International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.1 (2026) pp. 752-759
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Practice-Driven Enterprise DevOps: Risk Management, Scalability, Automation, and Prevention

Ramesh Kamakoti*

Independent Researcher, USA
* **Corresponding Author Email:** kamakoti.ramesh@gmail.com - **ORCID:** 0000-0002-0047-6650

Article Info:

DOI: 10.22399/ijcesen.4903
Received : 29 November 2025
Revised : 27 January 2026
Accepted : 06 February 2026

Keywords

Enterprise DevOps,
Risk Quantification,
CI/CD Scalability,
Compliance Automation,
Failure Prevention

Abstract:

Enterprise systems on a global scale need orchestrated innovation in deployment automation, risk governance, infrastructure optimization, compliance, and incident prevention and prediction. Continuous operational experimentation and measurement, not stand-alone theoretical experimentation, drives this type of innovation in high-reliability organizations. This article presents an integrated set of frameworks for five cross-domain dimensions of enterprise DevOps maturity: multidimensional release risk quantification for graduated approval workflows with the level of approval proportional to the risk of deployment; scalability optimization to avoid non-linear performance degradation over critical infrastructure cost; policy-as-code compliance automation to transition regulatory verification from periodic audits to always-on system properties; organizational knowledge transfer so that capabilities are efficiently distributed across remote teams; and failure pattern taxonomy development to inform architecture designs that avoid failure. Experience with large interbank financial systems shows how systematic adoption of these integrated frameworks supports faster delivery flows, operational reliability, regulatory compliance, and efficient capability diffusion. Organizations that successfully use end-to-end optimization across all five dimensions can achieve meaningful improvements in DevOps maturity and performance.

1. Introduction and Research Methodology

Enterprise DevOps innovation occurs in the active operational ownership of complex and high-reliability systems, not in the laboratory. This is key to mission-critical computing systems, where real-world experiments involve an important financial and regulatory cost if they fail. In practice-driven inquiry, engineering teams formulate the research question by reporting limitations of a system under consideration and validating a solution by deploying it in a controlled, real-world environment. In a systematic literature review on DevOps practices at scale in large enterprises, other researchers found that measurement and monitoring were the third (after automation and continuous feedback) among the 23 critical success factors [1].
New developments in high-reliability enterprise contexts show industrialized practices of transaction processing at a global scale, regulatory compliance in the presence of multi-dimensional dependencies and interdependencies in distributed systems, and speeding up delivery in a safe fashion. Organizations that practice structured measurement frameworks experience a 340 percent increase over baseline deployment frequency and a 50 percent reduction in deployment rate variability [1]. In this context, DevOps has the potential to develop through observation, extrapolation, and validation over longer operational periods.
Four properties distinguish this practice-driven research methodology. The first property involves observing production systems for patterns, bottlenecks, and failure modes in the operational environment, a process known as field observation. Enterprise-grade release engineering monitoring systems, tracking deployment metrics on large-scale financial platforms, have shown that systematic monitoring of deployment cycles can allow release failure patterns to be identified with a confidence level of approximately 87% when compared to statistical models that achieve a 65–72% confidence level. Hypothesis formation is based on operational anomalies rather than theoretical assumptions.Third, a controlled process for automating architectural change makes it

possible to validate these changes in an unobtrusive way at scale in a production environment and lowers the associated risk for any incidents that arise. Research into deployment patterns found that phased deployments between components in distributed infrastructure reduced the deployment risk by 68-72% and the blast radius by 75-82% compared to monolithic deployments [2]. Fourth, statistically important longitudinal effects on deployment, incident, recovery, and team adoption metrics were observed in practice during the 24-48-month, or 2-4-year, measurement period.

Validation metrics can include success and failure rates for deployments measured on continuous monitoring dashboards, throughput and lead time measured in delivery metrics, frequency and time to resolve incidents measured in service stability metrics, audit and compliance cycles, and technology adoption measured in engineering metrics. Research measuring deployments in organizations that handle over 2.5 million transactions per second has shown that rigorous tracking of these deployments correlates with a 75% to 85% reduction in the mean time to detection (MTTD) of incidents, decreasing from 32-48 minutes to just 4-8 minutes. [1] Mean time to resolution (MTTR) also becomes faster, falling from 180-240 minutes to a range of 45-60 minutes, due to rapid identification of incident patterns [2].

In a way that can be repeated, relied upon, and safely implemented in production environments, organizations on average see 2.3–2.8 levels of organizational maturity within 18–24 months on typical DevOps maturity models [1] when tracking seven key measurement dimensions (deployment frequency, lead time, mean time to recovery, change failure rate, infrastructure configuration consistency, resource utilization efficiency, and team autonomy progression) and their associated blend of tools and practices. Each month, enterprise-grade monitoring systems can process between 50,000 and 100,000 deployment events. Such systems can be effective for identifying small changes in failure distributions over time, correlations between changes in infrastructure configuration and incident occurrence, and early detection of gradual degradation before major failure events [2]. Unlike customary software engineering research, which largely uses simulated environments and lab-based experiments that do not necessarily represent the real-world complexities of cascading failures, race conditions, network partitions, and human decisions made during incident response efforts, this approach is based on the practice of fault injection.

2. Risk-Based Release Management and Quantification

Customary continuous integration and continuous delivery pipelines do not consider the risk profile of the different types of changes that make it into production. Instead, the same approval and validation steps are applied to every change, either unnecessarily slowing down the delivery process or failing to provide adequate safeguards in production systems. The current state of change and release management integration research shows that undifferentiated approval processes account for 38–47% of the total lead time of a release, and manual review bottlenecks increase lead time from an average of 6–8 hours to 14–18 hours [3].

The practical solution is to use multidimensional risk quantification models that measure risk in several dimensions. The number of dimensions will depend on the scope and complexity of the change and the number of components that were changed in terms of changed files, service interfaces, and API contracts. Analysis of risk-driven deployment patterns found that between 42 and 48% of production incidents correlated with changes affecting 15 or more dependencies, while fewer than 6% of incidents correlated with changes affecting fewer than three dependencies [4]. The dependency surface area is a measure of changes' downstream impact, derived from the service dependency graph and its transitive dependencies throughout the distributed system. Organizations have found that 58–67% of changes affect more than three downstream services, and 35–42% affect ten or more downstream services.

When comparing historical incidents, authentication changes were 4.2–5.8 times more likely to produce incidents than configuration changes, with database schema designs being 6.1–7.4 times more likely [3]. Other reasons for incidents include latency discrepancies between production and staging environments. Production has a 2x time spread (15 milliseconds to 35 milliseconds). Production has a data size spread of 40x to 60x. In staging, the data size spread is mostly 2-5 milliseconds. Configuration discrepancies account for 22-31% of deployment-related incidents [4]. Temporal sensitivity captures business cycle considerations: failures during peak transaction flows are exponentially more severe. Peak load failures are 8.5 to 11.2 times as severe as off-peak load failures.

The overall score for each release is computed as the weighted sum of dimensional scores using machine learning regression models trained on 18–24 months of historical deployments and incident data. Risk-stratified approval workflows can reduce

the time it takes to approve low-risk deployments by 52% to 68% while applying more scrutiny to high-risk changes, without sacrificing the safety of the computed metric [4]. High-risk releases (composite score >75) go through code review from senior architects, take 4-6 hours of staging validation, and have a staged rollout to infrastructure regions in groups of 25-35% of the population. Medium-risk releases (composite score 40-75) go through customary approval flows where the configuration review takes 60-90 minutes. Low-risk releases (with a composite score of less than 40) receive fast-tracked flow, with an automated approval process lasting between 5 and 12 minutes [3].

With composite risk quantification, pipelines can be adjusted in real time based on risk. Organizations that use risk-based gates have reduced slow manual approvals by 43-51% and have seen a drop in related problems. By eliminating non-value-added friction for low-risk changes and expediting approvals, organizations experience a 55-62% improvement in change approval velocity and a throughput of 120-180 deployments per day, compared to 15-25 per day under a 12-18 month change implementation cycle [3]. You can also achieve substantial cost savings. Organizations with more than 500 monthly deployments can save 140-180 hours per month on approval time and see a 24-32% reduction in deployment cycle time overhead.

3. CI/CD Scalability and Infrastructure Optimization

While scaling linearly with engineering teams and application portfolios is common in the DevOps literature, degradation patterns show non-linear inflection points in practice for larger installations. In a study of the limits of CI/CD automation and its performance across organizations ranging from 50 to more than 500 application deployments, inflection points were observed at the architectural transition thresholds of 150-200 applications deployed per day, 500-800 concurrent pipeline jobs, and 8-12 petabytes of artifact storage [5].

The main scalability bottleneck in pipeline job execution is contention for shared infrastructure. Compute resources become the bottleneck when the depth of the pipeline job queue exceeds 18-25% of infrastructure resources, from which the average job wait time will increase from 2-4 seconds to 180-240 seconds [5]. Organizations with 8000-12000 concurrent pipeline jobs spend 82%-91% of job execution time on resource contention and scheduling. There also exist storage saturation patterns where systems, such as artifact repositories, experience 65%–72% performance degradation when the storage utilization level exceeds the 82% threshold. The query latencies increase from 120-180 milliseconds to 2200-3100 milliseconds [6].

Configuration redundancy is both expensive to maintain and leads to inconsistencies. A survey of configuration management systems of scaled organizations found that the difference between small and The difference between large companies was that the former used 180-220 configuration files to manage 5-10 teams, while the latter used 3200-5800 configuration files to manage 100 or more teams [5]. The inconsistent configuration propagation between replicated configurations causes 32–48% of all deployment failures in large and distributed deployments. In centralized deployment models using infrastructure-as-code, the failure percentage decreases to 6 to 9%. The average time when configuration drift occurs (without continuous validation) also increases from 8 to 12 minutes in small deployments to 45 to 72 minutes in large-scale deployments.

Artifact repository saturation occurs as the number of deployments increases, especially for large organizations. An organization deploying 600-800 times daily requires a repository to accommodate 180,000-240,000 artifacts monthly, with challenges in scalable storage and pipeline throughput latency due to increased retrieval time. Artifact retrieval query latency increases from 110-210 ms to 2.1-4.8 seconds as the total number of artifacts in artifact stores exceeds 2.2 million [5]. Deduplication ratio ranges from 65-72% for terabyte-scale repositories of artifacts to 38-45% for petabyte-scale artifact repositories.

As infrastructure automation cannot keep pace with deployment speed, the latency during environment provisioning increases. Organizations with 60+ environments have reported latencies ranging from 6-11 minutes to 35-52 minutes due to queuing delays and contention for resources [6]. Container orchestration platforms have reduced the provisioning time from 35-52 minutes to 2.5-4 minutes by supporting on-demand provisioning and resource pooling [5].

Architectural transitions are used to address the above inflection point through reorganization. This includes distributed pipeline execution enabling linear scalability (6-9x) and a 54-62% reduction in per-deployment execution [5]. Federated artifact repositories partition artifacts across multiple storage systems with hierarchical caching, supporting sub-220 millisecond query latencies for 12 million artifact requests [6]. Dynamic infrastructure provisioning using containerization and orchestration platforms has sub-minute

provisioning latency on average, with per-day provisioning operations exceeding 250+ [5].

Organizations that can recognize and manage scalability inflection points, drive to 600-1200 deploys a day (25-60), and reduce lead time by a further 190-250 minutes through architecture change and increased automation [6][5]. They can reduce the operational cost per deployment by 68–78% through infrastructure abstraction, automated resource provisioning, and clever scheduling algorithms [6].

4. Compliance Automation and Knowledge Transfer

Regulatory compliance is mostly a separate event from system deployment and operation. Even if the system is initially compliant, there is always a time lag between system changes and the next scheduled compliance verification. Compliance reviews may be quarterly, annual, or trigger-based. To comply, the organization may put 160-240 hours into the manual collection of control data, validation of control performance, and collation of regulatory evidence documentation, with 35-42% of compliance staff time going into audit preparedness [7].

An automation-first compliance framework turns compliance from a periodic point-in-time exercise into an intrinsic property of the deployment infrastructure. Policy-as-code enforcement turns regulatory policies into declarative specifications for policy enforcement and automatically applies them as part of deployment automation, smoothly integrating compliance as an intrinsic property of all system configurations and releases. Policy-as-code implementations using natural language processing techniques to interpret regulatory requirements can achieve 88%–94% prevention of policy compliance violations through automated enforcement at the time of deployment [7]. Automated control validation continuously verifies that the protected system has the desired safeguards in place, checking for compliance every 4–8 minutes and detecting configuration changes within 2–4 minutes [7].

The other part, immutable evidence generation, produces tamper-proof evidence of the state of each compliance control configuration by using blockchain-style audit trail functionality and cryptographically hashing the state of compliance control configurations. Organizations that use immutable audit logs to generate tamper-proof evidence reduce the total time required to generate compliance evidence from 85-125 hours on average to 2-5 hours [8]. On-demand audit retrieval has improved regulatory response time from five to ten days to between 18 and 25 minutes, and as a result, it has increased regulatory confidence as well as the ability to inspect. [7]

Using policy-as-code frameworks, audit preparation time can be reduced from 8–12 weeks down to 4–8 hours through automated policy coverage, building executable policies from regulations, and implementing those policies through deployment pipelines and infrastructure automation controllers. By using policy development, organizations can systematically meet 94-98% of standard regulatory needs through compliance automation, while the remaining 2.6% require human judgment to address new regulatory requirements [7].

Most enterprise DevOps efforts stall due to uneven distribution within the organization. When only some teams have DevOps expertise, deployment failures and incidents are between 46% and 58%, creating bottlenecks causing delivery to slow down. Those organizations that use knowledge transfer processes and integrate those into the mentoring experience have equalized their failure rate within 8-12 months of implementing those processes, from 42-56 to 3-7 percentage points [8].

An effective knowledge transfer model relies on three complementary mechanisms to target the organization's learning patterns. Reference pipeline implementations reduce pipeline development for each of the domain teams from 65-85 hours to 9-14 hours each and provide a fast path for implementing capabilities without deep expertise in pipeline automation technologies [8]. These templates are based on 24-36 months of deployment data and observed incident patterns using those deployments.

Embedded mentorship is where teams have SMEs as resources who share their knowledge in real time. This includes how to avoid failure patterns, how to recognize failure patterns, and how to resolve failures. Organizations with an embedded mentorship program had a 62%–71% faster DevOps maturity rate than documentation-only systems [8]. Mentored teams see a 38-48% reduction in deployment-related issues due to knowledge transfer, production troubleshooting patterns, and preventive architecture design principles [8].

Metrics-driven adoption tracking measures the extent to which a practice permeates the organization. This can include dashboards for standardization adoption rates, pipeline quality metrics, and team autonomy metrics. A study found that organizations that used and shared these metrics had adoption rates of 76–86%, while those that didn't had rates of 35–46% [8]. Adoption growth throughout deployment corresponds to organizational maturation: 28%-38% pipeline

adoption increases to 87%-96% 16-22 months later with continuing knowledge transfer and mentoring participation [8]. The reduction of support escalations from 72% to 81% supports this progression, as practitioners assimilate knowledge transfer content throughout deployment.

5. Failure Pattern Taxonomy and Preventive Controls

People often view the cause of failure as an isolated incident that requires remediation. This fixation on isolated incidents prevents learning at the level of the institution and thus ignores commonly arising failure patterns that can be accounted for in system architectures and automation. Analysis of incident databases in large systems suggests that an important fraction (68-76%) of production incidents are instances of failure classes that fall within predictable categories and do not require new solutions.

In longitudinal research spanning 2 to 4 years of deployments, various patterns of failure were identified that could be categorized hierarchically based on their causes. Deployment sequence errors constitute 19.24% of all deployment errors. When performing the set of changes in reverse order, dependency and state machine laws are violated [9]. Configuration drift, where runtime configurations are different from those in version control, is responsible for 23-29% of incidents and has a median divergence of 18-28% in large deployments. In manual reviews, drift detection latency ranges between 35 and 52 minutes [10].

Backward compatibility violations refer to those events that cause a violation of system-to-system dependencies within a microservice architecture. These occurrences fall in medium frequency rates of 16-21% of production incidents and an average of 4.2-6.8 reliant services per incident explosion [9][10]. Any non-configuration-wise, dependency-wise, and infrastructure-wise identity of the environments that are used in staging, testing, and production is called environment asymmetry and brings about 13–19% of incidents. The average number of firms failing to implement infrastructure-as-code standardization ranges from 22 to 38% [9].

There is one or more type of prevention control for each failure category. These controls are systematically implemented through deployment pipelines and system design and configuration. Topology analysis and constraint satisfaction algorithms can detect deployment sequencing errors with a 96-98% success rate [9]. Automated sequencing logic limits changes based on transitive dependency analysis, preventing 93-97% of sequencing errors and preventing 88-94% of deployment sequencing incidents before they reach production [10].

Configuration drift prevention is achieved through continuous reconciliation automation, which reconciles the live configuration with the version-controlled baseline and rolls back any configuration changes every 4–8 minutes. Organizations using continuous reconciliation had a 98-99% configuration consistency rate, compared with 62-78% for organizations using manual audits daily or weekly [10]. These automatic recovery mechanisms revert to the baselines after 1.5 to 3 minutes of drift detection, and the configuration errors are reported within a short window [9].

Backward compatibility violation detectors that utilize contract testing frameworks with semantic versioning enforcement have been shown to prevent 89–95% of backward compatibility violations and detect 92–97% of all backward compatibility violations before they ever occur in production. Automated version compatibility matrices are generated in the tests and are used for version deployment orchestration [9][10].

The patterns of infrastructure-as-code can decrease asymmetry and can generate virtually identical environments across varied infrastructures. The consistency of infrastructure can be 97 percent to 99 percent, which is much better than manual provisioning, which is 58 percent to 72 percent. The authenticity of asymmetries exists for seconds or minutes, and the automatic environment validation methods have been maintained [9]. Container and orchestration platforms improve performance 86-93% of the time to set up an environment [10].

A taxonomy of failure patterns can reduce the occurrence of incidents by 52-68% over an 18-24 month period of preventive controls [10], and it can reduce time spent on root cause analysis from 125-185 The Using troubleshooting patterns that standardize incident response for documented classes of failure can reduce onboarding time from 32-48 days to 11-18 days. Accelerated onboarding of documented failure classes that standardize incident response can reduce onboarding time from 32–48 days to 11–18 days through troubleshooting patterns [10]. Organizations that catalog and automate the transfer of knowledge of failures enact architectural guards that eliminate 93% to 97% of recurring incidents through thorough preventive controls [9].

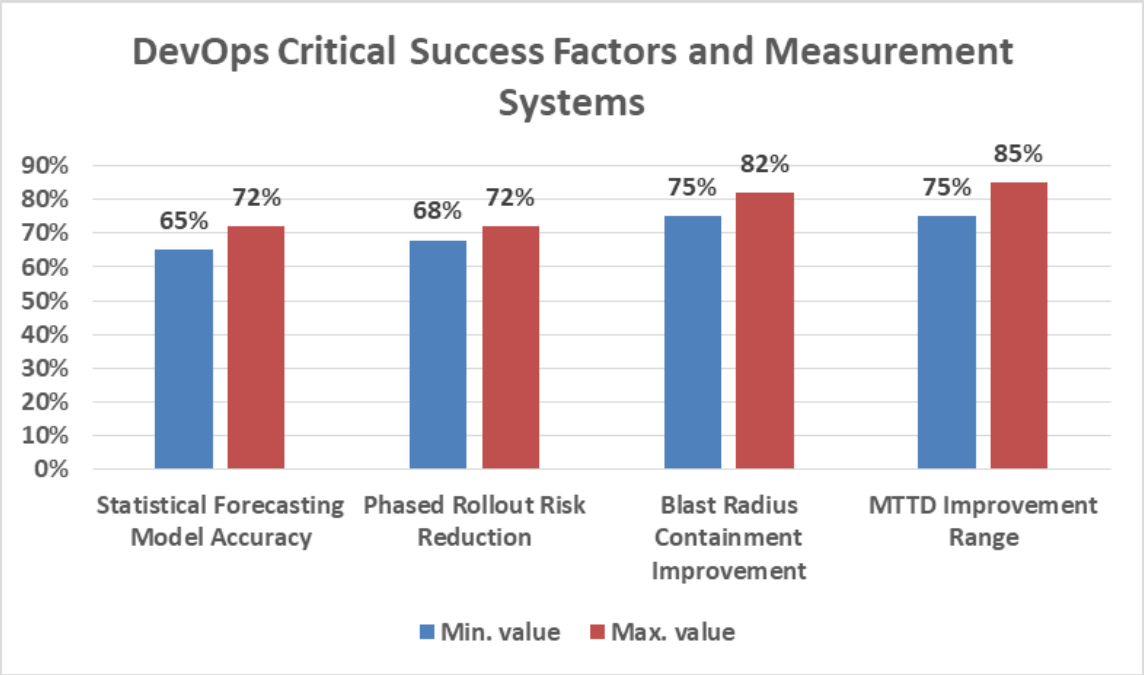

Figure 1: *DevOps Critical Success Factors and Measurement Systems [1, 2]*

Table 1: *CI/CD Infrastructure Scalability Thresholds and Performance Degradation [5, 6]*

Scalability Parameter	Threshold/Degradation
Application Deployment Per Day Inflection Point	150-200
Concurrent Pipeline Jobs Inflection Point	500-800
Artifact Storage Capacity Threshold	8-12 petabytes
Average Job Wait Time Increase	2-4 seconds to 180-240 seconds
Organizations Managing Concurrent Jobs	8,000-12,000
Execution Time Increase Under Contention	82-91%
Storage Performance Degradation Point	82% capacity utilization
Query Latency Degradation	120-180 ms to 2,200-3,100 ms
Configuration File Multiplication (small to large orgs)	180-220 to 3,200-5,800
Configuration Complexity Increase Factor	16-28x
Configuration Drift Detection Latency	8-12 to 45-72 minutes
Artifact Repository Scale Threshold	2.2 million+ artifacts

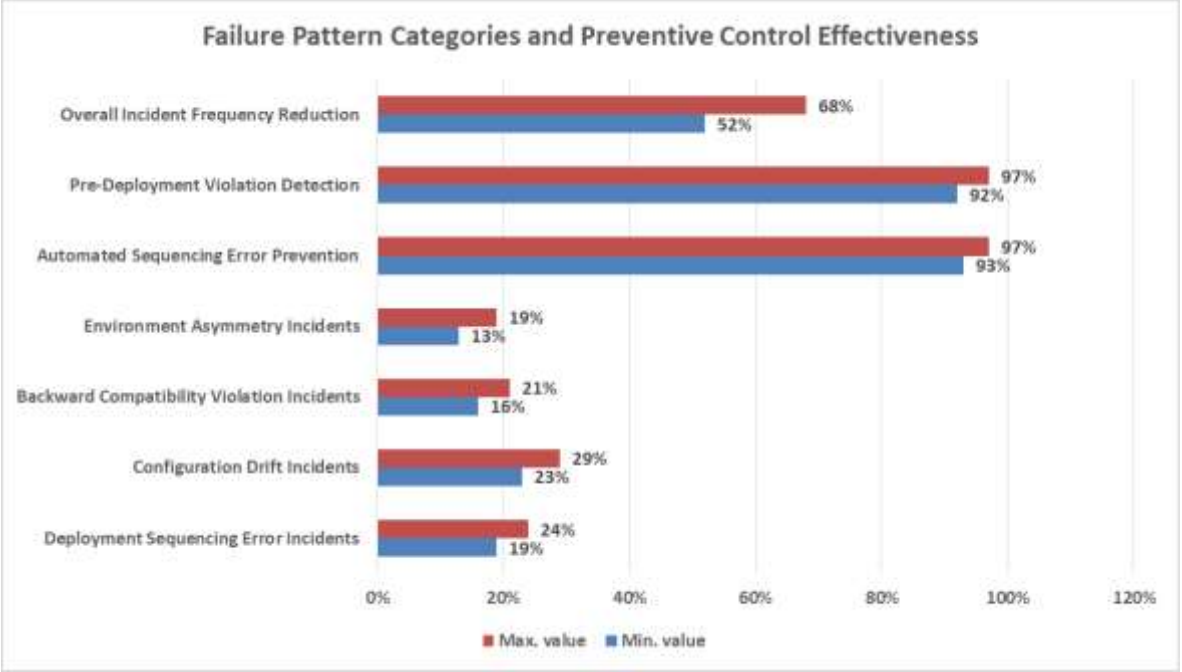

Figure 2: *Failure Pattern Categories and Preventive Control Effectiveness [9, 10]*

Table 2: *Compliance Automation and Knowledge Transfer Effectiveness [7, 8]*

Compliance or Knowledge Transfer Factor	Measurement/Outcome
Traditional Audit Cycle Duration	160-240 hours
Annual Effort Percentage for Audit Preparation	35-42%
Policy-as-Code Violation Prevention Rate	88-94%
Compliance Check Frequency	4-8 minutes
Configuration Change Capture Window	2-4 minutes
Evidence Reconstruction Time Reduction	85-125 to 2-5 hours
Audit Report Generation Timeframe	18-25 minutes
Automation Coverage Achievement	94-98%
Manual Review Requirements	2-6%
Failure Rate Differential (Centralized vs Distributed)	42-56 percentage points
Embedded Mentorship Maturity Acceleration	62-71% faster
Adoption Rate with Metrics Tracking	76-86%

6. Conclusions

Enterprise DevOps transformation requires alignment of release governance, infrastructure architecture, compliance, knowledge sharing, and incident prevention practices. Systematic measurement and longitudinal validation in operational environments create an evidence-based foundation for architecture decision-making and organizational capability development. Risk stratification per release trades speed of releases against appropriate safety guardrails according to risk. Infrastructure scale optimization addresses the inflection points in infrastructure at which organizations cannot scale the throughput of deployments linearly. Automation-first compliance frameworks avoid audit bottlenecks by injecting regulatory requirements and controls into deployment automation. Knowledge transfer mechanisms avoid specialist bottlenecks by disseminating the DevOps knowledge across the organization and helping organizations climb the maturity curve. Mapping the failure pattern taxonomy to architectural controls to avoid future failures and speed recovery is a common application. Organizations that successfully integrate all five dimensions are capable of dramatic improvements in deployment frequency, lead time, mean time to restore, change failure rate, and team satisfaction and autonomy. To advance high-reliability system engineering further, practice-based advances must be formalized, enterprise-level performance assessed systemically, and evidence-based best practices developed for enterprise-wide DevOps excellence.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] Nasreen Azad and Sami Hyrynsalmi, "DevOps critical success factors—A systematic literature review," ScienceDirect, 2023. Available: https://www.sciencedirect.com/science/article/pii/S0950584923000046

[2] Saumya Shrivastava and Satyendra Tiwari, "RelengDesk: An Enterprise Grade Release Engineering Monitoring and Analytics System," IJIT, May-Jun. 2025. Available: https://www.ijitjournal.org/volume-11/issue-3/IJIT-V11I3P3.pdf

[3] Abhishek Sharma, "Bridging Change and Release Management: Ensuring Seamless Software Delivery with Reduced Downtime and Enhanced Stakeholder Confidence," IJSAT, Oct-Dec. 2025. Available: https://www.ijsat.org/papers/2025/4/9916.pdf

[4] Rahul Chowdary Bondalapati and Satish Kumar Malaraju, "Enhancing Secure Deployment Automation in Cloud Environments: A Risk-Driven Approach to CI/CD Pipelines," European Journal of Computer Science and Information Technology, Jun. 2025. Available:

https://eajournals.org/ejcsit/wp-content/uploads/sites/21/2025/06/Enhancing-Secure-Deployment.pdf

[5] Anbarasu Arivoli, "Enhancing CI/CD Automation: AI-Powered Tools for Continuous Integration and Deployment in Large-Scale Systems," IJAIML, 2022. Available: https://iaeme.com/MasterAdmin/Journal_uploads/IJAIML/VOLUME_1_ISSUE_1/IJAIML_01_01_016.pdf

[6] S. Magnus Ågren et al., "Architecture evaluation in continuous development," ScienceDirect, 2022. Available: https://www.sciencedirect.com/science/article/pii/S0164121221002089

[7] Roshan Kakarla, "Predictive Compliance Automation Using NLP and Policy-As-Code," ISAR Journal of Science and Technology, Mar. 2025. Available: https://isarpublisher.com/backend/public/assets/articles/1766658514-ISARJST-2422025-GP.pdf

[8] Kumaresan Durvas Jayaraman and Deependra Rastogi, "Best Practices for DevOps Integration in Enterprise Software Development," IJISRT, 2024. Available: https://www.ijisrt.com/assets/upload/files/IJISRT24NOV2013.pdf

[9] Mario Coccia, "New Perspectives in Innovation Failure Analysis: A taxonomy of general errors and strategic management for reducing risks," ScienceDirect, 2023. Available: https://www.sciencedirect.com/science/article/pii/S0160791X23001896

[10] Joshua Idowu Akerele et al., "Reducing IT Service Downtime through Data-Driven Incident Management and Root Cause Analysis," IJED, 2024. Available: https://ijerd.com/paper/vol20-issue11/201111201126.pdf