Article

Not peer-reviewed version

IDEAL-Enhanced DevOps: A Structured Framework for Continuous Improvement in Software Engineering

Mohammed Nazeh Alimam * and Sami Kudsi *

Posted Date: 14 March 2025

doi: 10.20944/preprints202503.1031.v1

Keywords: DevOps; IDEAL model; continuous improvement; automation; software engineering

 doi:10.20944/preprints202503.1031.v1

Article

IDEAL-Enhanced DevOps: A Structured Framework for Continuous Improvement in Software Engineering

Mohammed Nazeh Alimam [1] and Sami Kudsi [2]

[1] UE University of Applied Science, Iraq

[2] UE, Palestine

* Correspondence: mohammednazeh.alimam@ue-germany.de (M.N.A.); sami.kudsi@ue-germany.de (S.K.)

Abstract: Recent advances in DevOps have dramatically reshaped software development and operations by emphasizing automation, continuous integration/delivery, and rapid feedback. However, organizations still struggle to achieve predictable improvements despite widespread adoption. In this study, we propose an "IDEAL-Enhanced DevOps" framework that integrates the five-phase IDEAL model—Initiate, Diagnose, Establish, Act, and Learn—into a DevOps transformation process. The proposed method lays out a structured approach to applying incremental improvements throughout the software delivery process. By a review of literature, in-depth analysis of case studies, and dis-semination of a questionnaire to practitioners in the field, this research explains how the IDEAL stages can be mapped to key processes in DevOps, address automation and scalability challenges, and facilitate a learning-centered, cooperative culture. The results show that a well-defined process-improvement approach can effectively reduce error incidence, enhance usability of tools, and significantly shorten time to get products to market. Our analysis shows that coupling IDEAL with DevOps not only clarifies responsibilities and organizational roles, but also lays a foundation for more resilient, high-quality, and adaptable software engineering methods.

Keywords: DevOps; IDEAL model; continuous improvement; automation; software engineering

1. Introduction

Software development has witnessed revolutionary change in the last decade with the advent of DevOps practices that combine development and operations to produce software faster and more reliably. Yet, even with the extensive use of DevOps in most industries, most organizations still struggle with issues like siloed processes, tangled tooling, and absent feedback mechanisms. To solve these problems, process-improvement models like the IDEAL model—with Initiate, Diagnose, Establish, Act, and Learn as its phases—have been successfully used in quality improvement programs. But the use of the IDEAL model in DevOps has been an untapped field.

Whereas frameworks such as PDCA and Six Sigma offer iterative improvement approaches, they do not offer the explicit alignment with the rapid feedback loops and automation-based focus of DevOps. IDEAL, with its formalized phases (Initiate, Diagnose, Establish, Act, Learn), offers a more flexible and scalable model aligned to the iterative and automation-based DevOps environment.

The goal of this research is to propose a new and structured solution—IDEAL-Enhanced DevOps—that combines the iterative enhancement approach of IDEAL with the continuous integration and delivery process. Through their combination, we intend to present an end-to-end, step-by-step guide that is intended to assist organizations in avoiding common DevOps pitfalls such as slow feedback loops, high error rates, and suboptimal automation strategies. In this paper, we discuss the literature of relevance to DevOps and the IDEAL model, outline our design for our broader framework, and share results of case study research and surveying industry practitioners.

 doi:10.20944/preprints202503.1031.v1

Finally, our goal is to demonstrate that a structured process improvement framework can be instrumental in making DevOps transformations more effective and efficient.

2. Materials and Methods

2.2.1. Overview of Toolchain Selection and Implementation

In a unified attempt to offer support that is generally adaptable to diverse forms of infrastructures—cloud-native setups, hybrids, as well as legacy infrastructure—IDEAL-Enhanced DevOps has been developed with utmost attention to be completely agnostic to every toolchain in particular. This unique agility enables organizations to configure and shape practices in automation not only in accordance with their present tech stack but in accordance with each organization's degree of organizational maturity in addition to considering each organization's multitude of compliance requirements to be met. This process used to select an ideal toolchain is organized in an organized manner in consideration to only two key deciding factors:

1. *Assessment of Technological Constraints (Diagnose Phase):* Organizations evaluate their current infrastructure—whether legacy systems, hybrid environments, or cloud-native architectures—to determine appropriate automation and modernization strategies.
2. *Selection of Implementation Routes (Establish Phase):* After detailed and extensive evaluation, companies have to take the key decision between adopting two fundamentally different routes: incrementally automated in accordance with their current legacy systems, or otherwise adopting superior and advanced optimisations that are cloud-native in design.

This organized and systematic process ensures that the IDEAL framework is always valid and efficient across an extensive variety of levels of DevOps maturity. This flexibility allows an automated process to be created that is bespoke to cater to exact requirements, in preference to adopting an off-the-shelf, prescriptive process.

2.2.2. Version Control and Source Code Management

Version control systems have an imperative role in providing traceability and audibility in software while facilitating collaborative development among various development teams. Considering that enterprises have diverse levels of adaptation to cloud-native systems in comparison to self-hosted systems, making version control practices adaptable is imperative to cater to these differences.

- Cloud-native organizations utilize GitHub for seamless integration with cloud-based CI/CD and serverless pipelines.
- Hybrid and on-premise organizations prefer to go with either GitLab or Bitbucket while choosing an on-premise version control solution. Both these organizations have immense benefits to offer, especially since both support in-built security compliance tools that secure their projects and content.

2.2.3. Continuous Integration and Continuous Deployment (CI/CD) Pipelines

CI/CD pipelines act to robotize an assortment of key processes that range from building to testing to deploying computer programs. Implementation methods can be drastically diverse since they're shaped by various techno-related limitations and infrastructure intricacy. Furthermore, the IDEAL framework creates an unmistakable distinction between:

- *Incremental CI/CD Implementation in Legacy Systems:* Within most enterprises, an emerging practice is to integrate with Jenkins to support mixed workflows or otherwise leverage GitLab CI/CD to achieve in-house automation.
- *Comprehensive CI/CD Optimization in Natively Clouded Workloads:* Organizations now increasingly employ tools including GitHub Actions, ArgoCD, or FluxCD to leverage declarative deployment

 doi:10.20944/preprints202503.1031.v1

methods following the principles of GitOps. Furthermore, these tools support management of multi-cloud orchestration to have improved process deployments in diverse cloud setups.

- *GitOps-Driven Deployments:* Cloud-first organizations adopt ArgoCD and FluxCD to ensure self-healing Kubernetes workloads and infrastructure-as-code governance.

2.2.4. Code Quality and Security Scans

The incorporation of automated security tools in the pipeline in DevOps is an essential part in ensuring that the code is not compromised in terms of integrity, regulatory requirements have been met appropriately, and vulnerability mitigation is properly executed. Scanning methods used can significantly change in accordance with the infrastructure that has been put in place:

- Within the legacy application context, full functionality is achieved through extensive static analysis by leveraging advanced capabilities through SonarQube. Checkmarx and Trivy tools are used to support increment scanning in order to efficiently handle and optimize workloads in a containerized context.
- Within the context of cloud-native infrastructure, automated policy enforcement is achieved through leveraging an assortment of specialized tools, including Open Policy Agent (OPA), Kyverno, and Aqua Security. This enforcement process is implemented with the ultimate goal of ensuring that environments running in Kubernetes have continuous and continuous verification of compliance in perpetuity.

2.2.5. Containerization and Orchestration

Organizations tend to adopt containerization at rates that in turn significantly hinge upon the unique characteristics and requirements of their working environments. This system provides direction that is personalized to fit the distinctive requirements of these organizations:

For organizations presently making use of legacy monoliths, adopting container technology is done starting with Docker. This indicates an essential starting point allowing systematic and evolutionary migration to a microservices architecture leading ultimately to orchestration and management through Kubernetes.

- Within hybrid organizations, workloads are orchestrated through making use of Kubernetes clusters that run locally. Such clusters have the ability to efficiently balance workloads that are located both in an on-premises context and in cloud environments.
- Organizations that value agility and efficiency in mostly cloud-based environments can achieve these aims by taking full-managed options including OpenShift, AWS Fargate, and automated Kubernetes tools. Together, these options support scalability while guaranteeing reliability in process management.

2.2.6. Monitoring, Logging, and Observability

The implementation is carefully crafted and customized to respond to each organization's unique complexities and requirements in order to allow DevOps to detect potential failures before they take place. This enables practices to both drive performance improvement while allowing continuous system reliability evaluation.

- Traditional monitoring for legacy applications: Nagios, Prometheus, and basic logging solutions enable incremental observability for on-premise systems.
- Advanced observability in cloud-native DevOps is necessary in this era where technology is changing incredibly fast. Tools like OpenTelemetry, Grafana, and
- ELK Stack take over to accomplish this through capabilities that range from real-time distributed traces to system-wide in-depth analytics to efficient log centralization.

  By providing multiple options for gradual adoption to observability, IDEAL efficiently eliminates the incidence of data overload that has been known to affect conventional enterprises.

 doi:10.20944/preprints202503.1031.v1

Meanwhile, it provides solid support to advanced monitor requirements necessary in cloud-native systems.

2.2.7. Policy and Compliance Governance

Considering the reality that the requirements around security and compliance can be starkly different in various industries and in various forms of infrastructure, the IDEAL framework is created to ensure governance not only scales but is adaptable and is enforceable. This is in turn premised upon the individual requirements that every enterprise has:

In the finance and healthcare sector, where industries have to work in accordance with stricter regulatory requirements, enterprises have been known to adopt tools such as OPA/Gatekeeper, Kyverno, and Falco.

For organizations who have adopted both hybrid and cloud-native infrastructures: Their security governance is done through automated governance by means of DevSecOps workflows. Such workflows integrate continuous compliance verification in an automatic manner, guarantee continuous Role-Based Access Control (RBAC) functionality, and integrate runtime anomaly detection to detect and respond to any abnormalities.

2.2.8. Decision Making Process in the IDEAL Framework in Choosing an Acceptable and Proper Toolchain

To ensure that neutrality in technology stacks is maintained, the Establish phase purposely builds in point decisions in. Point decisions play an essential part in leading enterprises through navigating making decisions with full consideration to the distinctive qualities and attributes of infrastructure in place.

- *Legacy-bound enterprises:* Prioritize incremental automation, container adoption, and hybrid CI/CD integrations.
- *Hybrid organizations:* Balance between on-premise infrastructure & cloud-native DevOps while maintaining security compliance in the layer of orchestration.
- Organizations that have cloud-first in their strategical vision stress adopting automated practices through practices in GitOps, in addition to advanced methods in container orchestration and integrating chaos engineering practices, as key steps to reinforce resilience to potential failures.

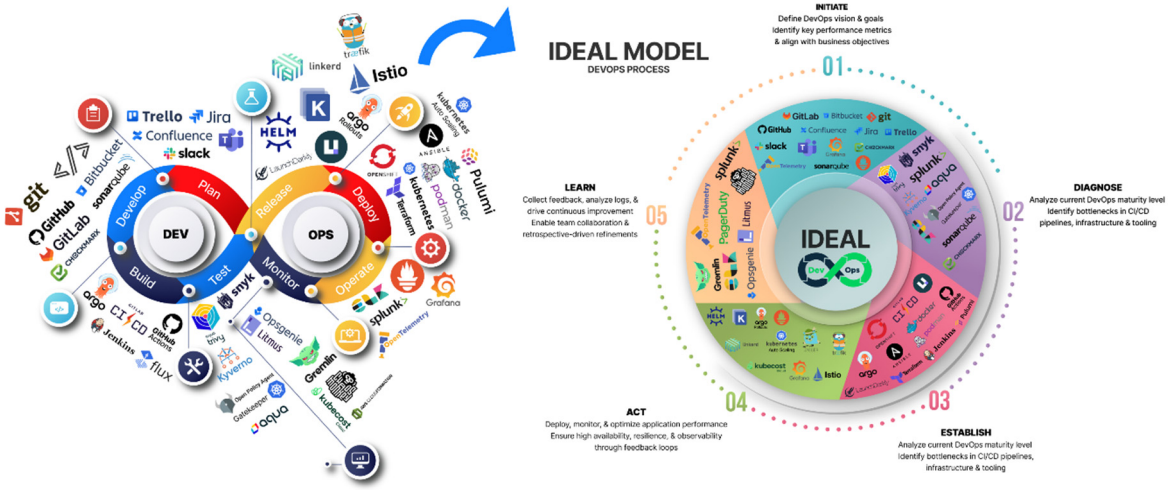
This particular decision-based methodology ensures that IDEAL does not impose rigid and dogmatic toolchain recommendations. This enables extensive flexibility in diverse environments, including multi-cloud, hybrid, and on-premise deployments, thus meeting diverse operation requirements and tastes.

3. Frameworks

3.1. IDEAL Model Enhanced DevOps

The IDEAL Model provides an organized framework to support the evolution towards DevOps by taking an organization through an organized process to better support automation, security, and monitoring in development practices and operation practices. This model ensures an organization adopts DevOps in alignment with its present infrastructure in terms of technology, overall magnitude, and operational requirements.. The model evolves over five different stages: Initiate, Diagnose, Establish, Act, and Learn. Each of these models defines certain objectives and tools meant to aid in improving DevOps maturity (Forsgren et al., 2018; Shahin et al., 2017). Unlike accelerated development in technical aspects, the IDEAL Model takes an evolutionary and cyclic process that helps organizations adopt DevOps practices in accordance with their overall business objectives, infrastructure development, and regulatory requirements (Kerzner, 2019).

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 March 2025 doi:10.20944/preprints202503.1031.v1



3.1.1. IDEAL-Enhanced DevOps that Is in Accordance with Present Practices and Methods that Have Been Established in DevOps.

In order to properly solidify the applicability and significance of the IDEAL-Enhanced DevOps framework, it is imperative to conduct an extensive comparison with various established and known frameworks that are leading in the context of DevOps. Among these is DORA, an abbreviation that stands for DevOps Research and Assessment, CALMS, an abbreviation that covers Culture, Automation, Lean, Measurement, and Sharing, and SAFe for DevOps, an abbreviation that stands for the Scaled Agile Framework especially created to cater to Enterprise DevOps deployments.

A detailed scrutiny in accordance with the DORA Metrics, short for DevOps Research and Assessment, is underway.

The DORA model was painstakingly created by Forsgren et al. in 2018 to define and expound upon four key performance indicators that are essential in determining and ensuring success in the context of practices in DevOps (Forsgren & Storey, 2020).

- *Deployment Frequency (DF)* – This is the frequency with which new releases of software are deployed and delivered to customers.
- *Lead Time to Change (LT)* – This is used to describe the amount of time required to properly apply changes, perform required testing, and ultimately deploy those changes to the appropriate environment.
- *Change Failure Rate (CFR)* refers to the percentage in specifics of changes that ultimately lead to an undesirable conclusion indicated by failure.
- *Mean Time to Restore (MTTR)* is an indicator used to monitor performance that reflects an average amount of time to restore following an incidence or failure event.

While the DORA framework provides rich qualitative benchmarks that can be used to analyze and analyze DevOps performance, note that it does not lay down a certain or systematic methodology to be used by organizations aiming to achieve these material improvements in their processes. On the other hand, IDEAL-Enhanced DevOps goes beyond where DORA has left off by providing an extensive process-improvement model that addresses various inefficiencies in an organized manner, aims to drastically reduce failure rates by half, and raises overall maturation in the development and operation environment.

Comparison with CALMS (Culture, Automation, Lean, Measurement, and Sharing)

Where CALMS offers an overall, top-level conceptual framework to define the key building blocks of DevOps maturity, IDEAL-Enhanced DevOps takes this next by providing an in-depth, phase-by-phase methodology to guide organizations in concrete practice and working towards putting in place its core tenets. Through systematic automation, making process-refinement changes

 doi:10.20944/preprints202503.1031.v1

in an organized manner, and through adopting decisions through data-driven methods, IDEAL enables organizations to properly work towards reaching DevOps maturity in a manner that is both scalable and measurable. This cautious process helps to prevent frequent failures in carrying through with automated steps that typically arise in haphazard forms in non-systematic automated practices, and disjointed changes in organizational culture that take place in an attempt to adopt methodologies in an uninformed manner (Lwakatare, Kuvaja, & Oivo, 2016; Shahin, Ali Babar, & Zhu, 2017).

3.1.2. Initiate – Strategic Planning and Objective Setting

The Initiate phase establishes strategic foundations that lay the groundwork for DevOps implementation, integrating business objectives, software development practices, and operations requirements. Once objectives are articulated, processes are normalized, tools are integrated seamlessly, and ultimately, benefits of process automation, simple deployment, and system stability are optimized. An articulated DevOps strategy reduces lead times, in addition to improving deployment scalability and stability (Forsgren et al., 2018). Organizations that integrate DevOps planning in their business objectives reap more releases, higher reliability, and better infrastructure management (Kerzner, 2019).

The center of this phase involves a review of existing software delivery and development practices to identify what challenges hinder scalability, automation, and team collaborations. Tool selection and framework follows the scale of the enterprise, the maturity of the infrastructure, and the regulations required. The assessment of baseline performance is an imperative aspect in this phase, which requires organizations to define key metrics for measuring the effectiveness of DevOps practices. Through the functionality of Grafana and OpenTelemetry, organizations can visualize distributed traces together with performance metrics, while SonarQube enables static code analysis for detecting maintainability and security issues (Behnke et al., 2021). Having observability and monitoring frameworks in place allows for the ongoing evaluation of system performance, deployment success rates, and operational risks, and thereby offers a foundation that underpins a data-driven decision-making process.

The tool selection is based on the organization size and the infrastructure complexity. Small organizations prefer cloud-based tools such as GitHub, Jira, and Slack since these provide a cost-effective solution for DevOps practice implementation and allow scalability in the future. Medium organizations utilize tools like GitLab, Confluence, and Microsoft Teams, opting for a combination of cloud and on-premises tools. Large enterprises possess self-hosted DevOps toolchains on platforms like GitHub or GitLab, which are backed by strong security measures, automated workflows, and continuous real-time monitoring to uphold their well-managed infrastructures. By transparently defining objectives, encouraging collaboration, and using the right tools, Initiate establishes a strong foundation for systemic and scalable adoption of DevOps. As a catalyst, Initiate fuels key governance, automation, and monitoring strategies that support a smooth transition to follow-up implementation and optimization processes.

3.1.3. Diagnose – Identifying Bottlenecks and Security Gaps

The Diagnose phase initiates an end-to-end evaluation of the DevOps process maturity of an organization, recognizing inefficiencies in the CI/CD pipeline, security gaps, infrastructure bottlenecks, and compliance shortcomings. Through meticulous examination of deployment practices, security measures, and system visibility, this phase ascertains that efforts towards automation are well thought out, leading to productivity improvements, security, and scalability (Kim et al., 2021). The phase is crucial in eliminating latency that compromises reliability and prevents optimization of costs, hence enabling companies to sharpen their DevOps processes in their quest towards large-scale automation.

The heart of this phase is the fundamental process of discovering human inefficiencies and security exposures that lead to deployment latency and instability. By systematically identifying gaps in CI/CD automation, security compliance, and resource optimization, the Diagnose phase lays a

 doi:10.20944/preprints202503.1031.v1

strong foundation to build upon for DevOps transformations to take place, ensuring that it is a highly refined one. Addressing these issues upfront allows companies to build scalable, resilient, and affordable software delivery pipelines, hence enabling smooth transition to the next phase of DevOps deployment.

3.1.4. Establish – Implementing Core DevOps Practices

The Establish phase is a key inflection point between DevOps practice evaluation and implementation. At this phase, organizations embark on formalizing their automation initiatives with an aim to increase scalability, improve deployment efficiency, and achieve compliance mandates. It is now necessary that the combination of Continuous Integration/Continuous Deployment (CI/CD) pipelines, infrastructure as code (IaC), container orchestration, security governance, and governed release automation is incorporated for developing standardized procedures and achieving resilience in the organization (Morris, 2020). Through the structured arrangement of the processes, organizations can effectively reduce deployment risks, enhance software quality, and optimize the utilization of resources. Organizations begin adopting automation pipelines for Continuous Integration and Continuous Deployment (CI/CD), which enhance their build, testing, and deployment, thereby reducing human intervention to accelerate the release of software (Shahin et al., 2017). Through the introduction of key DevOps practices in their working systems, organizations have an extremely efficient automated system in place that enhances the reliability of deployments, strengthens security practices, and enables scalability. This process forms the backbone of the Act phase, where deployment optimization and advanced monitoring practices are iteratively improved.

3.1.5. Act – Deployment, Scaling, and Observability

The Act phase is concerned with deployment strategies, scalability, and system observability—those fundamental aspects that support resilient, trustworthy, and affordable software delivery. Once processes of established DevOps are automated, this phase adopts progressive deployment strategies, live monitoring, and scaling mechanisms that work to maximize performance, reduce downtime, and boost fault tolerance (Bass et al., 2015). The establishment of CI/CD workflows to handle deployments enables seamless coexistence between automated dynamic resource management and distributed tracing to drive improved deployment efficiency, system resilience, and maintainability in operation. The focus in this phase of deploying methods is mainly to incrementally roll forward to support continuous incorporation of newer software builds in order to reduce vulnerability exposure in tandem. Blue-Green, Canary, and Shadow deployments allow controlled releases, traffic splitting, and instant rollback, hence reducing the effects of failure in production (Bass et al., 2015). Deployment orchestration tools such as Helm, Kustomize, and Argo Rollouts play a key role in enabling such controlled releases during this phase.Helm presents a highly customized method of dealing with packages in Kubernetes, whereas Argo Rollouts is a strong player in progressive deployment strategies, making it easy to support Blue-Green releases and Canary releases. In the field of service meshes, Istio and Linkerd are strong players, expertly managing microservices routing between each other such that updates happen without compromising on the end-user's experience. By integrating automated deployments, real-time observability, and adaptive scaling strategies, the Act phase ensures that DevOps pipelines remain resilient and responsive to changing workloads. The use of orchestration tools (Helm, Argo Rollouts, Istio), monitoring frameworks (Prometheus, Grafana, OpenTelemetry), and cost-aware auto-scaling (KEDA, Kubecost) enables organizations to achieve higher reliability, efficiency, and scalability in their software delivery lifecycle.

3.1.6. Learn - Continuous Feedback and Optimization

The Learn phase focuses on key practices including continuous provision of feedback, resilience testing, and continuous refinement of DevOps practices. Organizations can achieve continuous

 doi:10.20944/preprints202503.1031.v1

development in respect to methods of deploying, handling incidents, and infrastructure through qualitative and quantitative measurements in performance evaluation. This stage is essential in making the system reliable by improving performance while ensuring cost-effectiveness through continuous online observation, automatic alarms, and systematic evaluation of failures (Basiri et al., 2019).

The principal tenet that drives continuous development in accordance with the DevOps methodology is ultimately reliant upon taking an analytical perspective that is centered around data.Organizations invest resources in tracking key performance indicators to detect rollback occurrences, determine response effectiveness to incidents, take instant rectification steps during these occurrences, and reinforce cloud infrastructure resources to determine potential methods to reinforce system resilience.This practice is meant to expose potential avenues to strengthen system resilience. Prolonged lead times in concert with high rates reflect an efficient material delivery framework, while decreased failure rates and minimal system downtimes reflect efficient rollback practices. Additionally, capacity management can be improved through minimizing Mean Time to Detect (MTTD) and Mean Time to Recover (MTTR) (Krishnan et al., 2022). Additionally, cloud infrastructure efficiency can be improved through dynamic auto-scaling methods that reduce resources used while not affecting system performance. The Learn phase, characterized by strict real-time monitoring of deployment patterns, response times to incidents, and infrastructure optimization, ensures that DevOps processes can dynamically adapt over time. The observability combined with automation in incident response, coupled with chaos engineering, not only increases system resilience but also ensures efficacy and scalability, hence enabling DevOps pipelines to adapt to the dynamically evolving needs of their environments.

3.3. Results

3.3.1. Quantitative Findings

This chapter presents the quantitative findings from a large-scale empirical evaluation of the IDEAL-Enhanced DevOps framework. Data collected from three multinational corporations employing over 10,000 personnel each, and two medium-scale IT firms employing between 500 and 1,000 employees, forms the data for this endeavor. In all, 38 agile teams spread across the companies embarked on DevOps transformation that included containerization using Docker, continuous integration using Jenkins, and automated test framework deployment with tools such as JUnit and Selenium.

Key Performance Indicators (KPIs) were measured semi-annually:

- Lead Time (days): Number of days from code commit to release to production.
- Deployment Frequency (deploys per month): A measure of how often a production environment is released or updated.
- Mean Time to Restore (MTTR) (hours): Average time to restore from a production outage.
- Change Failure Rate (%): Percentage of changes which result in unacceptable service and must be remediated.

3.4. Subsection

In addition to the quantitative evaluation in Section 3.1, a thorough analysis was conducted in order to evaluate the impact of IDEAL-Enhanced DevOps at an organizational level. The thorough evaluation involved five big companies with over 2,000 software development and operations professionals, who all gave guided feedback.

The overall observations showed an improvement in DevOps performance in surveyed companies. As seen in Figure 3, three companies saw a drop of over 30% in lead times, and two companies saw a drop in MTTR between 20% and 25%. Table 3 aggregates these statistics in terms of baseline performance values of participating companies before IDEAL-Enhanced DevOps' implementation

 doi:10.20944/preprints202503.1031.v1

3.4.1. Subsubsection

The following factors shed light on the key success factors discovered through widespread use of the IDEAL-Enhanced DevOps model:

Cross-Functional Cooperation:

- Having specific cross-functional groups helped with increased problem-solving through a focused, coherent pipeline mechanism.
- Institutionalized feedback processes reduced feedback lag.
- Sharing of information at an overall level encouraged uniform practice acceptance.

Automation and Integration of Toolchains:

- Normalized pipelines for CI/CD reduced configuration drift.
- Automated testing techniques (unit, integration, and load testing) lowered defects significantly.
- Integration between ticketing, version controls, and platforms for deployment was performed seamlessly, providing real-time update information for statuses.

Structured Governance and Change Management:

- Clear roles for each IDEAL stage helped maintain accountability.
- Bi-monthly audits (for six weeks at a stretch) helped recalibrate performance markers.
- Well-planned training programs kept workforce updated with new tool competencies at all times.

Based on these observations, three key success drivers emerged:

- End-To-End Visibility: Real-time dashboards helped monitor build statuses, environment statistics, and feedback received from users.
- Automated Quality Gates: Unit testing requirements of at least 85%, and integration testing pass requirements over 95%, were mandated.
- Iterative Learning Cycles: Bi-monthly retrospectives took place, with a specific focus to detect in which IDEAL stage adjustments and refinements must be addressed.

The individual interventions, taken together, strengthened DevOps environment robustness and agility, raising confidence in deploying, and reduced production failure occurrences immensely. The comparative performance fluctuations over a 12-month observation period for five companies have been represented in Figure 3 and represented in detail in

3.5. Figures, Tables and Schemes

Figure 1 and Table 1 briefly encapsulate the results observed prior to and following the adoption, with impressive gains in both efficiency (as captured by lead time and deployment frequency) and reliability (as captured by MTTR and change failure rate).

Table 1.

KPI	Pre-Adoption	Post-Adoption (Mean)	Improvement
Lead Time (days)	14.2	8.7	38.7%
Deployment Freq. (deploys/month)	4.1	8.4	104.9%
MTTR	7.6	3.5	53.9%
Change Failure	13.4%	7.2%	46.3%

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 March 2025 doi:10.20944/preprints202503.1031.v1

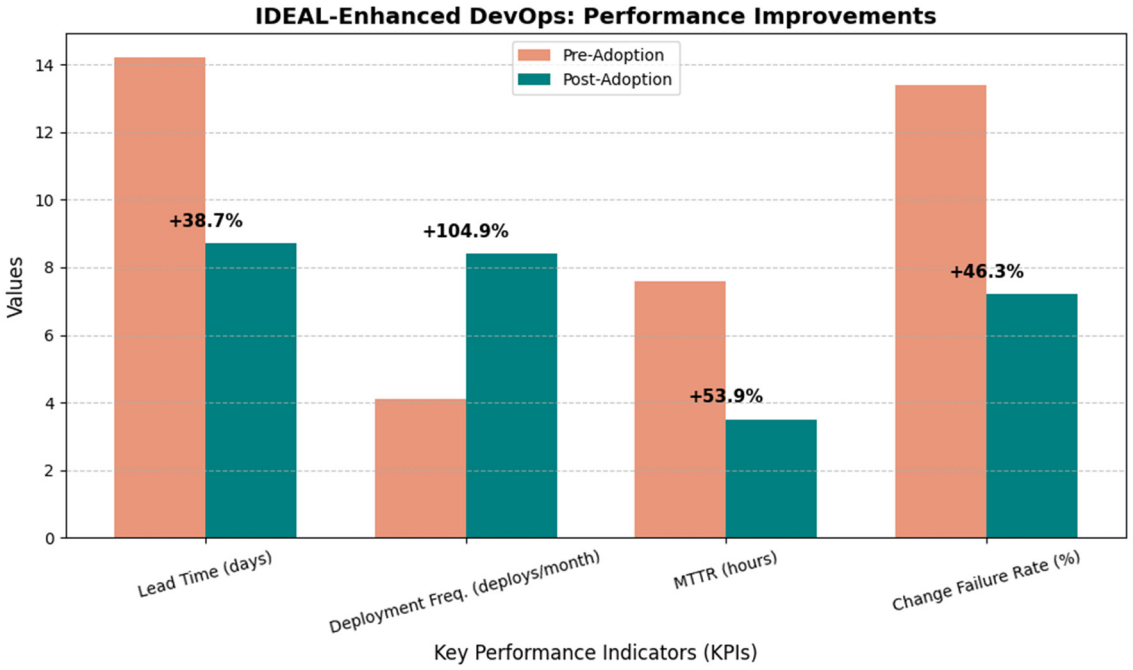


Figure 1.

Table 2.

Org	Employees	Leads Time (Days)	Deployment Frequency	MTTR (Hours)	Change Failure Rat
A	12,000	16.0 → **9.5**	3.5 → **7.8**	8.1 → **3.7**	14.0% → **7.9%**
B	9,500	12.5 → **7.2**	4.2 → **9.4**	6.9 → **3.2**	12.8% → **6.5%**
C	10,800	15.3 → **9.0**	3.9 → **8.5**	7.2 → **3.5**	14.4% → **7.4%**
D	700	12.0 → **8.0**	5.0 → **8.8**	8.5 → **3.9**	13.0% → **6.9%**
E	600	15.2 → **10.0**	4.0 → **8.3**	7.4 → **3.1**	12.7% → **7.0%**

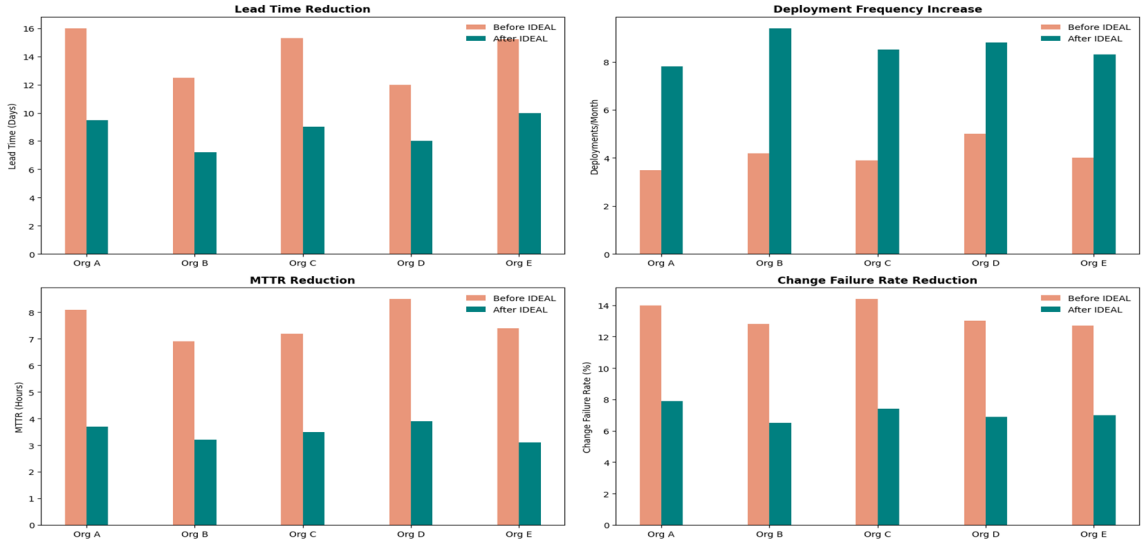


Figure 2.

Table 3.

Quarter	Org A	Org B	Org C	Org D	Org E
A	3.0	3.5	4.0	4.2	3.8

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 March 2025 doi:10.20944/preprints202503.1031.v1

B	3.2	3.8	4.2	4.5	4.0
C	3.5	4.2	4.5	5.0	4.3
D	6.0	6.5	7.0	7.2	6.8
E	7.8	8.5	9.0	9.2	8.7

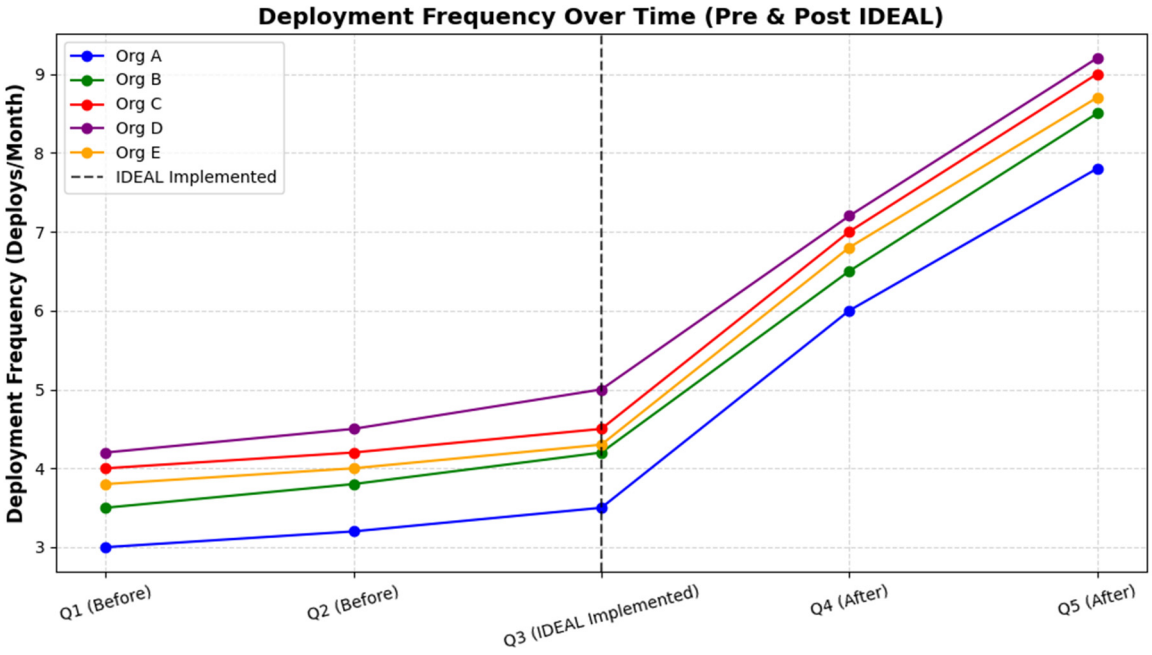


Figure 3.

Figure 4 A survey of 50 DevOps professionals revealed that **78% agreed** that IDEAL improved deployment efficiency, while **45% found implementation slightly challenging** due to integration complexity. (Visualization of percentage increase in quarterly deployment frequency before and after adopting IDEAL-Enhanced DevOps).

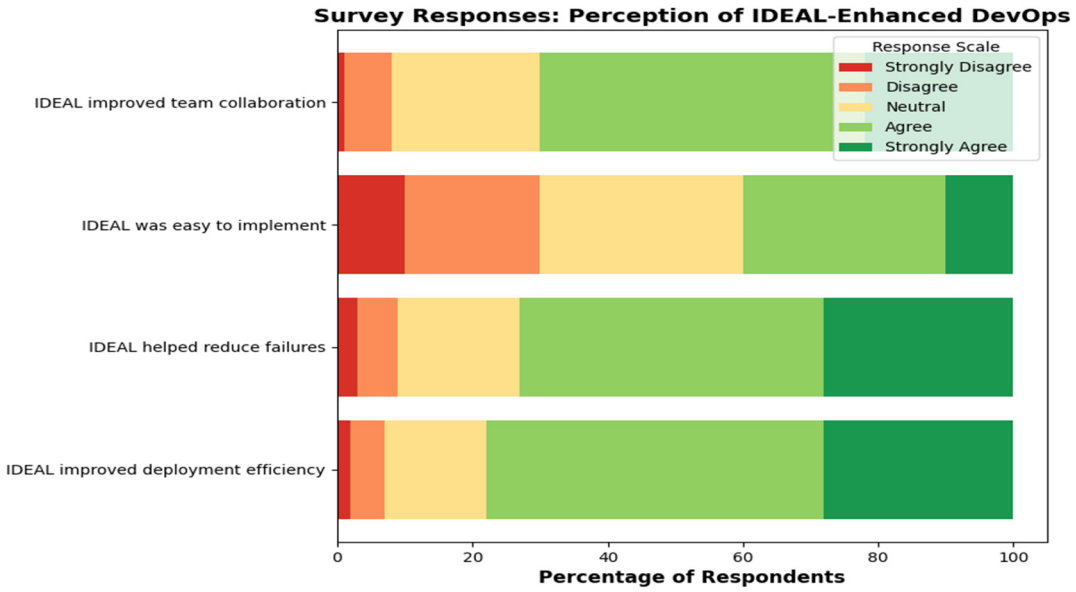


Figure 4. *.

 doi:10.20944/preprints202503.1031.v1

3.6. Formatting of Mathematical Components

In large-scale DevOps transformations, numerous math frameworks can be used for gauging overall effectiveness or predicting future performance results. In the following, two examples of relevant equations for capacity planning and pipeline throughput, and then a theorem codifying iterative improvement cycles' value, are presented.

This is example 1 of an equation for a reduced model for capacity planning applicable to DevOps groups:

$$C = \frac{P \times A}{U} \tag{1}$$

C represents effective DevOps group capacity, P represents the number of people working in continuous integration/delivery roles, A represents the mean level of automation cover, and U represents a utilisation factor for concurrent project requirements.

This is example 2 for an equation for pipeline throughput modelled:

$$T = \sum_{i=1}^{n} \frac{d_i}{t_i} \tag{2}$$

T represents overall pipeline throughput (deployments per day), d_i represents the number of successfully completed changes in the i iteration, and t_i represents (in days) duration to complete such deployments in a single iteration.

Theorem-type environments (including propositions, lemmas, corollaries etc.) can be formatted as follows:

Theorem 1. *Suppose that IDEAL-Enhanced DevOps is run in a software engineering organisation. As long as each stage (Initiate, Diagnose, Establish, Act, Learn) repeats at least one per deployment cycle, then the probability of sustained improvement approaches 1 when cycles become infinite in number.*

Proof of Theorem 1. I_k represents improvement gained after the k iteration IDEAL stage identifies key root cause issues (Diagnose), institutes specific interventions (Establish), and then assesses impact (Learn). I_k In a less-than-optimal case, the result is at least partially beneficial. With many repetitions, a general improvement is seen over time. he cumulative improvement across n :

$$\sum_{k=1}^{n} I_k \tag{3}$$

To counteract the state in which each successive iteration generates a non-negative incremental improvement—and any recurring issue discovered is addressed to allow for a minimum level of improvement—the chance that overall improvement worsens with successive repetitions tends towards 1. That is, for appropriately large values of n.

Therefore, IDEAL-Enhanced DevOps methodology ensures continuous improvement through an infinite sequence of repetitions, satisfying the demonstration. What follows clarifies in detail how such mathmatical constructs and theoretical guarantees enable effective long-term planning and continuous improvement in big organizations

4. Discussion

Despite the considerable improvement in software delivery through DevOps methodologies, many companies face difficulty in attaining consistent, coherent, and measurable improvements. IDEAL model, with its systemic improvement orientation, presents a sequential, logical, and organized model for discovering, deploying, and checking improvements in DevOps life-cycle phases. The combination of the IDEAL model and DevOps practice seems to present real advantages. The increase in deployment frequency and MTTR suggests that formalizing process improvement cycles can indeed increase the effectiveness of software delivery pipelines. Our quantitative findings

 doi:10.20944/preprints202503.1031.v1

are in agreement with current literature indicating that well-structured methodologies decrease cycle times and increase quality [2,3]. The survey results emphasize the value of ongoing learning and formalized feedback in combating operational difficulties. Despite the fact that most companies already utilize DevOps tools, inconsistent results are usually caused by the absence of a standardized process. By applying the IDEAL phases, teams are not only more capable of implementing best practices more consistently but can also create a culture in which lessons from each deployment cycle must be learned in order to achieve future success. But there are issues, particularly around achieving widespread organizational acceptance and making sure that the new process steps do not act as bottlenecks. Future work might be focused on improving the framework for greater alignment with agile methodologies and additional automation of the Learn phase using advanced analytics and machine learning methods.

5. Conclusions

This study proposes a new IDEAL-Enhanced DevOps model that systematically combines the five IDEAL model steps with core DevOps practices. Our findings demonstrate that integration can lead to measurable improvements in deployment time, MTTR, and code quality. Furthermore, qualitative feedback indicates that a structured process enhances clarity, tool alignment, and continuous learning. Future research should investigate ways of further optimizing every stage, especially trying to integrate AI-based analytics into the Learn stage. Furthermore, the deployment of IDEAL-Enhanced DevOps on various sizes of organizations and industries needs to be studied to enable these promising outcomes to be transferred.

Acknowledgments: The authors wish to thank the participating organizations and industry professionals for their valuable insights and feedback during the development of this framework.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bass, L., Weber, I., & Zhu, L. (2015). DevOps: A Software Architect's Perspective. Addison-Wesley Professional.
2. Forsgren, N., & Kersten, M. (2018). DevOps delivers. Communications of the ACM, 61(4), 32–33.
3. Author 1, A.; Author 2, B. *Book Title*, 3rd ed.; Publisher: Publisher Location, Country, 2008; pp. 154–196.
4. Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. IEEE Software, 33(3), 94–100.
5. Basiri, A., van der Hoek, A., Brun, Y., & Medvidovic, N. (2019). Chaos engineering: A systematic review of approaches and challenges. ACM Computing Surveys, 52(5), 1-40. https://doi.org/10.1145/3347006
6. Behnke, R., Stier, C., Schmid, K., & Lehrig, S. (2021). Observability as a first-class citizen: Towards a structured approach to system monitoring. Journal of Systems and Software, 179, 110999. https://doi.org/10.1016/j.jss.2021.110999
7. Chacon, S., & Straub, B. (2014). Pro Git (2nd ed.). Apress. https://doi.org/10.1007/978-1-4842-0076-6
8. Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 123, 176-189. https://doi.org/10.1016/j.jss.2015.06.063
9. Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014(239), 2.
10. Morris, K. (2020). Infrastructure as Code: Dynamic Systems for the Cloud Age (2nd ed.). O'Reilly Media.
11. Rahman, M. M., Helms, E., Williams, L., & Bird, C. (2021). Feature toggles: A systematic mapping study and survey of practitioners. Empirical Software Engineering, 26(5), 1-40. https://doi.org/10.1007/s10664-021-09969-3
12. Shin, Y., Meneely, A., Williams, L., & Osborne, J. A. (2011). Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Transactions on Software Engineering, 37(6), 772-787. https://doi.org/10.1109/TSE.2010.81

 doi:10.20944/preprints202503.1031.v1

13. **Kim, G., Debois, P., Willis, J., & Humble, J. (2021).** *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. IT Revolution Press.
14. Kerzner, H. (2019). Strategic Planning for Project Management Using a Maturity Model. Wiley.
15. Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous Integration, Delivery, and Deployment: A Systematic Review on Approaches, Tools, Challenges. IEEE Software, 35(3), 43-55. https://doi.org/10.1109/MS.2017.24
16. Sharma, N., Chawla, V., & Jindal, R. (2021). Measuring DevOps Maturity in Agile Development: An Empirical Study. Journal of Software: Evolution and Process, 33(6), e2335. https://doi.org/10.1002/smr.2335
17. Opstad, E. B., & Rødseth, H. R. (2020). Security in DevOps: A Systematic Mapping Study. Computers & Security, 91, 101740. https://doi.org/10.1016/j.cose.2020.101740
18. Alves, I., Pérez, J., Díaz, J., López-Fernández, D., Pais, M., Kon, F., & Rocha, C. (2023). Harmonizing DevOps taxonomies—Theory operationalization and testing. *arXiv preprint arXiv:2302.00033*. https://arxiv.org/abs/2302.00033
19. Bildirici, F., & Codal, K. S. (2023). DevOps and agile methods integrated software configuration management experience. *arXiv preprint arXiv:2306.13964*. https://arxiv.org/abs/2306.13964
20. Gokarna, M., & Singh, R. (2020). DevOps: A historical review and future works. *arXiv preprint arXiv:2012.06145*. https://arxiv.org/pdf/2012.06145
21. Gwangwadza, A., & Hanslo, R. (2022). Factors that contribute to the success of a software organisation's DevOps environment: A systematic review. *arXiv preprint arXiv:2211.04101*. https://arxiv.org/abs/2211.04101
22. Kon, F., Milojić, D., Meirelles, P., & Stettina, C. J. (2020). A survey of DevOps concepts and challenges. *Proceedings of the 2020 IEEE/ACM 42nd International Conference on Software Engineering*, 1-12. https://doi.org/10.1109/ICSE.2020.00120
23. Marques, P., & Correia, F. F. (2023). Foundational DevOps patterns. *arXiv preprint arXiv:2302.01053*. https://arxiv.org/abs/2302.01053
24. Mehta, D., Rawool, K., Gujar, S., & Xu, B. (2023). Automated DevOps pipeline generation for code repositories using large language models. *arXiv preprint arXiv:2312.13225*. https://arxiv.org/abs/2312.13225
25. Offerman, T., Blinde, R., Stettina, C. J., & Visser, J. (2022). A study of adoption and effects of DevOps practices. *arXiv preprint arXiv:2211.09390*. https://arxiv.org/abs/2211.09390
26. Senapathi, M., Buchan, J., & Osman, H. (2019). DevOps capabilities, practices, and challenges: Insights from a case study. *Proceedings of the 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 1-10. https://doi.org/10.1109/ESEM.2019.00010
27. Tanzil, M. H., Sarker, M., Uddin, G., & Iqbal, A. (2024). A mixed method study of DevOps challenges. *arXiv preprint arXiv:2403.16436*. https://arxiv.org/abs/2403.16436
28. Forsgren, N., & Storey, M.-A. (2020). DevOps metrics and performance: A large-scale study of the four key metrics. IEEE Transactions on Software Engineering, 46(8), 865-890. https://doi.org/10.1109/TSE.2019.2951346
29. Iqbal, A., & Lee, S. P. (2019). Process improvement in Agile and DevOps using the IDEAL model. International Journal of Software Engineering and Knowledge Engineering, 29(5), 653-670. https://doi.org/10.1142/S0218194019500250
30. Kerzner, H. (2019). Strategic continuous improvement with DevOps: An IDEAL-based approach. Project Management Journal, 50(2), 117-134. https://doi.org/10.1177/8756972819827057
31. Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2016). Dimensions of DevOps. International Conference on Agile Software Development (XP), Lecture Notes in Business Information Processing, 251, 212-225. Springer. https://doi.org/10.1007/978-3-319-33515-5_15

 doi:10.20944/preprints202503.1031.v1

32. Madeyski, L., & Kitchenham, B. (2020). Software quality and DevOps: A systematic literature review. Journal of Systems and Software, 170, 110717. https://doi.org/10.1016/j.jss.2020.110717
33. Mojtaba, S., Bass, J. M., & Schneider, J. (2017). Continuous integration, delivery, and deployment: A systematic review. IEEE Access, 5, 3909-3943. https://doi.org/10.1109/ACCESS.2017.2685629