application.This is explained by giving a comparative view between a traditional software development[2] and Agile development environments.In addition to this, the required skill-set for an individual/team to possess a DevOps mindset along with the cultural and organizational changes that DevOps brings in and how it forms a part of skill-set for a full-stack developer is analyzed.

5.1 Comparison between previous software development and current development

Earlier days, in a non-agile software development model as shown in figure 3, any programmer in a team did not have access to a central repository base.So, the changes made by a programmer is made available to the source code base only after a separate team responsible to merge all the changes, preferably all the developmental tasks has been completed.This way, each programmer is not aware of the other changes, as a result this results in code conflict and results in erroneous product during each stage of QA testing.This in-turn leads to a delay in product delivery,redundant code base with a lot of bugs, additional resource and equipments.

[2]Non-agile models like waterfall,V-model,..

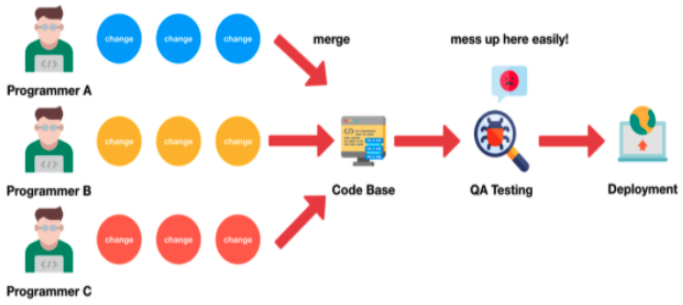

Figure 3: Developmental workflow of Waterfall model

Introduction of agile frameworks like *Scrum, Kanban, Extreme Programming* proved a major success in product development organizations.Here, the product delivery is done in short and frequent cycles.Once the product requirements and design has been planned, developers start to work on their code and make changes by making use of a centralized repository.Thanks to *'Linus Torvalds'* who created *'Git'*, a version control management system that attained it's fame among the developers and still evolving.

Figure 4 shows the entire workflow of a git repository. Whenever a programmer wishes to change the code, the latest version needs to be first *'pulled'* from the *'master'*,which is the source base for the entire team.A new *'branch'* is created to make new changes,*'Unit testing'* is done for the respective modules and then all the changes are 'committed' within the branch, the changes are compared and then pushed to branch.A new *'Pull request'* is created where the team members are notified.The code is available for *'Review'* and comments are addressed, issues are resolved.Once, all the changes has been *'Approved'* and there is no code conflict or breakage in the pipeline, the code is *'merged'* to the master and ready to be deployed.It is to be noted that the whole process is transparent and every team member is aware of modifications to the code base.This fosters team co-ordination,code duplication is avoided, no separate team is involved to handle the version control, faster and efficient product delivery.

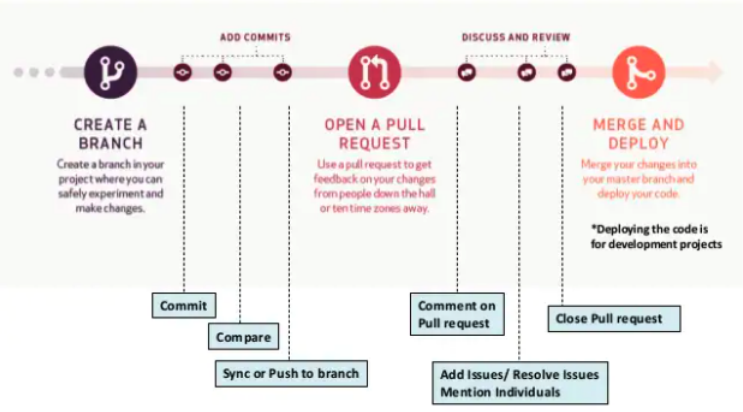

Figure 4: How Git works

5.2 Implementation of DevOps in real time

Continuous Integration(CI) and Continuous Delivery(CD) are the two pillars that support DevOps with the development and deployment process respectively.Obviously, these two processes involve a lot of steps and very different in their own way,but needs to be synced and successfully implemented so as to be able to react to the needs of the customer.The path to get through is proper implementation of CI and CD which is explained in the following sections.

5.2.1 Implementation of CI

Frequent integration of code into a central repository mostly by developers can be simply defined as Continuous Integration.Developers check-in their code into specific source control branches for the software modules that they are working on. This helps them to separate their work. To do continuous integration, the developers need to change how they work. They should now all use a central source control repository. This will require them to integrate code themselves, meaning that each team and each developer has to do a bit more work to make sure that their code works with the rest of the code.The developers that create the code know it best, so they know how it should fit with the rest and what could go wrong.To check if the code actually fits together and works it should be compiled regularly. Even every time when anybody checks in code,it should be compiled on a dedicated build server. This way, the configuration for building things is centralized and is therefore more controllable.

All of the developers should also have access to the results of the build, so that they can see if their code compiled or if there is a problem that they need to fix. The build should not only include compilation of code, but also automated tests to make sure that we have the best result possible. These tests would run against the compiled version of the code, possibly against a temporary, local deployment of the code. These tests can be of any kind, unit tests to check if code yields the expected results, automated UI tests to check if the UI responds as it should, security tests to see if the application is secure, and so on. The whole build process, compilation, and test should be quick. This way the results of code check-in can be communicated to the developers quickly, who can then fix whatever needs fixing so that the code base stays healthy and everybody is able to continue work.

The process of compiling and testing should be automated. It should run very regularly, possibly every time when somebody checks in code. To make this work, developers might need to change the way they work. Continuous integration can be successfully adopted if an organization adheres to these principles. There needs to be a single place where all of the code lives, this is usually in source control, in a central repository that can be called as the main line. All developers check-in their code at least once a day, maybe more, but at least once a day. This way each day it is assure that everything still compiles and all of the tests still succeed. Developers should take extra care to make sure that what they check-in compiles, as it now influences a lot of people immediately. The build process needs to be automated. Developers and the rest of the organization should prioritize fixing a failing build over building new functionality. There is no point in building anything new if the code can't be released anyway.This way it can be executed many times and developers get feedback quickly. Every time a developer checks in code the build runs. This is the principle for the strictest definition of continuous integration. You could also buffer check-ins if you have a lot of them and run the build every time you have a bunch of check-ins,but at least multiple times every hour.Automated tests should then be part of the automated build process.Everybody should be able to see if the build succeeds and if not, where it went wrong.This includes the things that other people are working on. All the source code, all of the builds and test results, and how the build works. Better communication translates into better results.

5.2.2 Implementation of CD

The practise of making the software that can be released to production at any time is Continuous Delivery.The point is to get to a state from which the code can be deployed to production always.Previously,the operations team received a release and installation instructions to deploy onto different environments.They would only receive this at the end of each iteration.This works differently now with the concept of continuous integration.All of the developers check-in their code into a central repository that gets compiled and tested automatically at least once every day.This results in a release every day, without installation instructions.

In continuous delivery,the latest release is fed into a release pipeline.This pipeline is an automated process,that is executed on a server.This process takes the release package and installs it with steps similar as those that were in the installation instructions. These steps are things like copy files to a certain location, reset the IAS server,and so on. All of these steps together deploy the release to whatever environment we want, like dev, test, or production.Instead of the release package for the application,the infrastructure for an environment like production as code is needed, or IaC, infrastructure as code. This is done by scripting the complete creation or update of an environment with things like PowerShell and JSON script.This is a practice that is becoming more and more popular.

With a stable environment,infrastructure as code can be deployed or updated it to a similar process as the application.The release pipeline would execute all of the scripts to create or update an environment.By using this, maintaining and environment becomes much more easy and all environments will be exactly the same, as they are created by exactly the same scripts.This is not a requirement for continuous delivery, but something that becomes possible when there is continuous delivery.

5.2.3 A note on Continuous Deployment

Continuous delivery is often confused with continuous deployment, and vice versa. However, these are two different concepts.In continuous delivery,there is always a choice to deploy the source code to production at any time.In continuous deployment,software is automatically deployed to production all the time, continuously.Deployments happen as soon as new code checks in.

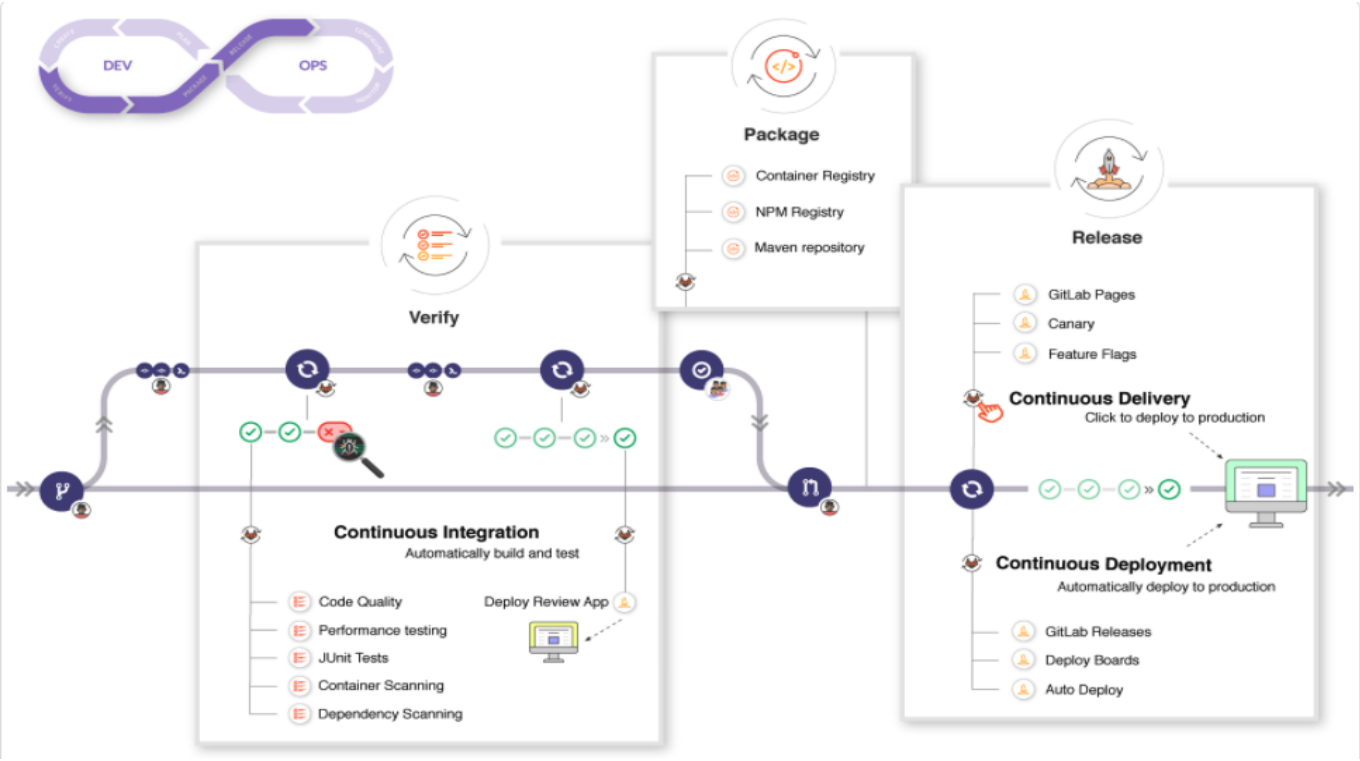

Figure 5: Workflow model of Continuous Integration,Continuous Delivery and Continuous Deployment

5.2.4 Successful DevOps practise

Continuous Integration needs to be in place to be able to do continuous delivery. If CI is not in place,delivery can be automated, but the code will never be in a state where it could be deployed to production at any time.Development and operations should work together, not against each other.In a traditional software development environment,they were incentivized to do different things, and thus, work against each other.This needs to change.This is not a requirement,but a very nice to have.When infrastructure as code exists with automation and proper maintenance of the same,there will be a consistent environment,causing less errors and costing less effort to maintain them.The release process should be automated.This means that the installation steps should be examined and automate these into scripts,like PowerShell scripts and other scripts that can be used by the release pipeline.

Deploying a release to production should be part of *'Definition of Done'*.A feature isn't done until it is successfully running in production.At the same time,there needs to be cautionary scenario where release doesn't happen automatically, but should be on demand.As with continuous integration, *communication* is key.Everyone should be able to see the status of the process.Everyone should be able to see what succeeded and what failed,and everyone should have access to the latest release that is running on dev, test,or somewhere else.

5.3 DevOps and Full Stack Development skills complement each other

This section gives a clear insight of how DevOps practise and Full stack development skills complement each other mutually in a development environment.The author shows that case studies conducted in two research papers is good enough to validate the above statement.Both of the case studies involved participants from different role,designation and age within various IT industries, also from academia.Along with other skills, be it technical or non-technical, it is worthy to mention that every participant mentioned Full Stack development skills as a necessary skill to adopt DevOps practise.(Taivalsaari et al,2021 ;Wiedemann et al,2018)

It is not sufficient to have either front-end or back-end knowledge.A comprehensive knowledge of multi-tier architecture, platforms is necessary to resolve problems at the earliest.In a traditional environment, development members with one specialized area were highly focused and given importance while recruitment.Now, the scenario has changed.One of the important criteria an employer looks while hiring a Full Stack Developer is to possess knowledge of Git CI/CD pipelines to enable build,automatic testing and deployment of latest version of the product to staging and production environments.By acquiring these skills, it can be argued that appropriate testing techniques across front-end and back-end is required.

6. DISCUSSION

This section discusses about the benefits of DevOps while it is practised or yet to be adopted pertaining to an individual or as a team.It also gives the reader a clear picture of what DevOps is not.

6.1 Benefits of DevOps

1. **Faster Development lifecycle** - One of the biggest advantage of DevOps adoption is incorporation of 'Agile culture'.There are shorter development cycles accompanied with accelerated deployment cycles leading to faster delivery.This ensures that business process is followed in an optimal direction directly contributing to success and increased ROI.

2. **Improved collaboration and interoperability among teams and team members** - DevOps contributes to better team communication and collaboration,which obviously results in greater efficiency and promotes healthy work culture.Every team member is equally responsible for contributing to the product delivery.Because of the improved interaction between different teams,DevOps promotes interoperability, transparent system that directly sync with the business goal.

3. **High Quality Software development** - DevOps facilitates early detection of defects which is one of the biggest threat to an application.Feedback is shared between teams and team members subsequently resulting in bug resolution at a significantly faster rate.The automated build, test and deploy process helps to improve the overall quality of software.User experience is enhanced by implementing more stringent quality control measures.

4. **Streamlined Business process** - DevOps results in quick incorporation of code changes resulting in low downtime thereby boosting the productivity of stakeholders involved in application development.The traditional waterfall method involved hand-offs of the software from one team to another.This gives way to an immense reduction in repetitive tasks and fosters creativity and innovation.

6.2 What is Not DevOps

DevOps is a way of working,DevOps cannot be a separate team like QA team or a production team.DevOps is a combined approach of a lot of practises, tools, culture and technologies.Devops is not only an IT effort.DevOps cannot be counted as an IT initiative to automate things.DevOps is not just a snap easy thing to adopt and make the team adapt easily.It has a great impact on the entire line of business.

7. CONCLUSION

This paper contributes to the understanding of DevOps and how it contributes as one of the required skillset for rapidly growing full stack developer.All the observations based on the concept of competencies and skills has been identified with the help of a multiperspective research approach.The findings show the importance of skills and skill categories for an efficient and successful Full Stack developer as an enhancement to career.Since there exists less common theory and history about DevOps, the author followed the path of prior literature to further investigate this phenomenon.This research enriches the understanding of several practises that were less defined in prior research and provided a connectivity between full stack application and embedding DevOps culture along with development,which have not been widely discussed in prior literature.

Moreover, by analyzing and prioritizing the observations, this study highlights that a combination of strong development, operations, and full stack development is necessary to successfully work within a development team.Existing research mainly focuses on either DevOps as a separate team or only Full stack Development.The findings deliver a strong foundation for further research opportunities about Full Stack with DevOps concept, possible relationships among the skill categories, several skills, and the effects of performance and outcome as a whole.This will effectively force to broaden the expectations over a wide spectrum of areas for study and research.

Although, the time to transition from legacy systems to current development involves a lot of things like people's mindset, cultural changes, financial outcomes, tools and technologies involved, it is worth to welcome a *'Change'* provided it has some value though and attain the ultimate goal of every organization, a *'Happy Customer'*.

APPENDIX

A. LIST OF ABBREVIATIONS

AI	Artificial Intelligence
CD	Continuous Delivery
CI	Continuous Integration
IAC	Infrastructure As Code
IAS	Internet Authentication Service
IT	Information Technology
JSON	Java Script Object Notation
QA	Quality Assurance
ROI	Return On Investment
UI	User Interface

B. REFERENCES

- ☐ Luz, Welder Pinto, Gustavo Bonifacio, Rodrigo. (2019). *Adopting DevOps in the Real World: A Theory, a Model, and a Case Study.* Journal of Systems and Software. 157. 10.1016/j.jss.2019.07.083.

- ☐ Wiedemann, Anna and Wiesche, Manuel, *"ARE YOU READY FOR DEVOPS? REQUIRED SKILL SET FOR DEVOPS TEAMS"* (2018). Research Papers. 123. https://aisel.aisnet.org/ecis2018_rp/123

- ☐ Taivalsaari A., Mikkonen T., Pautasso C., Systä K. (2021) *Full Stack Is Not What It Used to Be.* In: Brambilla M., Chbeir R., Frasincar F., Manolescu I. (eds) Web Engineering. ICWE 2021. Lecture Notes in Computer Science, vol 12706. Springer, Cham. https://doi.org/10.1007/978-3-030-74296-6_28

- ☐ https://www.botmetric.com/blog/what-is-devops [Accessed:24-Nov-2021]

- ☐ https://app.pluralsight.com/library/courses/continuous-integration-delivery-big-picture [Accessed:26-Nov-2021]

- ☐ https://app.pluralsight.com/library/courses/devops-big-picture-2021 [Accessed:29-Nov-2021]