Performance Assessment of Traditional Software Development Methodologies and DevOps Automation Culture

Poonam Narang
Department of Computer Science and Applications
Maharshi Dayanand University
Rohtak, Haryana, India
poonam.mehta20@gmail.com

Pooja Mittal
Department of Computer Science and Applications
Maharshi Dayanand University
Rohtak, Haryana, India
mpoojamdu@gmail.com

Received: 10 September 2022 | Revised: 25 September 2022 | Accepted: 26 September 2022

Abstract-Successful implementations of Software Development Methodologies significantly improve software efficiency, collaboration and security. Most companies are moving away from traditional development methodologies towards DevOps for faster and better software delivery. DevOps, which is a primary need of the IT industry, brings development and operation teams together to overcome communication gaps responsible for software failures. It relies on different sets of automation tools to robotize the tasks of software development from continuous integration, to testing, delivery, and deployment. The existence of several automation tools in each development phase raises the need for an integrated set of tools to reduce development time. For this purpose, we used the DevOps-based hybrid model Integrated Tool Chain (ITC), along with three sample java-based projects or code repositories to quantify the results. This paper evaluates and compares measurement metrics of java projects using traditional development methodologies and DevOps, and the results are shown in tabular and graphical format. The latest Google and Stack Overflow Trends have also been included to retrieve the best performer development methodology. This comparative and evaluative performance analysis will be beneficial to young researchers that study the metrics of software development, while also they will be introduced to the automotive environment of DevOps, the latest emerging buzzword in software development.

Keywords-automation; automation tools; DevOps; integrated tool chain; software development

I. INTRODUCTION

Software engineering is a branch of software design through the Software Development Life Cycle (SDLC), which consists of several development phases, from gathering requirements and thorough analysis, to design, coding, testing, deployment/delivery, and maintenance. Software development has introduced various methodologies from traditional and agile to the most recent DevOps culture. Traditional and agile development methods include models like waterfall, iterative, evolutionary, spiral, prototype, V Model, Scrum, XP RAD, etc. The availability of multiple models necessitates their careful selection for successful and quality software delivery. As development models under traditional methodologies struggled with many defects responsible for late software delivery and budget overruns, fast and successful development was one of the major challenges for traditional development models. Agile methods provided better solutions for quick releases with small sprint sizes as they introduced flexibility into their methods, but still lacked continuity in development and operations, which is where DevOps comes in. DevOps, the industry's common term for development, is based on five software development continuums, i.e. continuous integration, continuous testing, continuous delivery, continuous development, and continuous monitoring. The continuity in the environment of DevOps is achieved through different sets of automation tools employed at each and every phase of software development. DevOps works on the principles of an infinite continuous cycle that starts from continuous integration of code repositories and continues until the monitoring stage of the software development to deliver quality successfully.

The current research quantitatively evaluates the performance of different software development methodologies, including DevOps, based on various software performance metrics. For the application of these metrics and to obtain quantitative results, three sample code repositories or java-based projects have been taken into consideration. This paper also considers our proposed [1] and implemented [2] DevOps-based hybrid model of automation Integrated Tool Chain (ITC). The proposed and implemented ITC model demands automation tools in each phase of software development. As there are various automation tools in each phase of software development, prior selection of automation tools will not only reduce development time but also ensure delivery within budget and quality. This study examines the proposed and verified ITC [2] for DevOps implementation. The result of traditional methodologies and DevOps performance evaluation based on various software metrics validate the quality of the delivered product and also helps in recovering the best performing software development methodology. The outcome of our study will be useful for young researchers/students to understand the course of action of DevOps and its automation

Corresponding author: Poonam Narang

tools. It will also be important for software developers to choose the best execution tools from the various automation tool sets available.

SDLC contains detailed edits of the various steps or processes necessary to configure the software. It covers many methodologies such as traditional and agile methodologies and now DevOps, while these methodologies cover various models. DevOps, on the other hand, includes a set of different automation tools to achieve the goal of competitiveness in the era of digital transformation and to increase the speed of adaptation or reaction to changing software requirements [3]. DevOps, with its principles of continuity, helps companies survive in this competitive world. The main focus of the current paper is to describe and compare DevOps with traditional and agile methodologies

II. TRADITIONAL METHODOLOGIES

Traditional methodologies specifically include the waterfall model, first introduced in the 1970s. The cascade model is explained stage-by-stage [4]. Through these stages, including iterative approaches, the author introduced the development of large-scale systems. The step-by-step approach of traditional methodologies allows large-scale projects to be handled and developed more successfully. But on the other hand, traditional development methods are rigid and also fail to respond to aggressive or frequently changing customer requests [5]. Compared to the traditional methodology, the agile development methodology provides a set of practices that allow rapid adaptation to changing customer needs. In terms of research gaps, the authors discussed the failures of traditional models very well, but did not propose any solution to overcome the major flaws of traditional development methodologies. Current research included DevOps culture, with its validation analysis of out-performance, as a result of comparison with other development methodologies.

A. Agile Development Methods

Although agile methods were created to address any flaws or limitations of traditional development methodologies, they are also confirmed to be effective for large-scale distributed projects [5]. Agile approaches are drivers in huge and extremely large-scale development [6]. Therefore, the agile approaches encourage greater communication, continuous integration, and fast delivery of products using an incremental and iterative approach, but they also have several disadvantages, such as lack of planning, lack of documentation, lack of predictability, etc. [7.] To discover genuine constraints of agile approaches outside the existing literature review, the authors in [7] also performed an online survey. There are still research gaps in terms of solutions to the problems faced by flexible methods. Basic research talks about DevOps development culture to overcome many of the difficulties of software development.

B. DevOps Development Culture

DevOps proposes a solution to a variety of development and delivery challenges, including the quality of the generated product. In their research on DevOps, authors in [8] examined the effects of DevOps features on software quality. Increasing the speed, frequency of development and product quality is the main goal of DevOps. To minimize differences or fill in all kinds of communication gaps between dev and ops teams, DevOps promises to deliver successful projects with shorter deployment or sprint times [8]. To deliver or deploy high-quality products, authors in [9] agree that development and operations teams should cooperate well [9]. Although numerous studies support the use of DevOps, others are opposed to it and discuss the lack of quantification of performance and quality measures [10]. Authors in [11] found that DevOps helps businesses to achieve their goals and also increases their speed in reacting to changes [11]. The literature also confirms the existence and successful implementation of various DevOps models in practice [12]. Similarly, many papers, including [11, 16], accept the emerging paradigm as a response to the growing awareness of the existing many types of communication or collaboration and cultural gaps between dev and ops team functions. Again, there are research gaps in terms of quantified quality outcomes provided by DevOps.

C. Motivation

The motivation behind this research is to provide quantified results of DevOps quality and on-time delivery. To this end, the current research examines the proposed [1] and implemented [2] model of hybrid integrated automation tools based on DevOps. Study is also conducted from different perspectives of model building [13, 15] to the application of cloud computing [15] or the adoption of recent approaches for development industries. Based on the existing literature and our tool model, this paper performs an evaluation of different types of parameters or software metrics for quality measurement. This research has taken three sample software projects developed in JDK environment to measure DevOps performance and quality. Lines Of Count (LOC), number of different components/modules, development time, and risk consideration are considered as evaluation metrics.

III. DEVOPS BASED HYBRID MODEL OF INTEGRATED TOOL CHAIN

For the performance evaluation of DevOps, we propose [1] and apply [2] a DevOps-based hybrid ITC model as depicted in Figure 8 of [1]. For the hybrid model, different tools are proposed from which only one automation tool is selected as the best performer tool based on several performance evaluators. The current research work considers the same set of automation tools [1] for the implementation of sample java projects, mentioned in Table I, using DevOps development and deployment culture.

IV. RESEARCH DESIGN AND DATA SET

The current research compares traditional and agile methodologies with DevOps development. For this purpose, three sample projects, i.e. a website for online faculty recruitment, a car search engine, and a scientific calculator tool in java have been designed through traditional methods and later on uploaded local repositories to GitHub for implementation through DevOps tools. Different parameters for measurement of projects were calculated.

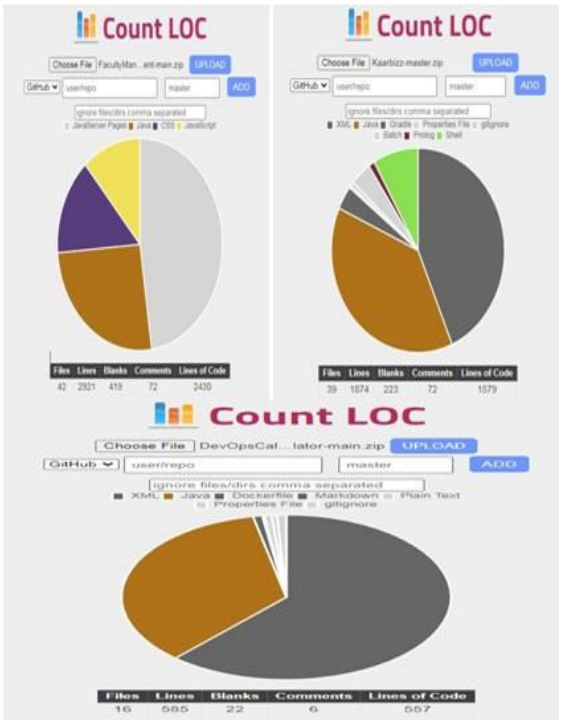

Fig. 1. Components/modules and LOC for sample java projects.

Figure 3, gives the total count of lines of code along with different components and other parameters to consider for metric evaluation purposes. Table I shows the descriptive measures of these sample projects.

TABLE I. SOFTWARE METRIC CLASSIFICATION OF THE SOFTWARE PROJECT

Name of Java project	Size	No of components/ modules	Project domain
Web site for online faculty recruitment (Project1)	2430	18	Web based
Car search application (Project2)	1579	14	Search application
Scientific calculator tool (Project3)	557	8	Tool

The sample projects mentioned in Table II have Java as the application development environment. These projects were designed in local repositories in the JDK environment and were uploaded in GitHub, which was used as a remote repository for the smooth working of the DevOps environment. After the uploading of the code, the next phase is to plan and write test cases for proper execution and implementation of the code. If the test case fails, then the test case must be rewritten after code refactoring. Successful test case execution is followed by code deployment and monitoring or the operations phase of the project. The complete step by step procedure in terms of process flow diagram for current research work is shown in Figure 2.

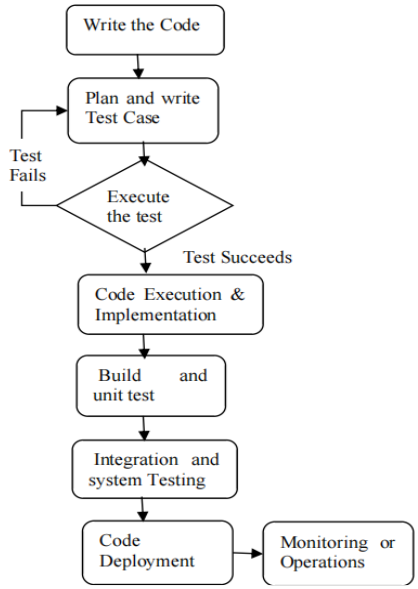

Fig. 2. Process flow diagram of the current research work.

As clearly depicted in the flowchart of Figure 2, writing the test cases and successful code testing is a major part of the DevOps process. For the implementation purpose, Jenkins continuous integration tool is selected which also takes charge of continuous delivery of the product. Different plugins are also installed with Jenkins to ensure smooth working of the tool. Code repository or maintenance is done through GitHub. Ansible with Jenkins is responsible for the continuous deployment of the code. Test cases were written using Junit. The build work was done by Maven. All these steps or procedures are fully automated and were done through the DevOps based hybrid model of automation tool set.

V. MEASUREMENT METRICS OF SOFTWARE DEVELOPMENT

Quality plays an important role in software acceptance. Software quality is not a single factor or value, but it covers many different parameters like testability, predictability, and maintainability. So, quantification of these performance or efficiency parameters becomes more important for measuring the quality value. This research work considers three types of software metrics, i.e. project, process, and product metrics. The project metrics covered in the current work are Project Defect Density (PDD) and Release Deployment Frequency (RDF), to measure defect density covered in the project along with the deployment frequency of the project. Similarly, Process metrics cover the Productivity (PP) of the whole process in the form of the throughput of the system. Lastly, product metrics involve System Risk Identification (SRI) and the reliability of the developed product. These software metrics along with their expected outcome are shown in Table II. The categorization of software metrics in Table II clearly mentions the expected results. These metrics with their calculation formulas and methods are explained below.

A. Project Defect Density (PDD)

PDD refers to whether the software can be deployed. PDD actually depends directly on the presence of defects in the

system. As defects can occur at any stage of software development, checks at regular intervals become a necessary activity of the development. The value of PDD must be low to ensure quality delivery of the software. The PDD formula is:

$$PDD = \sum_{i=1}^{n} \frac{\text{Total number of defects}}{\text{Size of software (in KLOC)}} \quad (1)$$

where n refers to the total number of components or modules in the system and n>0.

Equation (1) is used to first find the defect density of individual components or modules of the system and to get the defect density for the whole system/project by summing them up. Table III shows the PDD of the system as calculated with traditional methods of development and DevOps. TDM refers to Traditional Development Methods and DC refers to DevOps Culture.

TABLE II. CONSIDERED SOFTWARE METRICS AND THEIR EXPECTED OUTCOME

Software metric type	Software metric	Expected outcome
Project	PDD	Low
	RDF	High
Process	SRI	High
Product	PP	High

TABLE III. PDD MEASURE OF SAMPLE PROJECTS

Name of project	Size (LOC)	No of modules	Total no of defects		PDD	
			TDM	DC	TDM	DC
Project1	2430	18	30	15	12.35	6.17
Project2	1579	14	18	8	11.40	5.07
Project3	557	8	10	4	17.95	7.18

Defect density is computed by dividing the total number of defects with the size of the corresponding project as given by (1). In Table III, DevOps gives a low PDD value that clearly indicates more reliability and success of the underlying process. Figure 3 displays a graphical comparison of these methods.

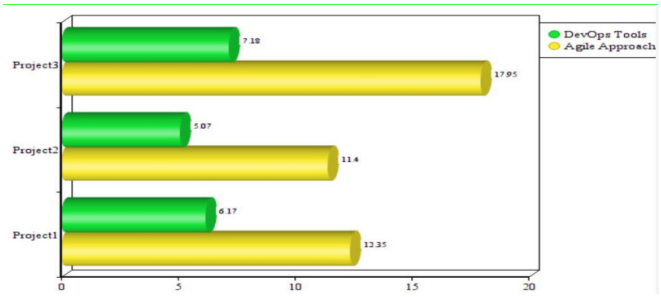

Fig. 3. Graphical comparison of traditional and DevOps development for PDD measure.

B. Release Deployment Frequency (RDF)

RDF reports the total number of deployments in a particular time period. In other words, RDF refers to the rate of released deployments. The higher the value of RDF, the lesser is the chance of errors/defects in the system. The calculation is RDF is done by:

$$RDF = \sum_{i=1}^{n} \frac{\text{Total number of Deployments}}{\text{Time Unit (in Hours)}} \quad (2)$$

Equation (2) calculates the total number of deployments or the release count in particular time units, taken in hours, for individual components and adding them all to get the RDF for the whole system/project. The deployment frequency data of the java projects with the usage of traditional principles and DevOps rules is shown in Table IV. Table IV clearly indicates the high value of deployments that leads to the acceptance of frequent changes in the system. The comparative graph of these development methodologies in Figure 4 shows similar results.

TABLE IV. RDF MEASURE OF SAMPLE PROJECTS

Name of project	Size (LOC)	No of deployments		Time taken to deploy (h)		RDF	
		TDM	DC	TDM	DC	TDM	DC
Project1	2430	18	20	1.8	1.1	10	18.18
Project2	1579	15	16	1.5	0.9	10	17.78
Project3	557	9	10	1	0.7	9	14.29

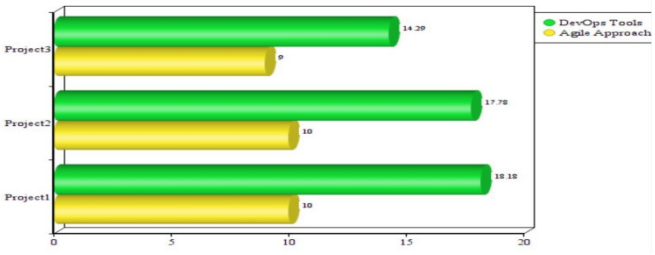

Fig. 4. Comparative graph for DevOps and traditional approaches.

As depicted in Figure 4, the DevOps deployment number is higher as compared to TDM. DevOps deployment speed is much higher for all types of development projects. Automation tools and prior selection of tool chain is the major reason of DevOps success over the other methodologies.

C. System Risk Identification (SRI)

SRI refers to the assessment of risk associated with the project/system to be developed. A high value reflects the identification of more risk components and ensures safe and risk-free delivery of software. The formula for SRI is:

$$SRI = \sum_{i=1}^{n} W_x \text{ , } n>0 \quad (3)$$

where n refers to the total number of components in the system, W_x refers to the weightage assigned to each individual risk and the sum of all W_x gives the SRI number. Risk coverage analysis or risk identification gives the total number of risk sets injected with risk sets executed positively and total risks not tested or broken. Table VI depicts different risk set coverage calculated for traditional development approaches and DevOps.

TABLE V. SRI ESTIMATE OF SAMPLE PROJECTS

Project	Total risk tests	Total no of executed risks tests		Risks broken		Total risks not tested		Risks not executed		Risk coverage (%age)	
		TDM	DC	TDM	DC	TDM	DC	TDM	DC	TDM	DC
Project1	275	188	211	42	23	22	24	23	17	68.36	76.73
Project2	150	104	116	23	12	12	13	11	9	37.81	42.18
Project3	60	40	46	9	5	15	5	6	4	14.55	16.62

In Table VI, the number of total risk sets included, executed, broken, and even ignored or not tested at all in DevOps culture are compared with the similar parameters for traditional development methods. The results again indicate a better risk coverage in DevOps. The graphical comparative study in Figure 5 confirms this statement.

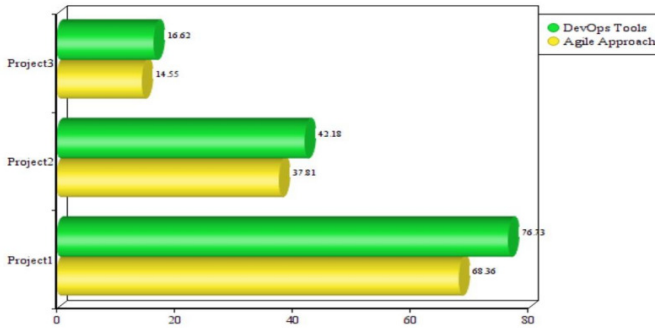

Fig. 5. Risk coverage comparative study graph.

D. Process Productivity (PP)

The productivity of any system is the total units of work done in a particular time period. It refers to the throughput of the system. Process or system productivity is measured by:

$$PP = \sum_{i=1}^{n} \frac{\text{Total number of user stories}}{\text{Time taken to complete}} \quad (4)$$

where user stories refers to the units of work done in a given time period.

Equation (4) finds the productivity of the process or system and computes the summation of productivity of individual components. PP is also measured as the throughput of the whole system or software. It measures the total units of work done in a particular time period. The PPs for traditional methods and DevOps are shown in Table VII. The results of Table VII and of the comparative graph of Figure 6 clearly indicate enhanced throughput/productivity measure for projects developed under the DevOps culture.

TABLE VI. PP ESTIMATION/MEASUREMENT OF CURRENT DATASET

Name of Java project	Size (LOC)	Total no of user stories	Time taken (weeks)		PP	
			TDM	DC	TDM	DC
Project1	2430	180	18	14	10	12.86
Project2	1579	117	12	9	9.75	13
Project3	557	41	4	3	10.25	13.67

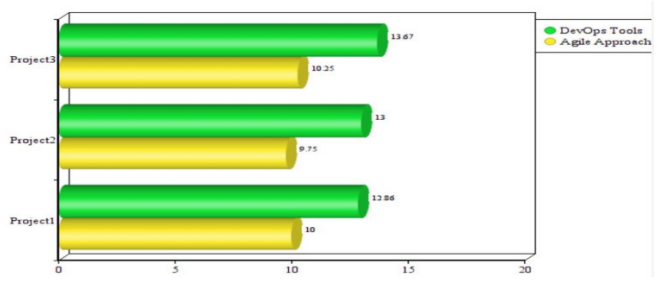

Fig. 6. PP comparative graph for DevOps and traditional approaches.

Figure 8 shows speedy releases or output in terms of productivity measure for DevOps in comparison with traditional development.

Each of the performance metric and graphic visualization tools proves the effectiveness of DevOps as a symbol of quality and speedy delivery with less defect density and high coverage of risk sets.

VI. CONCLUSION AND FUTURE WORK

Fast and quality delivery is an essential indicator for assessing the power of software development methodology. This study examined the efficacy of the previously proposed and implemented DevOps-based hybrid model of automation tools. In this esteem, we presented three java based projects from different domains and compared the performance of DevOps culture with traditional methods of software development based on performance metric measurement values. The measured results showed the superior performance of DevOps culture.

Although previous studies developed many models in software development, no such model has been developed to analyze the effect of prior automation tool selection, before commencement of actual development, on delivery or deployment. Our proposed, implemented and now validated, DevOps-based hybrid model of tool chains comes out to be more technically focused in terms of software development. As a part of future work, real time industry data can also be taken or our own tool can be designed to get more automated results. Finally, it would be interesting to go deeper in the analysis to identify another automation tool that could implement more DevOps projects in different arenas.

REFERENCES

[1] P. Narang and P. Mittal, "Hybrid model for software development: an integral comparison of DevOps automation tools," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 27, no. 1, pp. 456–465, Jul. 2022, https://doi.org/10.11591/ijeecs.v27.i1.pp456-465.

[2] P. Narang and P. Mittal, "Implementation of DevOps based Hybrid Model for Project Management and Deployment using Jenkins Automation Tool with Plugins," *International Journal of Computer Science and Network Security*, vol. 22, no. 8, pp. 249–259, Aug. 2022, https://doi.org/10.22937/IJCSNS.2022.22.8.31.

[3] M. Gomes, R. Pereira, M. Silva, J. B. de Vasconcelos, and Á. Rocha, "KPI's for Evaluation of DevOps Teams," in *Information Systems and Technologies*, Cham, 2022, pp. 142–156, https://doi.org/10.1007/978-3-031-04829-6_13.

[4] W. W. Rovce, "Managing the Development of Large Software Systems," in *Technical Papers of Western Electronic Show and Convention*, Los Angeles, CA, USA, Aug. 1970.

[5] G. Papadopoulos, "Moving from Traditional to Agile Software Development Methodologies Also on Large, Distributed Projects.," *Procedia - Social and Behavioral Sciences*, vol. 175, pp. 455–463, Feb. 2015, https://doi.org/10.1016/j.sbspro.2015.01.1223.

[6] T. Dingsøyr, N. B. Moe, T. E. Fægri, and E. A. Seim, "Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation," *Empirical Software Engineering*, vol. 23, no. 1, pp. 490–520, Feb. 2018, https://doi.org/10.1007/s10664-017-9524-2.

[7] A. Agrawal, Mohd. A. Atiq, and L. S. Maurya, "A Current Study on the Limitations of Agile Methods in Industry Using Secure Google Forms," *Procedia Computer Science*, vol. 78, pp. 291–297, Jan. 2016, https://doi.org/10.1016/j.procs.2016.02.056.

[8] A. Mishra and Z. Otaiwi, "DevOps and software quality: A systematic mapping," *Computer Science Review*, vol. 38, Nov. 2020, Art. no. 100308, https://doi.org/10.1016/j.cosrev.2020.100308.

[9] P. Debois, "Agile Infrastructure and Operations: How Infra-gile are You?," in *Agile 2008 Conference*, Toronto, ON, Canada, Dec. 2008, pp. 202–207, https://doi.org/10.1109/Agile.2008.42.

[10] A. A. Khan and M. Shameem, "Multicriteria decision-making taxonomy for DevOps challenging factors using analytical hierarchy process," *Journal of Software: Evolution and Process*, vol. 32, no. 10, 2020, Art. no. e2263, https://doi.org/10.1002/smr.2263.

[11] M. Ramzan, M. S. Farooq, A. Zamir, W. Akhtar, M. Ilyas, and H. U. Khan, "An Analysis of Issues for Adoption of Cloud Computing in Telecom Industries," *Engineering, Technology & Applied Science Research*, vol. 8, no. 4, pp. 3157–3161, Aug. 2018, https://doi.org/10.48084/etasr.2101.

[12] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A Survey of DevOps Concepts and Challenges," *ACM Computing Surveys*, vol. 52, no. 6, Aug. 2019, Art. no. 127, https://doi.org/10.1145/3359981.

[13] D. Trihinas, A. Tryfonos, M. D. Dikaiakos, and G. Pallis, "DevOps as a Service: Pushing the Boundaries of Microservice Adoption," *IEEE Internet Computing*, vol. 22, no. 3, pp. 65–71, Feb. 2018, https://doi.org/10.1109/MIC.2018.032501519.

[14] K. Aldriwish, "A Deep Learning Approach for Malware and Software Piracy Threat Detection," *Engineering, Technology & Applied Science Research*, vol. 11, no. 6, pp. 7757–7762, Dec. 2021, https://doi.org/10.48084/etasr.4412.

[15] M. F. Hyder and M. A. Ismail, "INMTD: Intent-based Moving Target Defense Framework using Software Defined Networks," *Engineering, Technology & Applied Science Research*, vol. 10, no. 1, pp. 5142–5147, Feb. 2020, https://doi.org/10.48084/etasr.3266.

[16] M. A. Silva, "Productivity Gains of DevOps Adoption in an IT Team: A Case Study," in *27th International Conference on Information Systems Development*, Lund, Sweden, 2018.

[17] J. Stoneham, P. Thrasher, T. Potts, H. Mickman, and C. DeArdo, *DevOps Case Studies: The Journey To Positive Business Outcomes*. IT Revolution.

AUTHORS PROFILE

Poonam Narang, is a research scholar, pursuing her PhD from the Department of Computer Science and Applications, Maharishi Dayanand University, Rohtak, Haryana under the supervision of Respected Dr. Pooja Mittal. Author's Qualification is M.Phil. (CS), MCA. She had attended many National and International Conferences and has also published many research papers.

Pooja Mittal, obtained her Ph.D. from Maharshi Dayanand University. Her areas of research and specialization include data mining, data warehousing, and computer science. She had published more than 50 research papers in renowned International and National Journals and has attended more than 30 Conferences. Currently she is working as an Assistant Professor in the Department of Computer Science & Applications, Maharishi Dayanand University, Rohtak (Haryana).