International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group | Volume 5, Issue 2, 157-164, 2024
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V5I2P116

Original Article

End-to-End AI-Driven DevSecOps: A Framework for Risk-Aware Testing, Monitoring, and Lifecycle Optimization

Dr. Bhavana Ramesh[1], Dr. Rahul Dutta[2], Dr. Priyanka Salian[3], Dr. Gokul Anand[4]
[1]Department of Artificial Intelligence, Academy of Intelligent Computing and Systems, Assistant Professor, Tiruchirappalli, India.
[2]Department of Information Technology, National School of Information Engineering, Assistant Professor, Patna, India.
[3]Department of Computer Science, Western Institute of Computer Research, Assistant Professor, Surat, India.
[4]Department of Artificial Intelligence, Digital Futures University, Assistant Professor, Salem, India.

Abstract *- Modern software delivery must simultaneously accelerate release cadence, strengthen security assurance, improve test efficiency, and sustain dependable operations in highly distributed environments. Yet many organizations still manage testing, security, observability, and post-release optimization as loosely connected activities, creating fragmented feedback loops and delayed risk response. This paper proposes an end-to-end AI-driven DevSecOps framework that unifies defect prediction, threat-informed testing, continuous security gating, telemetry-driven monitoring, and lifecycle optimization within a single risk-aware control architecture. The framework is grounded in recent literature on DevSecOps adoption, machine learning for software testing, software defect prediction, secure microservices, and production observability [1][2]. Its central contribution is a layered decision model that fuses code-level signals, service dependency context, runtime anomalies, vulnerability intelligence, and business criticality to prioritize actions across planning, build, release, and operations. Rather than treating testing and monitoring as isolated checkpoints, the proposed approach closes the loop from operational evidence back to backlog refinement, test generation, policy tuning, and architectural remediation. The paper further defines governance requirements, quantitative scoring logic, and an evaluation blueprint for enterprise deployment scenarios. By integrating predictive analytics with secure delivery controls and observability-informed adaptation, the framework provides a practical foundation for improving release confidence, reducing mean time to detection, and allocating engineering effort where technical and business risk are highest.*

Keywords *- Devsecops, Software Defect Prediction, AI-Driven Testing, Observability, Risk-Aware Quality Assurance, Continuous Security Testing, Lifecycle Optimization, Secure Software Engineering.*

1. Introduction

Software delivery pipelines increasingly operate under simultaneous pressure for speed, resilience, compliance, and continuous adaptation. In practice, however, many organizations still preserve fragmented boundaries between development, testing, security, release engineering, and operations. Recent reviews show that DevSecOps remains a socio-technical transformation rather than a simple toolchain upgrade, with persistent gaps in automation, measurement, governance, and developer-centered security integration [1][2][3]. NIST guidance likewise emphasizes that secure CI/CD in microservices settings requires explicit orchestration of build controls, policy checkpoints, and operational feedback rather than isolated point solutions [4]. Architecture-centered governance models further suggest that AI-supported lifecycle decision intelligence can help organizations prioritize risk, align changes with business objectives, and reduce coordination latency across complex delivery systems [5].

At the same time, machine learning is changing how teams reason about software quality. Reviews of AI and machine learning in software testing show strong growth in predictive prioritization, intelligent test selection, automated defect localization, and adaptive regression analysis [6]. Systematic reviews of software fault prediction also indicate that statistical learning, ensemble methods, and deep learning can expose fault-prone components earlier than traditional heuristic approaches when data quality and evaluation discipline are sufficient [7]. Comparative analysis of automated testing frameworks in enterprise Java environments demonstrates the operational importance of reliable automation infrastructure for sustained quality engineering [8]. Beyond pre-release activities, production ML observability research argues that effective deployed systems need end-to-end visibility for detection, diagnosis, and reaction to pipeline failures and silent data issues [9]. Continuous security testing studies similarly show that DevSecOps adoption becomes materially stronger when security evidence is embedded throughout the delivery flow rather than postponed to late-stage review [10].

These developments create an important opportunity, but they also expose a methodological gap. Much of the current literature focuses on either DevSecOps practices, defect prediction models, AI-enabled testing, or observability architectures in relative isolation. Work on the expanding role of machine learning in software development underscores the broader transformation of engineering workflows [11], while research on vulnerability databases highlights the continuing need to

operationalize external security intelligence within software engineering processes [12]. Deep learning reviews in software defect prediction show promising performance but also raise concerns regarding interpretability, imbalance, and operational fit [13]. Frameworks for embedding security practices across the software process strengthen coverage [14], and architecture-centered agile governance approaches demonstrate how defect prediction and automated testing can jointly support lifecycle control [15]. Still, organizations often lack a unified decision architecture that determines which assets should be tested more deeply, which security findings should block release, which runtime anomalies merit immediate rollback, and how post-deployment evidence should reshape future engineering work.

This paper addresses that gap by proposing an end-to-end AI-driven DevSecOps framework for risk-aware testing, monitoring, and lifecycle optimization. The framework is not limited to a single algorithm or sector. Instead, it defines a composable reference architecture that integrates predictive quality signals, threat-informed controls, service-topology awareness, runtime telemetry, and governance policies into a closed-loop operating model. The paper makes four contributions. First, it synthesizes recent literature into a coherent problem framing for risk-aware DevSecOps. Second, it introduces a layered framework that links planning, implementation, verification, release, and operations through shared risk objects and decision policies. Third, it formalizes a scoring approach for prioritizing tests, deployment gates, remediation actions, and optimization opportunities. Fourth, it outlines an evaluation blueprint and multi-domain deployment scenarios to guide empirical validation in enterprise environments.

Table 1: Literature-Derived Gaps Addressed by the Proposed Framework

Gap Theme	What Prior Work Does Well	Remaining Limitation	Framework Response
DevSecOps adoption	Surveys map practices, challenges, and metrics [1][2][3].	Weak cross-stage decision coupling.	Shared risk objects link planning, testing, release, and operations.
AI in testing	Recent studies improve prediction and prioritization [6][16][20].	Signals are often optimized in isolation.	Risk fusion joins defect, security, dependency, and business context.
Security integration	Continuous and model-based security testing are advancing [10][14][23].	Tool outputs are not always action-ranked.	Policy engine converts findings into stage-aware decisions.
Observability	Production and runtime monitoring expand post-deployment visibility [9][24][25].	Telemetry rarely updates upstream controls systematically.	Closed-loop optimization feeds incidents back into models and test policies.
Architecture sensitivity	Topology and propagation studies expose distributed impact [28][36].	Many pipelines still score changes at component level only.	Dependency-aware blast radius is a first-class input to prioritization.

2. Background And Related Work

Recent work on software defect prediction has broadened from static classification toward more data-efficient and deployment-relevant formulations. Active learning and ensemble-based analysis show that useful defect prediction can be achieved with reduced labeling effort while maintaining practical discrimination between fault-prone and less risky components [16]. At the same time, regression-performance research demonstrates that execution ordering matters: test case prioritization can surface regressions earlier and improve engineering feedback economics even when the system under test includes performance-sensitive microbenchmarks [17]. In secure enterprise platforms, microservices-based architectures require dependable interfaces, strong access controls, and infrastructure-aware deployment patterns to satisfy sectoral compliance and scale requirements [18]. JIT defect prediction studies in mobile software further indicate that shallow learners may outperform deeper models when data regimes, latency requirements, or explainability needs favor simpler decision boundaries [19]. Related work on diversified and fault-proneness-aware prioritization shows that regression testing benefits when historical fault signals are combined with coverage and diversity information rather than applied independently [20].

Risk-aware quality engineering is especially important in domains where operational failures propagate into financial, health, or customer-impacting events. A recent AI framework for automated testing and quality assurance in core banking systems emphasizes that prioritization mechanisms must combine technical risk with transactional criticality and regulatory sensitivity [21]. Search-based test prioritization research similarly shows that fault sensitivity and multi-objective optimization can improve ordering decisions under cost constraints [22]. Secure model-based development methodologies extend this line of thinking by integrating threat modeling, countermeasure selection, and security test generation into DevSecOps-compatible pipelines [23]. For cloud-native data platforms, observability frameworks tied to defect prediction point to the need for coordinated monitoring of pipeline stages, data quality, and failure signals rather than isolated dashboards [24]. Runtime monitoring research for responsible machine learning adds another dimension: operational governance should evaluate not only performance degradation but also fairness, privacy, and model behavior constraints during deployment [25].

Beyond technical optimization, governance and architecture remain central. Unified reliability engineering frameworks argue that secure and autonomous systems need policy-backed coordination between design choices, quality checkpoints, and

operational decision rights [26]. Empirical studies of ML testing in the wild show that real-world teams test data, model behavior, infrastructure, and integration logic in uneven ways, with important blind spots around certain properties and stages of the ML workflow [27]. Graph-based dependency models for distributed systems demonstrate that failure propagation cannot be understood solely at the component level; service relationships and topology centrality materially influence incident impact [28]. DevSecOps reviews in IoT and other cyber-physical settings also reinforce that continuous attack monitoring and security automation are essential when operational exposure is high [29]. Meanwhile, practitioner studies on DevOps risk identify organizational, cultural, technical, and ethical hazards that cannot be solved through tooling alone [30].

Taken together, the literature points to five recurring observations. First, software quality and security risk are multi-source phenomena that demand data fusion. Second, model effectiveness depends on context, interpretability, and feedback timeliness, not only predictive accuracy. Third, architecture and service dependencies shape which failures become business critical. Fourth, observability must extend beyond operational uptime to cover data, model, release, and security evidence. Fifth, governance must explicitly determine how evidence changes engineering action. What remains insufficiently specified is a unified, end-to-end control framework capable of absorbing these observations into one practical operating model.

3. Research Gaps and Design Requirements

Based on the foregoing literature, an end-to-end AI-driven DevSecOps framework should satisfy six design requirements. R1: shared risk representation, so that defect likelihood, vulnerability severity, architecture criticality, and business impact can be reasoned about together. R2: stage-aware actionability, meaning the same evidence can drive different actions in planning, coding, testing, release, or operations. R3: observability closure, so that runtime evidence updates future quality and security controls rather than remaining local to monitoring tools. R4: explainable prioritization, because release and remediation decisions must remain auditable in enterprise contexts. R5: architecture sensitivity, since distributed systems require service-topology and dependency awareness. R6: governance compatibility, ensuring that technical recommendations remain aligned with compliance, ownership, and change-management constraints.

These requirements are reinforced by applied studies in cloud-native prescription automation and healthcare operations, where AI-enhanced document processing, secure microservices, and predictive inventory analytics demonstrate that engineering decisions increasingly blend software assurance with business workflow optimization [31]. Similarly, empirical work on security in large-scale agile development shows that governance structure, process adaptation, automation, and tool support jointly determine how well security is operationalized [32]. Standard-compliant DevSecOps design principles for operational technology further indicate that information flows and evidence models must be deliberately structured to enable adoption at scale [33].

Table 2: Layers of the End-To-End AI-Driven Devsecops Framework

Layer	Primary Inputs	AI / Analytics Function	Primary Output	Governance Check
Risk ingestion	Code changes, scanners, tests, telemetry, incidents, business events	Normalization, lineage building, evidence graph construction	Unified risk records	Data completeness and provenance
Decision intelligence	Risk records, history, topology, domain criticality	Defect prediction, exploitability ranking, anomaly detection, propagation estimation	Prioritized scores and rationales	Model performance and explainability
Policy and gating	Scores, thresholds, environment profile, exceptions	Rule evaluation and action mapping	Test expansion, hold, rollout, approval, rollback	Threshold adherence and override traceability
Runtime observability	Logs, metrics, traces, SLOs, business-process indicators	Anomaly detection, signature comparison, drift checks	Incident signals and confidence updates	Operational evidence sufficiency
Lifecycle optimization	Incident history, false positives, drift, remediation outcomes	Retraining, threshold tuning, suite rebalance, architecture recommendations	Backlog and control improvements	Learning-loop review

4. Proposed End-To-End Framework

The proposed framework is organized as five tightly coupled layers: risk ingestion, AI decision intelligence, policy and gating, runtime observability, and lifecycle optimization. A central feature store and evidence graph bind these layers together. Every significant software artifact requirement, code change, test suite, build, deployment, service, incident, vulnerability record, and business process event is represented as a typed node with associated signals, lineage, and ownership metadata. This representation allows the framework to reason across the entire delivery lifecycle rather than within isolated tools.

Layer 1, risk ingestion, collects heterogeneous evidence from source control, static analysis, dependency scanners, vulnerability databases, unit and integration tests, code review systems, deployment manifests, service catalogues, tracing platforms, metrics backends, incident systems, and business telemetry. In this layer, data is normalized into a shared schema with fields such as artifact type, timestamp, confidence, severity, impact scope, affected service, environment, and control status. The rationale echoes converged AI architectures for lifecycle optimization and cyber-risk reduction [34], but the proposed framework adds explicit cross-stage risk semantics so the same issue can be examined from testing, security, reliability, and business viewpoints simultaneously.

Layer 2, AI decision intelligence, transforms raw evidence into operationally meaningful scores. Unlike single-purpose predictors, this layer contains multiple specialized models: module-level defect prediction, just-in-time change risk estimation, vulnerability exploitability ranking, failure propagation estimation over service graphs, test-case prioritization, anomaly detection, and policy-violation prediction. Continuous security testing case studies show that security tools become more useful when their outputs are embedded within delivery decision points [35]. Likewise, microservices research highlights that service decomposition improves scalability but also increases coordination and failure-surface complexity [36]. Therefore, the framework combines model outputs through a risk fusion engine instead of allowing each tool to act independently.

The core fusion equation is defined as follows:

$$\text{RiskPriority}(a, t) = w1*\text{DefectProb} + w2*\text{VulnScore} + w3*\text{PropagationImpact} + w4*\text{OperationalAnomaly} + w5*\text{BusinessCriticality} - w6*\text{ControlConfidence}$$

Where a denotes an artifact or service, t denotes time, and the weights are policy-calibrated by domain and environment. DefectProb is produced by software quality models trained on code metrics, change history, test outcomes, and issue lineage [37]. VulnScore aggregates scanner findings, external vulnerability intelligence, and exploitability indicators. PropagationImpact is computed from topology-aware dependency analysis and incident blast-radius estimation. OperationalAnomaly captures abnormal traces, latency shifts, error spikes, or data drift. BusinessCriticality represents customer, revenue, compliance, or mission impact. ControlConfidence discounts risk when evidence indicates strong mitigation coverage, such as effective tests, security controls, canary success, or incident-free stability windows.

Layer 3, policy and gating, translates scores into stage-specific decisions. This is where the framework differs from traditional quality dashboards. The same underlying risk object can trigger backlog refinement, test augmentation, mandatory threat review, release delay, progressive rollout, or production rollback depending on context. For example, a high-risk code change with moderate vulnerability severity may be allowed into staging but require expanded regression selection, strengthened contract testing, and canary monitoring. By contrast, a medium-risk functional change in a high-criticality payment or prescription workflow may require stronger pre-release evidence despite lower predicted defect probability. Separate policy profiles can be maintained for sandbox, non-production, and production environments to prevent over-blocking.

To preserve explainability, each automated recommendation is accompanied by a rationale vector. For testing actions, the vector includes the dominant features influencing defect likelihood, the historical defect density of affected modules, impacted dependencies, and the specific tests selected for prioritization. For security actions, the vector includes vulnerability source, affected asset class, exploitability context, and proposed controls. For release actions, the vector contains deployment scope, service centrality, recent anomaly context, and business criticality. This design keeps the framework compatible with audit-heavy environments while still enabling machine-assisted acceleration.

Layer 4, runtime observability, continuously measures whether pre-release confidence was justified. Traditional observability stacks focus on logs, metrics, and traces. The proposed framework extends this by linking runtime telemetry to pre-release risk hypotheses. Each deployment receives an expected-risk signature: the modules predicted as fault-prone, the services most exposed to propagation, the tests covering dominant concerns, and the controls expected to mitigate known issues. Post-release telemetry is evaluated against that signature. When incidents arise outside the predicted envelope, the framework flags a model-governance event and updates feature engineering, thresholds, or missing-signal inventories. This design draws on production ML observability [9], runtime monitoring for responsible ML [25], and business-process observability concepts that connect technical anomalies with workflow degradation [38].

Layer 5, lifecycle optimization, closes the loop from operations back to engineering planning. When incidents recur, false positives rise, or model confidence erodes, the framework does not simply generate alerts. Instead, it issues structured improvement actions: retrain defect models, add or retire features, rebalance test suites, revise architecture boundaries, tune rollout policies, strengthen infrastructure controls, or refactor services with excessive dependency centrality. Recent comparative studies of machine learning models for defect prediction [39] and predictive optimization in pharmacy fulfillment workflows [40] both underscore that sustained performance depends on continuous recalibration rather than one-time

deployment of analytics. In the proposed framework, optimization therefore becomes an explicit DevSecOps concern rather than an ad hoc exercise.

Operationally, the framework can be deployed with a modular implementation pattern. A feature pipeline extracts static code metrics, change metrics, scanner outputs, dependency graphs, and runtime telemetry into a versioned store. A model orchestration service generates risk scores on code push, pull request creation, build completion, pre-release approval, and post-deployment intervals. A policy engine evaluates scores against environment-specific rules. An evidence API exposes rationale, lineage, and recommendation history to developer portals and incident tools. Finally, a governance console tracks threshold tuning, exception decisions, drift alerts, and compliance evidence.

A practical advantage of the evidence-graph approach is that it supports cross-artifact lineage. A failed production trace can be linked back to the deployment bundle, the pull request, the tests executed, the vulnerabilities accepted, the service dependencies touched, and the business process steps affected. This makes root-cause analysis faster, but it also enables richer model supervision. For instance, a post-release incident can be labeled not only as a defect escape, but as a topology-amplified defect escape, a misclassified dependency risk, an under-tested integration path, or a policy exception with inadequate compensating controls. Those distinctions improve retraining quality and reduce the tendency of feedback systems to collapse all failures into one generic signal.

The framework also supports progressive delivery. Risk scores can determine not only whether a release proceeds, but how it proceeds. High-confidence changes may follow standard canary and automated promotion rules. Medium-confidence changes may require smaller blast-radius deployments, tighter anomaly thresholds, and a longer observation window before full promotion. Low-confidence but business-critical changes may still ship under executive or domain-owner approval, but only with pre-authorized rollback plans, temporary feature flags, and intensified telemetry capture. This graduated deployment logic ensures that risk awareness influences release strategy continuously rather than through a binary ship-or-block decision.

5. Decision Models and Governance Policies

To make the framework usable in enterprise delivery, decision logic must balance predictive sophistication with operational clarity. Accordingly, the proposed model separates estimation, prioritization, and enforcement. Estimation produces probabilities, severity scores, and anomaly indicators. Prioritization converts those signals into relative action ordering. Enforcement applies organization-specific rules for gating, escalation, or approval. This separation prevents models from becoming opaque release authorities.

For test optimization, the framework assigns each test case a composite utility score:

$$TestUtility(tc) = a1*CoverageNovelty + a2*FaultExposure + a3*DependencyCriticality + a4*HistoricalYield - a5*ExecutionCost$$

Where FaultExposure is derived from change-risk and module-defect models, and DependencyCriticality reflects the business and architectural significance of the services exercised by the test. This formulation allows teams to prioritize tests likely to expose failures early while still preserving diversity and cost-awareness. For release decisions, a separate DeploymentConfidence score aggregates the breadth and recency of successful controls, including test pass quality, scanner cleanliness, canary health, rollback readiness, and model confidence. Releases are blocked only when fused risk exceeds policy thresholds and available evidence fails to offset that risk sufficiently.

Governance is implemented through three control loops. The first is the engineering loop, where developers receive prioritized actions during coding and review. The second is the release loop, where change managers and platform services evaluate deployability under policy. The third is the learning loop, where model owners and reliability engineers assess false positives, false negatives, drift, and unintended behavioral consequences. This tri-loop structure reduces the tendency of DevSecOps programs to over-invest in scanning volume while under-investing in decision quality.

An additional benefit of the model is exception transparency. In regulated or high-availability domains, teams sometimes accept residual risk for urgent releases. Under the proposed approach, every override must record the dominant risk factors, mitigating controls, decision owner, expiry horizon, and follow-up actions. Those exception records become training data for later policy refinement. In this way, governance does not obstruct automation; it enriches it.

Table 3: Suggested Evaluation Metrics Across the Lifecycle

Lifecycle Stage	Representative Metric	Why It Matters	Decision Use
Pre-commit / review	Risk calibration, precision at top-k	Focuses review effort on the most consequential changes	Selective review depth and test expansion
Build / test	APFD, execution cost, severe issue	Balances early fault exposure with	Regression ordering and

	escape rate	pipeline speed	gate strictness
Release	Change failure rate, canary health, rollback readiness	Measures deployment confidence	Promotion, pause, or rollback
Operations	MTTD, MTTR, anomaly precision, blast-radius containment	Assesses runtime detectability and resilience	Incident escalation and remediation
Learning loop	False-positive rate, drift rate, override rate	Shows whether models and policies remain trustworthy	Retraining and threshold tuning

6. Evaluation Blueprint and Illustrative Deployment Scenarios

A rigorous empirical program is needed to validate the framework in practice. The proposed evaluation blueprint has four phases. Phase 1 assesses predictive quality using historical code, issue, scanner, and incident data. Relevant metrics include AUC, F1-score, calibration error, false-negative rate for severe issues, and lead-time reduction in defect discovery. Phase 2 evaluates delivery impact through controlled rollout experiments that compare baseline pipelines with risk-aware orchestration. Key measures include test execution cost, escaped defects, release frequency, change failure rate, and mean time to detection. Phase 3 measures operational value by tracing the link between pre-release scores and post-release anomalies, incidents, or service degradations. Phase 4 examines governance outcomes, including override volume, policy adherence, and auditor interpretability.

Three deployment scenarios illustrate the breadth of the framework. In healthcare and pharmacy workflows, the framework can connect document-processing AI, secure microservices, inventory optimization, and prescription-critical business events so that releases affecting sensitive patient-facing services receive stronger evidence requirements and tighter runtime surveillance [18][31][40]. In financial platforms, it can align transaction-critical modules, core banking quality controls, and change-risk models to allocate regression depth according to monetary and compliance exposure [21]. In cloud-native data and analytics platforms, the same architecture can combine pipeline observability, dependency-aware failure propagation models, and business-process observability to reduce silent data failures and accelerate incident diagnosis [24][28][38].

Importantly, the blueprint supports both centralized platform teams and federated product organizations. Centralized teams can own feature pipelines, model governance, and policy baselines, while domain teams customize thresholds, business criticality mappings, and remediation playbooks. This balance preserves standardization without sacrificing contextual relevance.

7. Discussion

The proposed framework suggests a shift in how DevSecOps maturity should be measured. Tool adoption alone is insufficient. A mature AI-driven DevSecOps program is one in which evidence is shared, risk is fused across stages, automated recommendations are explainable, and runtime outcomes continuously reshape upstream engineering work. The practical implication is that organizations should invest less in isolated dashboards and more in evidence models, feature governance, and policy design.

- The framework also clarifies an important boundary: AI does not replace engineering judgment. Instead, it raises the quality and timeliness of that judgment by concentrating attention on the most consequential changes, tests, services, and incidents. This makes the approach suitable for domains where accountability remains non-negotiable.

8. Threats to Validity

This paper is conceptual and framework-oriented, so its primary limitation is the absence of a full-scale empirical validation within a single industrial dataset. Although the design is grounded in recent literature, model effectiveness, threshold calibration, and governance cost will vary across sectors, architectures, and data maturity levels. A second limitation is that observability quality depends heavily on instrumentation completeness; sparse telemetry can weaken the feedback loop. A third limitation is that business criticality mappings may be subjective unless supported by stable organizational policy.

9. Conclusion

This paper presented an end-to-end AI-driven DevSecOps framework for risk-aware testing, monitoring, and lifecycle optimization. The framework unifies predictive quality signals, continuous security evidence, topology-aware impact analysis, runtime observability, and policy-backed governance into a single control model spanning planning through production operations. Its main value lies not in any single algorithm but in the disciplined coupling of estimation, decision support, enforcement, and learning.

As software systems become more distributed, data-intensive, and operationally consequential, the ability to connect pre-release assurance with post-release evidence will become central to both reliability and security. Future work should implement the framework in longitudinal industrial case studies, evaluate model and policy co-evolution over time, and

investigate how generative AI can assist rationale generation, remediation synthesis, and developer-centered explanation without undermining governance discipline.

References

[1] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: a systematic review," Information and Software Technology, vol. 141, p. 106700, 2022. https://doi.org/10.1016/j.infsof.2021.106700.

[2] Myrbakken, H., & Colomo-Palacios, R. (2017). DevSecOps: A multivocal literature review. Communications in Computer and Information Science, 770, 17–29. https://doi.org/10.1007/978-3-319-67383-7_2

[3] T. Myrbakken and R. Colomo-Palacios, "Security as culture: a systematic literature review of DevSecOps," ACM Computing Surveys, vol. 54, no. 2, 2021. https://doi.org/10.1145/3387940.3392233.

[4] National Institute of Standards and Technology, "Implementing DevSecOps Practices for a CI/CD Pipeline Using a Microservices Architecture," NIST Special Publication 800-204C, 2022. https://doi.org/10.6028/NIST.SP.800-204C.

[5] Gunda SK, Yettapu SDR, Bodakunti S, Bikki SB. Decision Intelligence Methodology for AI-Driven Agile Software Lifecycle Governance and Architecture-Centered Project Management, 2023 Mar. 30;4(1):102-8. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I1P112.

[6] Zhang, J. M., Harman, M., Ma, L., & Liu, Y. (2019). Machine learning testing: Survey, landscapes and horizons. IEEE Transactions on Software Engineering, 48(1), 1–36. https://doi.org/10.1109/TSE.2019.2942331

[7] I. Batool and T. A. Khan, "Software fault prediction using data mining, machine learning and deep learning techniques: a systematic literature review," Computers & Electrical Engineering, vol. 100, p. 107886, 2022. https://doi.org/10.1016/j.compeleceng.2022.107886.

[8] Gudi, S. R. (2023). Enhancing Reliability in Java Enterprise Systems through Comparative Analysis of Automated Testing Frameworks. International Journal of Emerging Trends in Computer Science and Information Technology, 4(2), 151-160. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I2P115.

[9] S. Shankar and A. G. Parameswaran, "Towards Observability for Production Machine Learning Pipelines," Proceedings of the VLDB Endowment, vol. 15, no. 13, pp. 4015-4022, 2022. https://doi.org/10.14778/3565838.3565853.

[10] Rajapakse, R. N., Zahedi, M., & Babar, M. A. (2022). Collaborative application security testing for DevSecOps: An empirical analysis of challenges, best practices and tool support. Journal of Systems and Software, 190, 111319. https://doi.org/10.1016/j.jss.2022.111319

[11] Gunda, S. K. G. (2023). The Future of Software Development and the Expanding Role of ML Models. International Journal of Emerging Research in Engineering and Technology, 4(2), 126-129. https://doi.org/10.63282/3050-922X.IJERET-V4I2P113.

[12] S. S. Alqahtani, "A study on the use of vulnerabilities databases in software engineering domain," Computers & Security, vol. 116, p. 102661, 2022. https://doi.org/10.1016/j.cose.2022.102661.

[13] Z. M. Zain, S. Sakri, and N. H. A. Ismail, "Application of Deep Learning in Software Defect Prediction: Systematic Literature Review and Meta-analysis," Information and Software Technology, vol. 158, p. 107175, 2023. https://doi.org/10.1016/j.infsof.2023.107175.

[14] Gupta, A. (2022). An integrated framework for DevSecOps adoption. International Journal of Computer Trends and Technology, 70(6), 19–23. https://doi.org/10.14445/22312803/IJCTT-V70I6P102

[15] Sivva SD, Thalakanti RR, Bandari SSG, Yettapu SDR. AI-Driven Decision Intelligence for Agile Software Lifecycle Governance: An Architecture-Centered Framework Integrating Machine Learning Defect Prediction and Automated Testing. 2023 Dec;4(4):167-72. Available from: https://www.ijetcsit.org/index.php/ijetcsit/article/view/554.

[16] Alsawalqah, H., Hijazi, N., Eshtay, M., Faris, H., Al Radaideh, A., Aljarah, I., & Alshamaileh, Y. (2020). Software defect prediction using heterogeneous ensemble classification based on segmented patterns. Applied Sciences, 10(5), 1745. https://doi.org/10.3390/app10051745

[17] C. Laaber, H. C. Gall, and P. Leitner, "Applying test case prioritization to software microbenchmarks," Empirical Software Engineering, vol. 26, p. 133, 2021. https://doi.org/10.1007/s10664-021-10037-x.

[18] Gudi, S. R. (2024). Design and Evaluation of Secure Microservices Architecture for HIPAA-Compliant Prescription Processing on AWS and OpenShift. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 5(2), 144-149. https://doi.org/10.63282/3050-9262.IJAIDSML-V5I2P116.

[19] R. van Dinter, C. Catal, G. Giray, and B. Tekinerdogan, "Just-in-time defect prediction for mobile applications: using shallow or deep learning?" Software Quality Journal, vol. 31, pp. 1281-1302, 2023. https://doi.org/10.1007/s11219-023-09629-1.

[20] M. Mahdieh, S.-H. Mirian-Hosseinabadi, and M. Mahdieh, "Test case prioritization using test case diversification and fault-proneness estimations," Automated Software Engineering, vol. 29, art. 50, 2022. https://doi.org/10.1007/s10515-022-00344-y.

[21] Gunda, Sai Kumar. "A Risk-Aware AI Framework for Automated Testing and Quality Assurance in Core Banking Systems." International Journal of Multidisciplinary Evolutionary Research, vol. 5, no. 1, 2024, pp. 117-120. https://doi.org/10.54660/IJMER.2024.5.1.117-120.

[22] K. Garg and S. Shekhar, "Optimizing test case prioritization through ranked NSGA-2 for enhanced fault sensitivity analysis," Innovations in Systems and Software Engineering, vol. 20, pp. 307-328, 2024. https://doi.org/10.1007/s11334-024-00561-6.
[23] V. Casola, A. De Benedictis, C. Mazzocca, and V. Orbinato, "Secure software development and testing: A model-based methodology," Computers & Security, vol. 137, p. 103639, 2023. https://doi.org/10.1016/j.cose.2023.103639.
[24] Mittamidi, V. K. R. (2024). An automated AI-driven monitoring and observability framework for cloud-based data pipelines by software defect prediction research. International Journal of Multidisciplinary Evolutionary Research, 5(1), 109-112.
[25] Naveed, H., Grundy, J., Arora, C., & Khalajzadeh, H. (2023). Runtime monitoring of human-centric requirements in machine learning components: A model-driven engineering approach. arXiv preprint arXiv:2310.06219. https://arxiv.org/abs/2310.06219
[26] Yettapu, S. D. R. (2023). A unified artificial intelligence governance and reliability engineering framework for secure and autonomous software-intensive and cyber-physical systems. Journal of Frontiers in Multidisciplinary Research, 4(1), 605-608. https://doi.org/10.54660/.JFMR.2023.4.1.605-608.
[27] Li, S., Guo, J., Lou, J.-G., Fan, M., Liu, T., & Zhang, D. (2022). Testing machine learning systems in industry: An empirical study. In Proceedings of the IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 263–272). https://doi.org/10.1145/3510457.3513036
[28] Mariani, L., Pezzè, M., Riganelli, O., & Xin, R. (2019). Predicting failures in multi-tier distributed systems. Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE).
[29] A. Bahaa, A. Abdelaziz, A. Sayed, L. Elfangary, and H. Fahmy, "Monitoring real time security attacks for IoT systems using DevSecOps: a systematic literature review," Information, vol. 12, no. 4, p. 154, 2021. https://doi.org/10.3390/info12040154.
[30] Azad, N., & Hyrynsalmi, S. (2022). DevOps challenges in organizations: Through professional lens. Lecture Notes in Business Information Processing, 463, 260–277. https://doi.org/10.1007/978-3-031-20706-8_18
[31] Moro, S., Cortez, P., & Rita, P. (2021). Automated data-driven approach for healthcare management using machine learning and intelligent systems. Journal of Biomedical Informatics, 117, 103735. https://doi.org/10.1016/j.jbi.2021.103735
[32] S. Nägele, J.-P. Watzelt, and F. Matthes, "Investigating the Current State of Security in Large-Scale Agile Development," in Agile Processes in Software Engineering and Extreme Programming (XP 2022), Lecture Notes in Business Information Processing, vol. 445, 2022, pp. 203-219. https://doi.org/10.1007/978-3-031-08169-9_13.
[33] H. Haverinen, T. Päivärinta, J. Vänskä, and H. Joutsijoki, "Information-Centric Adoption and Use of Standard Compliant DevSecOps for Operational Technology: From Experience to Design Principles," in Software Business (ICSOB 2023), Lecture Notes in Business Information Processing, vol. 500, 2024, pp. 400-415. https://doi.org/10.1007/978-3-031-53227-6_28.
[34] Balerao, M. (2023). A converged artificial intelligence architecture for innovation, software lifecycle optimization, and cybersecurity risk mitigation. International Journal of Multidisciplinary Futuristic Development, 4(1), 117-120. https://doi.org/10.54660/IJMFD.2023.4.1.117-120.
[35] T. Rangnau, R. van Buijtenen, F. Fransen, and F. Turkmen, "Continuous security testing: a case study on integrating dynamic security testing tools in CI/CD pipelines," in 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC), 2020. https://doi.org/10.1109/EDOC49727.2020.00026.
[36] J. Soldani, D. A. Tamburri, and W.-J. van den Heuvel, "The pains and gains of microservices: a systematic grey literature review," Journal of Systems and Software, vol. 146, pp. 215-232, 2018. https://doi.org/10.1016/j.jss.2018.09.082.
[37] Ghotra, B., McIntosh, S., & Hassan, A. E. (2017). Revisiting the impact of classification techniques on the performance of defect prediction models. Proceedings of the IEEE/ACM 39th International Conference on Software Engineering (ICSE) (pp. 789–800). https://doi.org/10.1109/ICSE.2017.76
[38] A. Saha, P. Agarwal, S. Ghosh, N. Gantayat, and R. Sindhgatta, "Towards Business Process Observability," in Proceedings of the 7th Joint International Conference on Data Science & Management of Data (11th ACM IKDD CODS and 29th COMAD), 2024, pp. 257-265.
[39] Alsaeedi, A., & Khan, M. Z. (2019). Software defect prediction using supervised machine learning and ensemble techniques: A comparative study. Journal of Software Engineering and Applications, 12(5), 85–100. https://doi.org/10.4236/jsea.2019.125007
[40] Galli, L., Levato, T., Schoen, F., & Tigli, L. (2021). Prescriptive analytics for inventory management in health care. Journal of the Operational Research Society, 72(10), 2211–2224. https://doi.org/10.1080/01605682.2020.1776167