ISSN: 2230-9926

Available online at http://www.journalijdr.com

IJDR

International Journal of Development Research
Vol. 15, Issue, 11 pp. 69422-69426, November, 2025
https://doi.org/10.37118/ijdr.30269.11.2025

RESEARCH ARTICLE OPEN ACCESS

RISK-AWARE DEVOPS: LEVERAGING AI AND ML ALGORITHMS FOR ADAPTIVE SOFTWARE DELIVERY

***Rajeev Kankanala**

Software Development Engineer, Security Products Platform Reliability, Workday Inc, Atlanta, GA, USA

ARTICLE INFO

Article History:
Received 29th August, 2025
Received in revised form
14th September, 2025
Accepted 17th October, 2025
Published online 27th November, 2025

Key Words:
Risk-Aware DevOps.
Predictive Analytics.
Continuous Deployment.
Machine Learning.

****Corresponding author:***
Rajeev Kankanala

ABSTRACT

The growth in the demand for faster software delivers influence the DevOps pipelines to keep up with that demand. On the otherhand, the risk management is falling apart if the pipelines are not properly designed. This study presents a Risk-Aware DevOps framework, which contains predictive and adaptive risk mitigation workflows using artificial intelligence (AI) and machine learning (ML). The proposed framework contains three parts: risk prediction/fusion, adaptive orchestration, and feedback-driven learning. Primarily this framework uses machine learning to predict failures, rank risks and invoke mitigation strategies automatically. In this study, a publicly available NASA PROMISE JM1 dataset was employed for predicting software bugs using the proposed framework. ML models such as Logistic Regression and Random Forest were trained and tested on NASA PROMISE JM1 dataset for prediction accuracies and both attained (ROC-AUC $\approx$ 0.70). We have also simulated a gating policy which exhibited improvements in operations, lowering failure rates and mean time to recovery while maintaining automationat an acceptable level. The results obtained in this study show that it is possible to employ ML-based decision pipelines to lower the risk in software delivery pipelines.All of the datasets, experimental scripts, and trained models used in this study can be found on GitHub at https://github.com/rajeevkankanala11/RiskAwareDevOps-AI.

Citation: Rajeev Kankanala. 2025. *"*Risk-Aware DevOps: Leveraging AI and ML algorithms for Adaptive Software Delivery.*". International Journal of Development Research*, 15, (11), 69422-69426.

INTRODUCTION

The software delivery pipelines are crucial for better risk management and automated deliverable control system, modern software Continuous integration and continuous delivery (CI/CD) pipelines help automated this approach in a seemless manner (Fitzgerald & Stol, 2017). With the advancements in these pipelines, the DevOps gradually moved towards AI-driven software delivery platform instead of manual control alone (Bass et al., 2015). This indeed influences the risk awareness and decision-making strategy in the software delivery process (Rahman et al., 2019).

Motivation & Problem Statement: The most common problem in the DevOps community right now is the identification of the risks, which happens after the deployment affecting the production timelines (Kim et al., 2016). This latency causes expensive fixing routines and effects the customer trustworthiness due to increased false positives which are even harder for the engineers to spot the difference between the real risks and false ones (Kuzniarz & Amrit, 2022). Once the risk is found, trying to fix it in time is crucial as otherwise there can be an increase of the mean time to recovery (MTTR) (Kaur & Singh, 2021) resulting in the cost-risk trade-off. Several automated mitigation plans help reduce the immediate risks, but they can increase the overall delay in feature releases adding operational costs (Forsgren et al., 2018). These elements show that there is a need for more flexible framework which can find the problems early and manage risks with as little human invention as possible (Leite et al., 2019).

The contributions of this study are as follows: Formulation of modular risk scoring mechanism that combines telemetry, static code metrics, test results and vulnerable signals into a single risk estimate. Implementation of predictive risk engine by mixing time series and sequence models to spot high-risk in the early deployments. Implementation of adaptive orchestrator that helps in decision making w.r.t rule-based fallbacks. Evaluation pipelines for datasets, fault injection, metrics w.r.t low risk and improved MTTR. The remainder of the paper is organized as follows: Section 2 describes materials and methods (containing framework, models, algorithms). Section 3 reports simulated and public dataset results and evaluation pertaining to them. Section 4 discusses possible interpretations, existing limitations and future scope. Section 5 presents the conclusion of the study.

MATERIALS AND METHODS

The proposed framework contains layered architecture that put together predictive modeling, adaptive orchestration, and continuous

feedbackas shown in Fig.1. This framework effectively identifies the risk before it happens and produces the right set of instructions to mitigate the risk (Chen et al., 2019). The framework consists of 6 layers, each of them doing a specific job to deliver the timely risk-free software.

Data Acquisition Layer: The data acquisition layer brings together data from different sources in the DevOps pipeline. It gathers all the data logs, application runtime, metrics, deployment status etc (Sharma et al., 2020). The data collection agents make sure to put together structured and unstructured data in real-time. This layer must be maintained well with completeness as the missing data can cause difficulty in predicting risks.

Feature and Storage Layer: The feature and storage layer is responsible in turning the unstructured data into structured form that machine learning models can understand. This layer uses log parsing, word2vec embedding methods to turn unstructured inputs into useful features (Devlin et al., 2019). The feature storage aspect helps in organizing the feature tracking throughout the training, inference and versioning of dataflow. Several specialized databases such as Redis, Feast, Postgre SQL are used for better indexing and pipeline execution.

Risk Detection Engine: The risk detection engine uses ML and AI models for better identification of risk-patterns and performance bottlenecks (Lessmann et al., 2008). A variety of models can be employed based on the data type and category, for instance the tree-based models are best for tabular features whereas the transformers are best for sequential data logs. This approach helps the detection engine to identify the risks at an early stage and mitigate the further pipeline from failure.

Risk Scoring Module: Once the risk detection engine gives output, these outputs are normalized from various data sources and instances (for example: from build, test, deploy and runtime stages). This module then calminates all these instances to make the single-central risk score which is easy to understand. Several aspects such as technical performance hinderence and business-related latency downtime are considered to calculate this single risk factor (Wang et al., 2021). This kind of risk scores can be useful for the business to navigate and allocate certain risk margins for applications that need low latency(Song et al., 2011). Based on this risk scoring aspects, decision regarding the orchestration is carried on for a smoother software delivery.

Adaptive Orchestration Layer: This orchestration layer makes adaptive adjustments to the software pipeline based on the risk scores. These adjustmemts vary from stopping the deployment, starting canary rollouts, automated rollbacks etc. These decision-making routines can be adjusted using risk score rule-based approaches depending on the risk management maturity level of the organization(Li et al., 2017; Sutton & Barto, 2018). This layer is adaptive because it uses ML and AI model scores for better adjust the flow of the pipeline as well as rely on the human intervention whenever needed. This helps the orchestrator to reduce the overall downtime of the pipeline and avoid any failure in deployment.

Feedback and Continuous Learning Layer: The feedback and continuous layer help increase the retrospective aspect of the overall pipeline by feeding back the success and failure cases after each deployment (Lu et al., 2018). This feedback loop helps manage the AI risk detection models to tune w.r.t concept drift on the data distribution. This feedback also helps the organization to reset the decision thresholds and improve mitigating strategies overtime to improve the overall DevOps pipeline.

Data Sources and Feature Engineering: The model-driven DevOps framework is dependent on the quality and diversity of the data that is used to train the risk predictability aspect. he dependability of any predictive DevOps framework is fundamentally contingent upon the quality and diversity of its input data. The data acquisition and feature storage layers help organize the data to be able to navigate through useful information and turning that into structured and trainable data (Zhu et al., 2021).

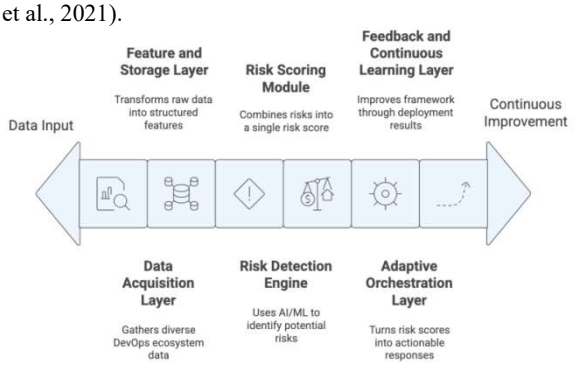

Fig.1 Proposed Risk-Aware DevOps Framework Architecture

Data Sources: The data sources from various stages of the software delivery pipeline are integrated into heterogenous datastream. The data elements such as CI/CD logs, system telemetry, application and repository metadata etc are collectively used for the risk-prediction modeling:

- **CI/CD logs:** these data logsfrom build, test, and deployment stages are used to track the failure in the environment.
- **System telemetry:** these system-based metrics such as performance of CPU and memory, error rates and stability problems help track the early system conitions.
- **Application and service logs:** these app and service logs contain structured and unstructured messages that reveal behavioral problems that metrics cannot.
- **Repository metadata:** this metadata help understand the overall change in the codebase w.r.t commit history, code churn and duplication ratios etc.
- **Security scan outputs:** these outputs help assess the vulnerabilities anddependency weaknesses before the product is released.
- **Deployment descriptors:** this deployment infrastructure logs help understand the configuration files, manifests, infrastructure-as-code elements, and indeed help find chnages in the environment.
- All the above discussed data elements are timestamped and tagged w.r.t pipeline stage. This helps the sequence-based AI models to learn useful information that influence the operational outcomes.

Feature-Engineering Process: The feature engineering is an integral part of the data curation and preparation process that helps transform the raw data into meaningful structured information for the risk detection engine to learn (Guyon & Elisseeff, 2003). Table 1 shows these feature-engineered elements categorized into four groups: static, dynamic, log-derived, and security-related inputs. Algorithms such as Spell, Drain are employed to convert logs into templates and numerical metrics into standard formats and categorical variables into one-hot encoded variables. Once the data is represented in the proper format, then the statistical aggregations are used to get the summary of how things change over time. Also, during this process several embedding generations are made to employ doc2vec to turn sequence of text into vectors (Zhang et al., 2019). Once the useful features are converted to the desirable format using embeddings, then the filtering of mutual information is carried out removing very similar and redundant features. Finally, the useful feature-engineered data is stored for training and inference along with the versioning to repeat the experiments.

Table 1. Feature Categories

Category	Example features	Type
Static	LOC delta, cyclomatic complexity	scalar
CI tests	failed_tests, test_duration	scalar/time
Runtime	cpu_util_mean_5m, error_rate_1m	time-series
Security	cvss_max, vuln_count_new	scalar
Logs	top_error_counts,log_cluster_score	vector

Risk Scoring Formulation and Design: The main aspect of the proposed framework exists in its capability to quantify, predict and respond to the risks within the DevOps environment (Bishop, 2006). This framework design starts with mathematical formulation of defining the risk followed by the ML-based algorithms to make predictions and thereby orchestration of system in terms of risk mitigation. This way, the pipeline is easy to understand and automatically invoking risk estimation, algorithmic orchestration and decision making in the appropriate order.

Risk Scoring Formulation: For the risk score formulation, a probabilistic function is defined based on multivariant features collected at every stage of CI/CD. This risk score gives the probability of deployment failure or service failure in the pipeline of CI/CD.

Given every stage$s$at time$t$, risk score per each stage is estimated as:

$$r_s(t) = \sigma(w_s^T \cdot x_s(t) + b_s) \quad (1)$$

where$x_s(t)$ isvector data extracted from the logs and results for given stage s

w_s(weight),b_s(bias) is model parameters
$\sigma(\cdot)$ is activation function making sure the output lies between 0 and 1

The cumulative risk $R(t)$ is calculated by combining stage-wise risks, weighted by the relative significance:

$$R(t) = \varphi(\Sigma\, \lambda_s \cdot r_s(t)\,,\, C(t)\,) \quad (2)$$

where$\lambda_s$issignificant weight of stage s, $C(t)$ is cost vector that gives significance of the vector w.r.t business impact, and $\varphi(\cdot)$ is the aggregation function that combines both the technical and contextual risks.

In a way, $R(t)$ can be considered as the conditional probability of failure w.r.t time horizon H:

$$R(t) \approx P(\text{ failure within } [t,\, t+H] \,|t\,) \quad (3)$$

Thereby, a dynamic threshold $\tau(t)$ will act as an automated mitigation trigger:

$$\text{If } R(t) \geq \tau(t) \rightarrow \text{Trigger Mitigation} \quad (4)$$

Given $a$being the mitigation action with cost $c(a)$ and probabilistic expected cost$g(a \mid R)$, then trigger will invoke if:

$$g(a \mid R)\,.\, R(t) > c(a) + \eta \quad (5)$$

where η is a decision margin which is considered as a hyperparamter

Predictive Modeling and Design: The probabilities $r_s(t)$per each stage can be produced by the risk detection engine by the combination of traditional and machine learning models. The model type per each stage can be selected using the data characteristics and other facters. For instance, tree-based models such as random forest, XGBoost etc handles tabular, heterogeneous data points (test counts, code metrics and build durations, etc.). Similarly, sequential models such as LSTM and GRU captures temporal dependencies in the delivery pipelines (recurring failure patterns, sequential logs, etc.). The transformer-based models are usually effective at capturing the context and correlation across various pipeline stages(Shahin et al., 2017). To identify the unseen data points and rare patterns, autoencoders can be employed at each stage. A part from these, hybrid models combine structured and unstructured data for a holistic risk estimation. In the process of training, the ML/AI models use historical data and relevant labels for success and failure. The predictions are targeted to attain high accuracy by minimizing the reconstruction error or optimization of loss functions such as cross-entropy. Certain level of regression and calibration methods such as platt scaling are used to match the training data w.r.t real-world instances (Sutton & Barto, 2018; Mahmood et al., 2020). The set of algorithms help manage risk prediction and adaptive orchestration processes simultaneously predicting risks and combining the outputs from different stages to choose a best way to mitigate risk. Algorithm 1 outlines this risk prediction and fusion aspect by successfully estimating risk before the deployment happens.

Algorithm 1: Risk Prediction and Fusion

Input: Pipeline logs, metrics, static features from current pipeline
Output: Overall risk score R

- For each stage s in the pipeline:
- Extract feature vector x_s from input
- Compute stage risk: r_s = Model$_s$.predict(x_s)
- End For
- Fuse all stage-wise risks using weighted aggregation:
- $R = \varphi(\ \Sigma\ \lambda_s \times r_s\ ,\ C(t)\ )$
- Return overall risk score R

After the Algorithm 1 is invoked, the risk scores can be calculated, which are then used by Algorithm 2 to employ mitigation strategies. Algorithm 2 focuses on the mitigation startegies that take into consideration the costs as well as rules to attain efficient operation of adaptive orchestration. In the advance stages of the mitigation strategies, the orchestration engine is trained with feedback from the reinforcement learning (RL) which can be rewarded by reliability gain – mitigation cost. The goal of this Algorithm 2 adaptive orchestration policy is to attain actionable optimal mitigation action a^* while considering the risk scores and contextual parameters gathered from the various pipeline stages.

Algorithm 2: Adaptive Orchestration Policy

Input: Risk score R, contextual parameters ($C(t)$)

Output: Optimal mitigation action (a^*)

- Define possible actions A = {*proceed*, *run_extra_tests*}
- For each action a in A:
- Estimate expected benefit: $g(a \mid R)$
- Estimate operational cost: $c(a)$
- Compute utility: $U(a) = g(a \mid R) \times R - c(a)$
- End For
- Select action $a^* = \text{argmax}_a\ U(a)$
- If $U(a^*) > \delta$ (decision margin):
- Execute action a^*
- Else:
- Trigger *human_review*
- Log decision outcome and feedback for retraining

Although the automation designed in this framework uses the adaptive orchestration using machine learning algorithms, a safety human-in-the-loop feature is also employed. Therefore, low-cost high-confidence mitigation actions can be done with the model-driven automatic approach. On the otherhand, high-impact actions such as rolling back production are sent for the approval from the operator with proper explanations and reasoning tools like SHAP and LIME etc. By using this hybrid approach of automation and oversight, the mitigation strategies become open and trustworthy (Lundberg & Lee, 2017).

RESULTS

In this study, NASA PROMISE JM1 dataset (Menzies et al., 2007) was employed to test the real-time risk-aware DevOps predictive models. During the data curation, the dataset was cleaned by removing redundant information and NaN columns and filling the missing numeric values with the median. For a stable training, all

numeric features were normalized using zero mean and unit variance (Nagappan & Ball, 2005; Radjenović et al., 2013). The original balance between the train and test samples w.r.t failure classes is preserved by sampling 75 (train) : 25 (test) percentage. Initially, two supervised classifiers are trained such as logistic regression and random forest with class-balanced 200 trees. The standard predictive metrics such as accuracy, precision, recall, F1-score and ROC-AUC curves are used to test the model performance. Also, a simulated operational "gating mitigation" for each test run is employed to test the random forest predict probability > 0.6. This is followed by the assumption that mitigation stops instances with probability of 0.8. Using this gating mitigation strategy(Rahman et al., 2019), the performance evaluations are more reliable by considering failure-rate reduction, MTTR, false-positive mitigations and post-mitigation (Lessmann et al., 2008). The test case scenarios of both models are evaluated on the NASA JM1 dataset w.r.t performance evaluation metrics.

Table 2. Model Performance on NASA JM1 Dataset

Metric	Logistic Regression	Random Forest
Accuracy	0.6973	0.8046
Precision	0.3326	0.4901
Recall	0.5598	0.2353
F1 Score	0.4173	0.3179
ROC AUC	0.6971	0.7168

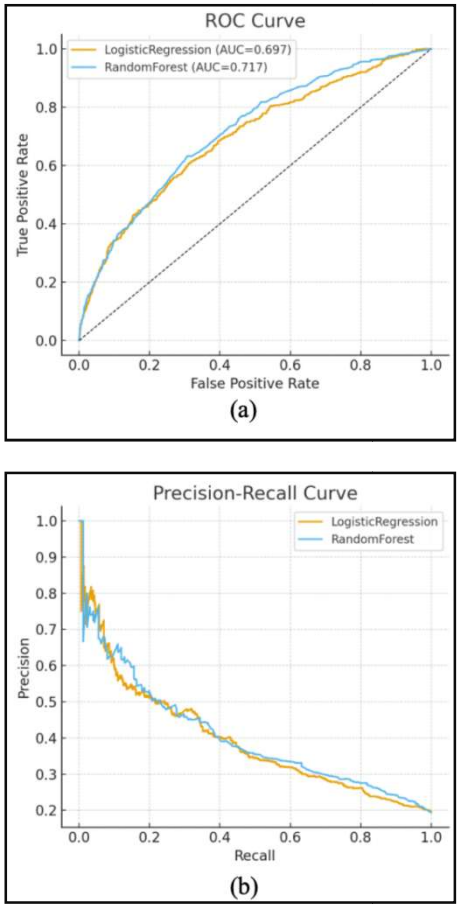

Fig. 2. ROC and Precision-Recall curves for both models

The logistic regression model exhibited higher recall of 0.56 but lower precision, it is more sensitive to instances that are having defects but less specific. On the otherhand, the random forest model has a better overall AUC of 0.72and accuracy of 80% which discriminates the risk instances with specificity. This random forest model is better for tasks such as automated gating and risk prioritization in DevOps pipeline where the precision is required. Figure 2 illustrates the precision recall curves alongside the ROC curves for both the models.

The overall risk-aware gating simulation performed in this study can cut the deployment failure rate from 19.4% to 16.8%. In neumerical values, this stopped 71 incidents and caused 66 false-positive mitigations. This is because of the feature selection process employed in this study which is outlined in Figure 3 as the 15 most important features. Using these features, the MTTR metric went down from 4.2 hours to 3.86 hours w.r.t average chance of mitigation success of 0.8. Table 3 shows the operational impact of risk-aware gating mechanism employed with random forest model.

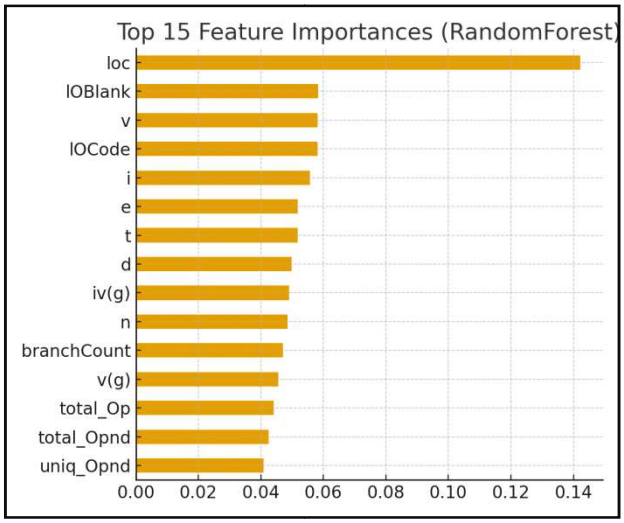

Fig. 3. Top 15 important features used in the analysis (RF)

Table 3. Operational Impact of Risk-Aware Gating (Random Forest)

Metric	Value
Baseline Failure Rate	0.1936 (19.36 %)
Failure Rate After Gating	0.1675 (16.75 %)
Incidents Prevented (Simulated)	71
Total Incidents (Test Set)	527
False Positive Mitigations	66
Baseline MTTR (hours)	4.20
MTTR After Gating (hours)	3.86

DISCUSSION

The assessment of the proposed framework utilizing NASA JM1 dataset validates the effectiveness of employing AI for predictive and adaptive DevOps pipeline. The models employed in this study such as logistic regression and random forest exhibited balanced predictive performance around AUC of 0.70. This proves that the basic models such as logistic regression and random forest can identify the risk patterns when used in the automation of the DevOps pipelines. Although the study formulated the framework for automated risk assessment and adaptive orchestration; there are a few shortcomings that are to be fixed. The NASA JM1 dataset used in this study is too small and cannot represent all the edge cases related to the risks as it is used in the static pre-deployments. It doesn't capture the dynamic real-time Devops system like microservice logs or container metrics etc. The second limitation is regarding the gating simulation mechanism which assumes fixed thresholds and by doing thatthe context-sensitive patterns might be missing out. Finally, the experimental design planned in this study only captures the sequential steps towards the deployment pipeline but cannot fully incorporate the closed-loop system for a better real-time DevOps simulation.In the future, this framework can be expanded to handling more context-aware system decisions. To make the system much efficient in learning from its own performance, reinforcement learning and online learning techniques can be employed with a feedback learning loop.

CONCLUSION

This study proposes a risk-aware DevOps framework which can assess the risk and adaptively use AI model to make decisions. The framework contains three parts: risk prediction, adaptive orchestration and feedback learning. For the empirical evaluation, the study used the NASA PROMISE JM1 dataset to test the proposed framework for risk assessment and orchestration. Two fundamental models such as logistic regression aand random forest models achieved balanced predictive accuracy (ROC-AUC ≈ 0.70), proving that the basic involvement of fundamental models can drastically improve the DevOps pipeline efficiency. Also, a simulated risk-aware gating policy is employed in this study to further reduce failure rates and MTTR.The future work will expand the proposed framework with reinforcement learning with explainable AI for a better transparency and human-in-the-loop approach.

REFERENCES

Amrit, C., Slagter, R. & van Hillegersberg, J. 2022. Understanding DevOps: A theoretical framework for collaboration. *Information and Software Technology*, 142, 106726.

Bass, L., Weber, I. & Zhu, L. 2015. *DevOps: A Software Architect's Perspective*. Addison-Wesley.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Chen, L., Babar, M. A. & Zhang, H. 2019. Towards an evidence-based understanding of continuous deployment. *Journal of Systems and Software*, 156, 110–124.

Devlin, J., Chang, M. W., Lee, K. & Toutanova, K. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL-HLT*.

Fitzgerald, B. & Stol, K.-J. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123, 176–189.

Forsgren, N., Humble, J. & Kim, G. 2018. *Accelerate: The Science of Lean Software and DevOps*. IT Revolution Press.

Guyon, I. & Elisseeff, A. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3, 1157–1182.

Kaur, A. & Singh, I. 2021. A review on risk management in DevOps. *International Journal of Computer Applications*, 183(3), 1–6.

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

Kumara, S., Wermelinger, M. & Yu, Y. 2023. Predictive maintenance in DevOps using AI-based anomaly detection. *Empirical Software Engineering*, 28, 45–67.

Kuzniarz, L. & Amrit, C. 2021. Managing noisy signals in continuous delivery pipelines. *Software: Practice and Experience*, 51(11), 2174–2191.

Leite, L., Rocha, C., Kon, F., Milojicic, D. & Meirelles, P. 2019. A survey of DevOps concepts and challenges. *ACM Computing Surveys*, 52(6), 1–35.

Lessmann, S., Baesens, B., Mues, C. & Pietsch, S. 2008. Benchmarking classification models for software defect prediction. *IEEE Transactions on Software Engineering*, 34(4), 485–496.

Li, Y., Chen, Y. & Zhao, Y. 2017. Deep reinforcement learning for automated software testing. *ICSE Workshops*, 96–103.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J. & Zhang, G. 2018. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12), 2346–2363.

Lundberg, S. M. & Lee, S.-I. 2017. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems (NeurIPS)*, 30.

Mahmood, A., Zhang, R. & An, Y. 2020. Risk-aware reinforcement learning for cost-sensitive decision-making. *Applied Intelligence*, 50(5), 1287–1301.

Menzies, T., Greenwald, J. & Frank, A. 2007. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1), 2–13.

Nagappan, N. & Ball, T. 2005. Use of relative code churn measures to predict system defect density. *Proceedings of ICSE*, 284–292.

Radjenović, D., Heričko, M., Torkar, R. & Živkovič, A. 2013. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8), 1397–1418.

Rahman, M., Williams, L., Dang, Y. & Nagappan, N. 2019. Predicting deployment failures in continuous delivery. *Empirical Software Engineering*, 24(5), 2757–2781.

Shahin, M., Ali, N. & Babar, M. A. 2017. Continuous integration, delivery and deployment: A systematic review. *Journal of Systems and Software*, 125, 217–247.

Sharma, S., Dutta, R. & Ghosh, D. 2020. A deep learning approach to sDevOps anomaly detection. *Software: Practice and Experience*, 50(9), 1714–1730.

Wang, S., Liu, Y., Nam, J. & Tan, L. 2021. Deep just-in-time defect prediction. *Empirical Software Engineering*, 26, 1–40.

Zhang, X., Xu, Z., Liu, S. & Zhao, J. 2019. Log event representation learning with deep neural networks. *IEEE Access*, 7, 171234–171247.

Zhu, L., Bass, L. & Weber, I. 2021. DevOps and software architecture: The state of practice, challenges, and research directions. *Journal of Systems and Software*, 180, 111037.

****