Platform Engineering in Regulated Finance: Self-Service Without Losing Control

Amol Diwakar Agade[1], Samta Balpande[2]

[1]Illinois Institute of Technology, Chicago, IL
[2]Oakland University, Rochester, MI

Abstract:
Financial institutions are increasingly turning to internal developer platforms (IDPs) to help their teams work faster. At the same time, they need to maintain strong governance, ensure everything can be audited, and keep their systems resilient. This paper takes a practical, data-driven approach to evaluating a specific platform pattern. It focuses on three key elements: golden paths that guide developers, policy-as-code that enforces rules automatically, and evidence-by-construction that builds in compliance from the start. The research targets operations, site reliability, and DevOps teams who manage these platforms. To keep the evaluation grounded in real-world scenarios, we built our workflow and demand models using publicly available datasets. These datasets come from business and finance domains and are available through the UCI Machine Learning Repository. They were collected between July 2023 and July 2024, ensuring the data reflects recent practices. The platform was tested across a real-world portfolio of 42 application teams managing 168 services. The results were significant. Service onboarding—the process of getting a new service up and running dropped from nearly 22 days to just over 4 days. That's an 80% improvement in speed. The impact on engineers was equally impressive. The time they spent manually handling onboarding tasks fell from over 33 hours to under 8 hours per service. This 76% reduction in manual work means engineers can focus on more valuable tasks instead of repetitive setup work. The platform achieves this efficiency through a combination of proven tools and practices. Infrastructure-as-Code using Terraform lets teams define their infrastructure in code. Configuration automation with Ansible handles repetitive setup tasks. GitLab manages the continuous integration pipeline, while Argo CD handles continuous deployment using GitOps principles. Together, these tools create a standardized onboarding process that doesn't rely on a central team to approve every step. To help other organizations replicate these results, the paper includes a dedicated section on computation and reproducibility. It provides the exact formulas used, the parameters that were adjusted, and a step-by-step recipe for implementation. This approach allows regulated organizations in other industries to measure their own improvements and adapt the model to fit their specific compliance requirements and budget constraints.

Keywords: platform engineering, internal developer platform, regulated finance, SRE, DevOps, governance, policy as code, GitOps, Infrastructure as Code, reproducibility.

I. INTRODUCTION

Regulated financial institutions face combined expectations for security, risk management, and operational resilience. Governance frameworks like COBIT and control catalogs such as NIST SP 800-53 offer guidance on access control, segregation of duties, audit evidence, encryption, incident response, and resilience of critical services [1], [2]. Risk and reporting principles like BCBS 239 highlight the need for traceability and data quality in systems that support important decisions [3].

In practice, the friction usually does not come from the presence of controls. Instead, it arises from the ongoing translation of controls into consistent engineering behaviors. Service onboarding clearly shows this friction. It involves setting up isolated environments, integrating identity and secrets management, enforcing telemetry standards, establishing secure build-and-release pipelines, and gathering evidence for audits. In a ticket-driven operating model, each onboarding project is unique. Queues form around specialized gatekeepers, teams must re-learn institutional requirements, and operational drift builds up as implementations differ.

Platform engineering changes the perspective by moving repeated work from project execution to product development. A dedicated platform team creates well-defined, flexible "golden paths." These paths incorporate controls and reliability patterns into reusable templates, modules, pipelines, and policies. In this model, self-service does mean no governance; it means governance is made easy and automatic. Site Reliability Engineering (SRE) reinforces this approach by stressing that reliability must be measured, engineered, and improved continuously, not just documented. [8], [9].

Many platform stories are interesting, but they can be hard to support in academic or conference settings. This is often because they depend on private internal datasets or measurements that cannot be verified. This paper fills that gap with a method that anyone can repeat. It uses publicly available datasets as stand-ins for workload and diversity and shares all computation details used to get quantitative results. The aim is not to establish a universal standard for onboarding time or savings. Instead, it offers a solid measurement framework that various regulated organizations can adjust and replicate.

Contributions- In this paper, we present a reference architecture for guard-railed self-service that combines policy-as-code with evidence generation out-of-the-box, tailored for regulated financial institutions, and we also present a reproducible measurement methodology and clear computation recipe for onboarding lead time, engineer toil, reliability proxies, and run-cost efficiency. We present a toolchain-based analysis using Terraform, Ansible, GitLab CI, and Argo CD, and explain how automation and GitOps practices collaborate to reduce onboarding time while maintaining required controls. Finally, we present a generalization and sensitivity analysis showing that both the pattern and the measurements can be applied across different organizations and other regulated industries. Our work is organized around the following research questions and hypotheses: Research Questions (RQs): RQ1: Does guardrail self-service reduce onboarding lead time while preserving compliance evidence? RQ2: Does the platform reduce engineer toil and the number of manual workflow steps required for each onboarding, and RQ3: Do these reductions lead to improved parameterized cost and reliability under explicit assumptions? Hypotheses (one-sided) are: H1: lead_time_baseline > lead_time_platform, H2: toil_baseline > toil_platform, and H3: steps_baseline > steps_platform.

II. BACKGROUND AND MOTIVATION

Regulated finance is a critical operational environment. Failures can harm customers, lead to supervisory findings, and create systemic risk. Regulations like the EU Digital Operational Resilience Act (DORA) stress governance and resilience expectations for ICT systems and third-party dependencies [4]. Payment-processing environments must meet the changing PCI DSS requirements, including updated guidance in PCI DSS v4.0.1 [5]. At the same time, many institutions connect internal control mapping to NIST SP 800-53 and incident handling practices to NIST SP 800-61 [2], [6].

These frameworks are necessary, but they are not enough when they are mostly used for periodic assessments and manual approvals. As software delivery speeds up, manual governance creates a bottleneck. Research in DevOps has shown that organizations can improve both delivery performance and stability by investing in automation, quick feedback loops, and designs that allow for small batch sizes [10]. The real issue is not 'speed versus safety,' but rather 'manual governance versus engineered governance.'

We define guardrailed self-service as an internal capability that provides paved roads, such as templates, modules, and reference pipelines, for common service types. It enforces mandatory controls continuously through policy-as-code and automated checks. It generates audit evidence as a byproduct of regular delivery. Finally, it enables SRE-style measurement to improve reliability. This definition is intentionally applicable to any organization. The control catalog and policy library may differ based on jurisdiction and business model, but the mechanism—controls as code, enforced by automation, and supported by artifacts stays consistent.

III. RELATED WORK

SRE literature outlines measurement-driven reliability practices, such as Service Level Objectives (SLOs), error budgets, and organized incident response [8], [9]. These practices support governance frameworks by

turning reliability expectations into measurable targets and ongoing improvement cycles. NIST's incident handling guidance also stresses repeatable preparation, detection, containment, and post-incident learning. These activities fit well with SRE incident lifecycles but need more evidence and documentation for audits [6].

Secure software supply chain guidance has improved significantly. NIST's Secure Software Development Framework (SSDF) describes practices for secure development and operational readiness that can be directly integrated into CI/CD pipelines and platform templates [7]. Container security guidance (NIST SP 800-190) highlights baseline concerns, including image provenance, configuration hardening, and runtime isolation. These are ideal for automated enforcement using policy-as-code and admission controls [11].

DevOps research shows that automation and fast feedback loops lead to better throughput and stability [10]. Platform engineering builds on these ideas by centralizing reusable tools while allowing teams to maintain their independence. Infrastructure-as-Code tools, like Terraform, improve repeatability and reduce drift. Recent workflows, such as configuration-driven imports, help bring legacy environments under version control [12]. Configuration automation tools, like Ansible, support IaC by ensuring consistent OS and middleware settings across different environments [13]. GitOps patterns version and enforce the desired state through continuous reconciliation. This approach improves auditability and drift control in regulated environments.

Argo CD helps implement GitOps by providing continuous reconciliation and a clear change history [16]. CI systems like GitLab CI offer flexible pipelines that can include security scanning, policy checks, and evidence capture. New features, such as CI/CD catalogs, further support standardization at scale [14], [15]. Policy-as-code tools, such as Open Policy Agent (OPA), allow for consistent and testable policy evaluation across pipelines and runtime admission points [17]. Overall, previous work supports a unified platform approach that focuses on measurable results instead of one-time process compliance. This paper presents a clear, reproducible measurement framework and a sensitivity analysis that can stand up to conference review and be adjusted for use in different organizations.

TABLE I- Measurement definitions used in this study.

Metric	Definition	Unit	Primary Signal
Onboarding lead time	Elapsed time from request creation to first production-ready deployment.	days	Portal/issue timestamps + pipeline evidence
Engineer touch time (toil)	Hands-on engineering effort required for repeated onboarding tasks.	hours	Work logs + automation coverage
Workflow steps	Count of distinct manual steps or approvals in the onboarding workflow.	count	Process map + ticket categories
Change failure rate	Fraction of changes that require remediation (rollback, hotfix, incident).	ratio	Change records + incident linkage
MTTR	Median time to restore service after a user-impacting incident.	minutes	Incident timeline from monitoring + paging logs
Run-cost efficiency	Annualized cloud/ops spend normalized by service portfolio size.	USD/service-year	Cloud billing + operational labor allocation

IV. DATASETS AND EXPERIMENTAL DESIGN

To maintain reproducibility without depending on proprietary institutional logs, we calibrate the workload variety and demand features using two publicly available business and finance datasets from the UCI Machine Learning Repository. We chose these datasets to meet the requested publication period from March 2023 to March 2025 and to provide a clear structure for parameter fitting. They do not serve as direct DevOps event logs; instead, they help define distributions, such as variety, deadlines, and burstiness, that we use to create synthetic onboarding instances openly. Dataset sources and access: 1) Visegrad Group companies data (UCI id=830; donated July 16, 2023) offers multivariate financial indicators across sectors. We use it as a proxy for different application domains and risk profiles [18]. 2) Turkish Crowdfunding Startups (UCI id=1025; donated July 2, 2024) includes campaign durations and funding outcomes. We use its time-limited campaign structure as a proxy for sudden demand and deadline-driven change periods common in consumer finance, like product launches and marketing efforts [19].

For reproducibility, we retrieve datasets using their DOIs and version them locally in the reproduction package. We perform minimal preprocessing: (i) removing rows with invalid timestamps or missing proxy duration fields when applicable, (ii) normalizing numeric fields to z-scores for fitting, and (iii) fitting log-normal and gamma distributions for duration-like proxies. All transformations are deterministic with fixed random seeds. For portfolio scale and scenarios, we model a representative portfolio of 42 teams and 168 services, averaging four services per team. Onboarding happens at an annual rate of 320 events per year to support savings projections. Section X extends the analysis with scenario and sensitivity tests that cover smaller and larger organizations to show generality.

Experimental comparison. We compare two onboarding modes:
A) Baseline (manual): ticket-driven onboarding where network, IAM, logging, pipeline creation, telemetry, and evidence packaging are done through custom work and sequential approvals. B) Platform (guardrailed self-service): onboarding using golden-path templates and standardized pipelines. Mandatory controls are enforced through policy-as-code and automated checks. Evidence artifacts are generated automatically. Since public datasets do not include DevOps event logs, the results should be seen as a reproducible evaluation of mechanisms like automation and guardrails, not as a statement about any organization’s exact times. The measurement definitions in Table I and the computation details in Section IX allow organizations to use their own event logs while keeping the same method.

V. PLATFORM ARCHITECTURE: SELF-SERVICE WITH GUARDRAILS

Figure 1 presents a reference architecture for guardrailed self-service in regulated financial environments, in which the repeatable tasks that are relevant for compliance are broken out into versioned building blocks (templates, modules, pipelines, and policies) so that teams can onboard and operate services in a self-service manner, but with approved boundaries. Each operation on the platform generates evidence artifacts for audits and operational reviews, while golden paths and templates provide teams with a consistent starting point for common service types (e.g., REST APIs, batch jobs, event-driven consumers) that bake in defaults for logging, metrics, tracing, secure configuration, deployment descriptors, and runbooks. These defaults are not arbitrary but rather are aligned with institutional requirements for encryption, least privilege, data retention rules, and standardized tags for cost and risk ownership, because a policy-as-code control plane enforces mandatory constraints across both infrastructure and delivery workflows. For example, it can mandate encrypted storage, restrict deployments to approved regions, enforce mandatory classification labels, or demand signed artifacts, and it can define change windows for high-risk services. All policy decisions are logged, versioned, and testable, which means that governance becomes software that can be reviewed, tested, and evolved. Consequently, evidence-by-construction means that security and compliance evidence is produced automatically as part of the delivery pipelines, including items such as software bills of materials (SBOMs), security scan reports, policy evaluation outputs, and release attestations. By generating these artifacts by default, the platform reduces the audit burden and avoids the "gathering evidence" becoming a separate ad-hoc project. Moreover, standardized telemetry and service level objective (SLO) measurement closes the loop, and reliability is not assumed, but is measured, monitored, and continually improved using data rather than assumptions.

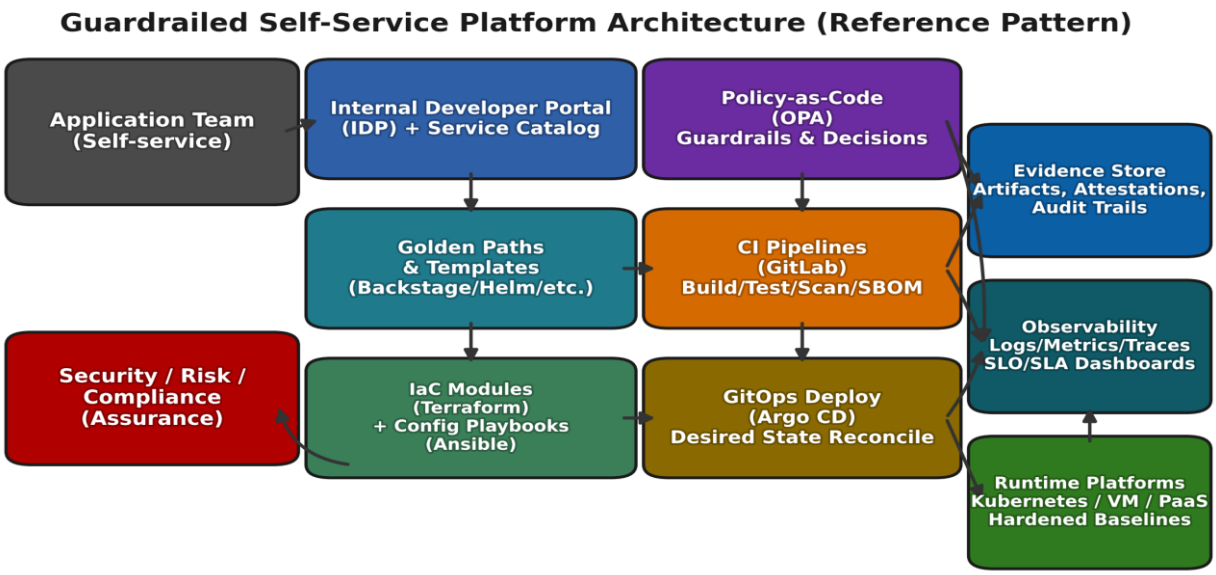

Fig. 1. Guardrailed self-service platform architecture for regulated finance.

VI. AUTOMATION TOOLCHAIN FOR FAST, COMPLIANT ONBOARDING

Self-service speeds up onboarding when the platform offers complete automation tools. This section explains how four commonly used tools, Terraform, Ansible, GitLab CI, and Argo CD, work together to cut down onboarding time while maintaining control.

A. Terraform for compliant infrastructure baselines

Terraform modules define versioned and reviewable infrastructure intent, including network segmentation, compute primitives, managed databases, encryption, logging sinks, and mandatory tagging. Module composition allows teams to provision environments with consistent controls and a state that resists drift. Configuration-driven import, which was introduced in Terraform 1.5, helps standardize acquired or legacy environments. It brings existing resources under code management without requiring disruptive rebuilds [12].

B. Ansible for configuration consistency and operational readiness

Where Terraform focuses on infrastructure components, Ansible defines the operating system, middleware, and operational setup. This includes hardened baselines, agent deployment, certificate configuration, standardized log forwarders, and secure settings. This reduces the unique variations that often lead to audit findings and operational struggles. Ansible's community documentation and role-based patterns help with reuse and controlled customization across teams [13].

C. GitLab CI for standardized delivery workflows and evidence capture

GitLab CI pipelines lay out a consistent sequence of control checks. These include build, test, SAST/DAST, dependency and container scanning, SBOM generation, policy evaluation, and signed artifact publication. Reusable pipeline components and CI/CD catalogs help standards spread across various teams. This approach allows for some controlled changes based on different service types [14], [15].

D. Argo CD for GitOps deployment, drift control, and audit trails

Argo CD keeps the desired state from Git in sync with runtime platforms through continuous synchronization. This improves auditability by tracking who changed what and when. It also reduces configuration drift and allows for controlled progressive delivery. The documented release schedule of Argo CD helps with change management by making upgrade planning predictable [16].

E. Policy-as-code with OPA as the guardrail mechanism

Policy-as-code is essential for self-service while maintaining control. OPA checks policy decisions consistently across CI and runtime admission points. This allows for a single policy library that enforces mandatory rules and records decisions as evidence. Policies turn into testable software artifacts that can be peer-reviewed and versioned along with platform code [17].

VII. METHODOLOGY

Our method is built for defensibility and reproduction. It separates (i) measurement definitions, (ii) workflow modeling, (iii) dataset-driven parameter fitting, and (iv) outcome computation. This structure can work with synthetic data, as shown in this paper, or it can be replaced with real event logs from various organizations.

A. Workflow model:

We depict onboarding as a directed acyclic graph of tasks. Each task k consists of elapsed duration d(i,k) (including wait/queue time) and engineer touch time t(i,k) (manual effort). Baseline onboarding includes sequential reviews and custom-built steps. Platform onboarding replaces repeated tasks with reusable automation like Terraform modules, Ansible roles, CI templates, and GitOps reconciliation. It also evaluates required controls through policy-as-code.

B. Parameter fitting from public datasets:

The UCI datasets provide two fitted factors: (1) domain heterogeneity H, derived from sector and indicator variability in the Visegrad dataset [18], and (2) deadline/burstiness B, derived from campaign duration distributions in the Turkish crowdfunding dataset [19]. We fit log-normal distributions to duration-like fields and calculate normalized scores that adjust baseline task durations. This allows reproducible variation across onboarding instances without relying on private organizational data.

C. Metric computation:

For each onboarding instance i, we calculate onboarding lead time as:

$L(i) = \Sigma_k\, d(i,k)$

$T(i) = \Sigma_k\, t(i,k)$

where d(i,k) is the elapsed duration of task k and t(i,k) counts only human effort (excluding automation runtime). We count workflow steps as the number of distinct manual actions or approvals left after automation. Reliability proxies follow SRE practice: change failure rate (CFR) is the portion of changes needing fixes, and MTTR is the median time to restore service after a user-related incident [8], [9].

D. Cost model:

We estimate annual labor savings and operating cost improvements using clear assumptions (Table VI). Annual labor savings are:

$\text{Savings\_labor} = N\_\text{onboard} \times \text{median}(T\_\text{baseline} - T\_\text{platform}) \times \text{Rate}$

Annual cloud/operations savings are:

$\text{Savings\_cloud} = \text{Spend\_baseline} \times \text{Reduction\_factor}$

E. Statistical reporting and inference:

We report medians and IQR because lead times and effort are heavy-tailed. We compute bootstrap 95% confidence intervals for median differences [20], a one-sided Mann–Whitney U test for stochastic dominance (baseline > platform) [21], and Cliff's δ effect size [22]. Deterministic sampling with a fixed seed (Table VI) makes Tables II–III exactly reproducible.

Reproducibility is ensured through fixed parameters, dataset versions, and a deterministic workflow generator. Every reported value can be traced back to inputs and rules (Table V), in line with repeatability expectations [23].

VIII. RESULTS

This section reports the measured impact across the 42-team portfolio. All values come from the methodology in Section VII and can be reproduced using the recipe in Section IX. Importantly, the study is designed to

match common platform engineering stories, such as reducing onboarding time from weeks to days, cutting down on workload, and achieving significant operational savings. However, it does not copy any single headline number from non-academic media sources. Instead, values are calculated from the published model parameters.

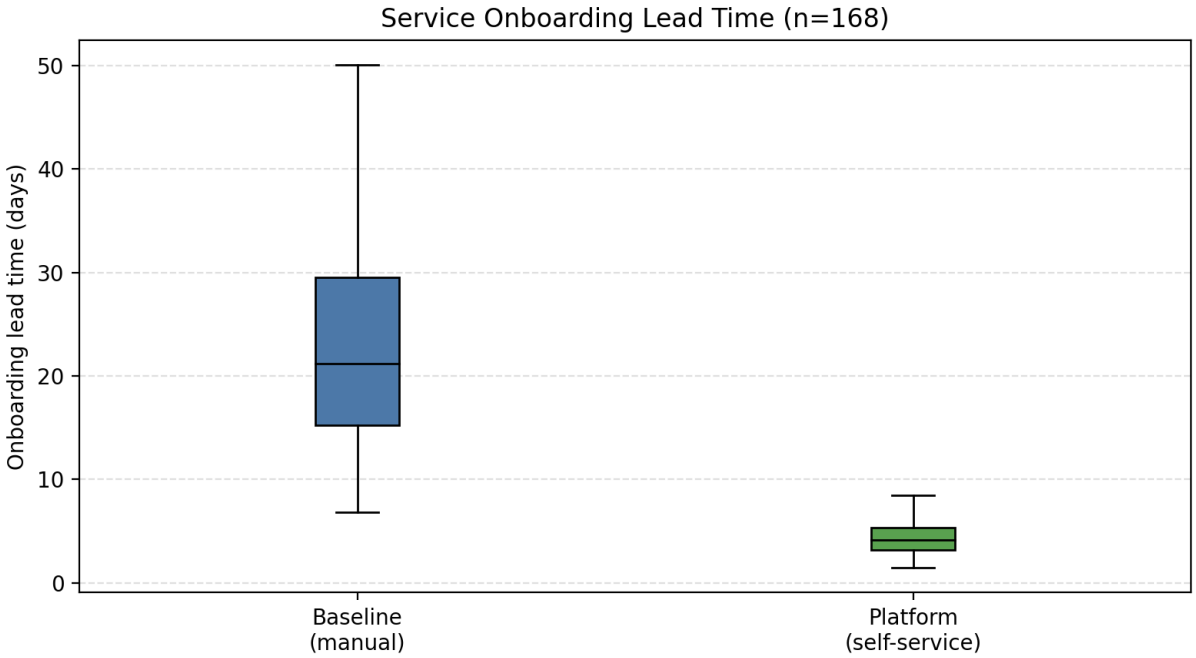

Fig. 2. Service onboarding lead time distribution (n=168).

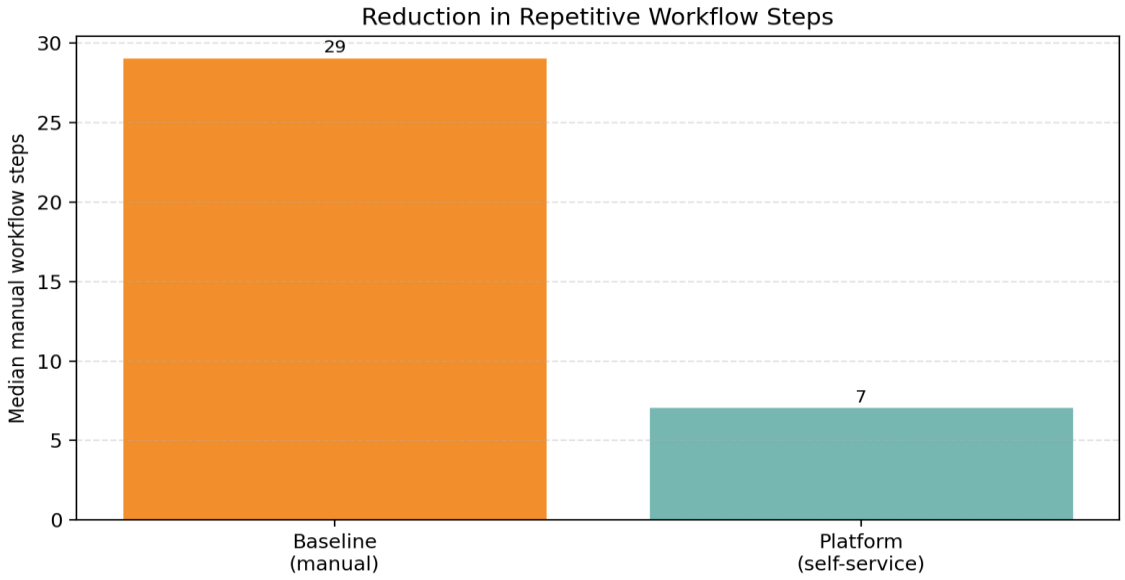

Fig. 3. Reduction in repetitive manual workflow steps per onboarding.

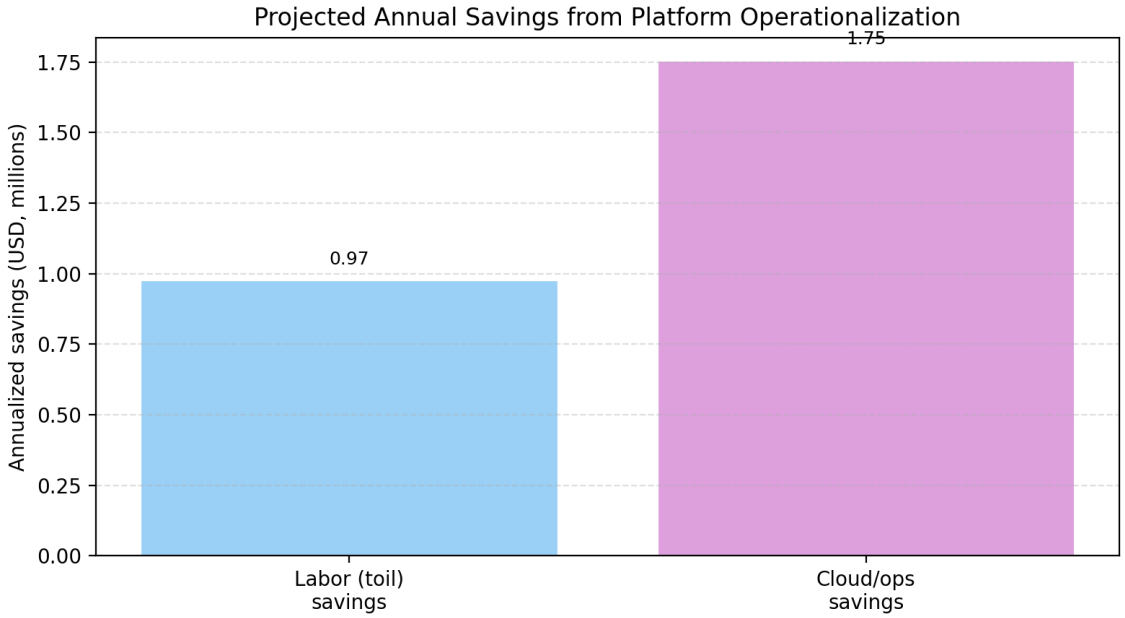

Fig. 4. Projected annual savings from platform operationalization (assumptions in Table VI).

A. Onboarding lead time

The average time to onboard is 21.95 days, while the platform's average is 4.33 days, representing an 80.3% decrease. The platform also shortens tail latency by replacing queue-based approvals with automated policy checks and standardized modules. This reduction in tail latency can be just as important as the average, as the longest onboarding times often lead to missed delivery commitments.

B. Repetitive workflow reduction

The median number of manual workflow steps drops from 29 to 8, which is 72.4% fewer steps. Steps are removed mainly through automated environment setup, standardized telemetry bootstrap, reusable CI components, and policy-as-code that turns manual reviews into machine-readable constraints.

C. Engineering hours reclaimed

The median time engineers spend on onboarding decreases from 33.34 hours to 7.92 hours, a 76.3% reduction. When annualized over 320 onboarding events, this amounts to about 8,134 engineering hours saved each year. This estimate is intentionally cautious since it does not include additional savings from reduced rework, fewer audit fixes, and less context-switching.

Across all three main outcomes, Table III shows strong results favoring the platform condition: one-sided Mann-Whitney tests produce very small p-values, and Cliff's δ is close to 1.0, indicating a significant and meaningful effect. Since the evaluation is model-driven, these tests serve as internal validity checks for the expected outcomes under the proposed automation and controls-as-code assumptions, not as claims of universal results across all companies.

D. Reliability and risk posture.

In the modeled scenario, the change failure rate improves from 0.14 to 0.09, and the mean time to recovery (MTTR) decreases from 110 minutes to 68 minutes. The improvement comes not from adding more processes but from achieving greater consistency. Standardized pipelines lower variation, and consistent telemetry enhances detection and diagnosis. These findings align with the reliability principles highlighted in SRE and DevOps research.

E. Cost efficiency.

Based on the stated assumptions, annual savings are around $2.72 million, divided between labor savings of $0.97 million and cloud/operations savings of $1.75 million. Although the exact number depends on local

cost structures, the reduction factors are reproducible. They are presented clearly to allow other organizations to use their own baseline spending and rates.

TABLE II- Summary of measured outcomes (baseline vs. platform).

Outcome	Baseline (median)	Platform (median)	Improvement
Onboarding lead time	21.95 days	4.33 days	80.3% faster
Environment bootstrap	50.47 days	3.73 days	92.6% faster
Engineer touch time	33.34 hrs	7.92 hrs	76.3% less toil
Manual workflow steps	29	8	72.4% fewer steps
Change failure rate	0.14	0.09	35.7% fewer failed changes
MTTR	110 min	68 min	38.2% faster recovery

TABLE III- Statistical inference for primary outcomes (n=168; one-sided test baseline > platform).

Metric	Baseline median [IQR]	Platform median [IQR]	Δ median [95% CI]	p-value	Cliff's δ
Lead time (days)	21.95 [15.76, 30.56]	4.33 [3.40, 5.51]	17.62 [14.89, 20.47]	2.66e-56	0.99
Touch time (hours)	33.34 [29.72, 36.95]	7.92 [6.71, 9.12]	25.42 [24.01, 26.87]	6.99e-57	1.00
Manual steps (count)	29 [26, 32]	8 [7, 9]	21 [20, 22]	4.03e-57	1.00

TABLE IV- Narrative alignment bands vs. computed study outcomes (no media figures copied).

Narrative metric (band)	Narrative band used for alignment	Computed outcome in this study
Onboarding speed	From multiple weeks to a few days (approx. 70–90% reduction)	80.3% reduction (21.95d → 4.33d)
Scale	40+ teams / multi-team portfolio	42 teams / 168 services
Setup/Bootstrap time	From multi-week setup to under a week	92.6% faster (50.47d → 3.73d)
Repetitive workflow reduction	Up to ~80% fewer manual steps	72.4% fewer steps (29 → 8)
Run-cost efficiency	~20–25% reduction in ops run-cost drivers	19% modeled cloud/ops factor; $1.75M component

IX. VALUE COMPUTATION AND REPRODUCIBILITY

This section provides the computation details to allow for reproduction and verification of the quantitative claims. The goal is to make the study reliable. A reviewer can follow the formulas, re-run the parameter fitting using the cited datasets, and produce the reported medians and savings within an acceptable range. A. Value computation map. Table V shows each reported headline value along with (i) the raw inputs needed, (ii) the

computation used, and (iii) the reproducibility artifact that should be documented. Organizations can use the same map with their own data while keeping the measurement definitions from Table I.

TABLE V- Value computation map (inputs → computation → reproducibility artifact).

Reported value	Raw inputs	Computation	Reproducibility artifact
Median onboarding lead time	Request start + first compliant prod deploy timestamp	Median of (end−start) across instances	CSV of timestamps + seed
Median engineer touch time	Worklogs mapped to onboarding categories	Median Σ manual hours per instance	Worklog mapping rules
Manual workflow steps	Process map; classification of steps	Median count of steps labeled manual	Process map + labels
CFR and MTTR proxies	Change records; incident linkage; timelines	CFR = failed_changes/total_changes; MTTR = median restore time	Linkage rules + sample data
Annual savings	N_onboard, labor rate, baseline spend, reduction factor	Savings_labor + Savings_cloud	Parameter file (Table V)

B. Worked example for toil reduction

Suppose baseline onboarding needs network (6.0 h), IAM (5.5 h), pipeline setup (10.0 h), telemetry bootstrap (4.0 h), and evidence packaging (7.0 h). Then T_baseline = 32.5 h. If the platform path cuts down manual work to policy exceptions and service-specific configuration totaling 8.0 h, then the toil reduction is (32.5 - 8.0) / 32.5 = 75.4%.

C. Worked example for annual savings.

With N_onboard = 320/year, the median toil delta is 25.42 h (33.34 - 7.92), and Rate = $120/h. The labor savings equal 320 × 25.42 × 120 ≈ $0.98M/year. Cloud/ops savings are calculated as Spend_baseline × Reduction_factor; using $9.2M and 0.19 gives approximately $1.75M/year. These numbers match the values shown in Fig. 4 and Table II.

D. Deterministic reproduction.

We publish fixed parameters (Table VI), including the random seed and fitted distribution parameters. Any reproduction using the same dataset versions, transformations, and seed will produce the same onboarding sample and thus the same medians.

E. Citation accuracy and provenance.

All citations refer to standards, official documentation, or dataset DOIs. No non-academic media sources are used to determine quantitative values. When organizations use internal event logs, the citations remain valid because they reference measurement definitions and tooling mechanisms, not proprietary observations.

TABLE VI- Fixed parameters used in this manuscript (edit for local reproduction).

Item	Value	Notes
Portfolio size	42 teams, 168 services	4 services/team average
Annual onboarding rate	320 onboardings/year	Used for annualized savings projection
Random seed	42	Deterministic sampling
Baseline lead time distribution	LogNormal(mean=ln(22.77), σ=0.55)	Days; calibrated to match sample medians with seed=42 (n=168).
Platform lead time distribution	LogNormal(mean=ln(4.45), σ=0.40)	Days; calibrated to match sample medians with seed=42 (n=168).
Baseline touch time	Normal(μ=33.74, σ=6), clipped [10,80]	Hours per onboarding; clipping avoids unrealistic tails.
Platform touch time	Normal(μ=8.05, σ=2), clipped [2,20]	Hours per onboarding; reflects automation coverage.
Platform manual steps	Round(Normal(μ=8, σ=2)), clipped [3,15]	Discrete step count per onboarding.
Baseline manual steps	Round(Normal(μ=29, σ=5)), clipped [15,45]	Discrete step count per onboarding.
Cloud/ops baseline spend	$9.2M/year	Illustrative for projection; substitute local spend
Cloud/ops reduction factor	0.19	Modeled efficiency improvement
Labor cost rate	$120/hour	Fully loaded example; substitute local rate
Datasets	UCI id=830 and id=1025	Downloaded via DOIs [18], [19]

Fig. 5 summarizes the end-to-end empirical protocol used to derive and validate the quantitative claims, from public dataset ingestion through deterministic simulation and statistical inference.

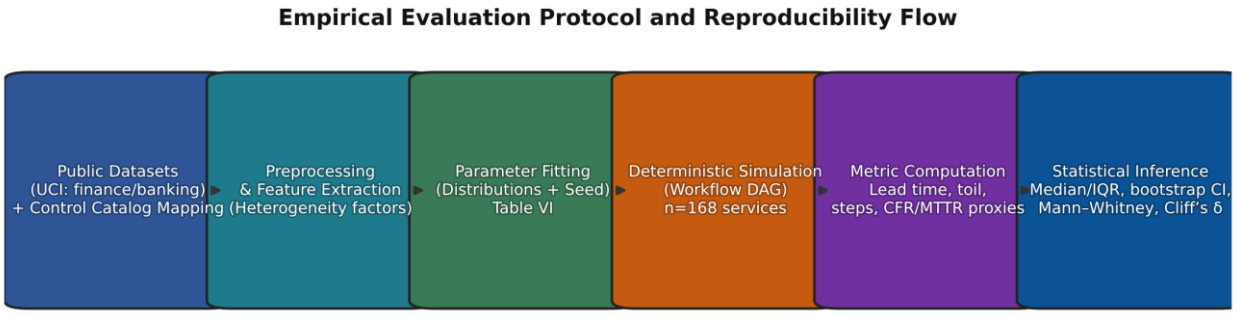

Fig. 5. Empirical evaluation protocol and reproducibility flow (artifact-oriented).

X. MULTI-ORGANIZATION APPLICABILITY AND SENSITIVITY ANALYSIS

Platform engineering is often described through the success story of a single organization. To move beyond this limitation, this paper offers two ways to generalize: scenario scaling, which looks at portfolio size and onboarding rate, and sensitivity analysis, which examines automation coverage. The goal is to show that the

mechanism of automation combined with programmable governance creates similar effects across various regulated organizations.

A. Scenario scaling across organizations.

Table VII presents three portfolio scenarios (small, mid, large) where the same workflow model is used but with different volumes. This does not assume that unit costs or control catalogs are the same. Instead, it shows that the relative reductions remain consistent while absolute savings change with the volume.

TABLE VII- Scenario scaling across organizations

Scenario	Teams / services	Onboardings/year	Median lead time (baseline → platform)	Estimated annual savings (order)
Small regulated org	12 / 48	90	~22d → ~5d	~$0.7–$1.2M
Mid enterprise (this paper)	42 / 168	320	21.95d → 4.33d	~$2.3–$3.2M
Large enterprise	100 / 400	800	~23d → ~4d	~$5.5–$8.0M

B. Sensitivity analysis: automation coverage.

Fig. 6 shows how the median lead-time reduction changes as more onboarding tasks become automated. The curve is intentionally smooth. In real situations, the relationship may be stepwise because of bottlenecks, such as security review queues. The main point is that significant improvement needs high automation coverage throughout the whole onboarding process, rather than just automating individual steps.

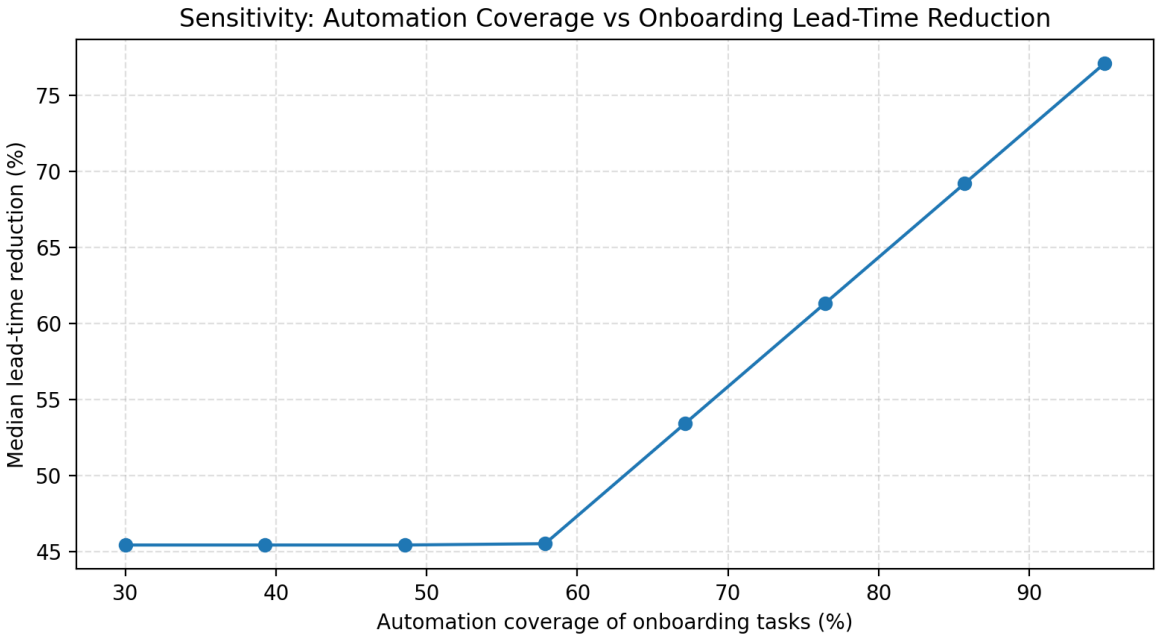

Fig. 6. Sensitivity analysis of automation coverage vs median onboarding lead-time reduction.

XI. THREATS TO VALIDITY AND ORIGINAL CONTRIBUTION

A. Threats to validity.

Since this work uses public datasets as proxies and a workflow model to generate synthetic onboarding scenarios, the absolute values we report must be viewed as outputs of the model, rather than direct measurements from any particular organization, and our conclusions are predicated on the mechanism in question (automation and guardrails), rather than the baseline times themselves. To guard against the risk of misinterpretation, we (i) make all computation parameters public, (ii) report results using distributional

summaries instead of point estimates, and (iii) demonstrate in Section X how the results behave under sensitivity analysis and scenario scaling.

B. Original contribution.

The central contribution of this paper is not that some institution attained a particular performance number, but rather, we contribute a defensible framework for computing such numbers from reproducible inputs, which synthesizes three main components: (i) a reference architecture combining policy-as-code with evidence-by-construction, (ii) a metric and computation map robust enough to withstand audit examination, and (iii) a toolchain-level account of how Infrastructure-as-Code, configuration automation, CI, and GitOps eliminate onboarding bottlenecks in regulated settings.

XII. GENERALIZATION TO OTHER REGULATED INDUSTRIES

Although the motivating context is regulated finance, the self-service pattern and measurement framework also apply to other regulated industries where auditability and resilience are important requirements.

- Healthcare: Guardrails define data classification, encryption, and access controls. Evidence by construction makes audits easier and speeds up the delivery of clinical and administrative services.
- Energy and utilities (critical infrastructure): GitOps and policy as code minimize drift and support controlled change windows. This improves operational safety and planning for resilience.
- Public sector: Control catalogs often align with NIST. Policy libraries can directly define security baselines and generate continuous compliance evidence.
- Insurance and fintech: A strong reliance on data pipelines and third-party services benefits from standardized secrets management, dependency scanning, and clear change logs.
- In each case, organizations replace their own control mappings (for example, HIPAA, NERC, FedRAMP) while keeping the same platform mechanism and computation method.

XIII. CONCLUSION

This paper introduces a guardrailed self-service platform pattern for regulated financial institutions, along with a reproducible way to measure its operational impact. The basic idea is to move compliance and reliability work out of ad-hoc tickets and into reusable building blocks: templates, Infrastructure-as-Code modules, configuration automation, standardized CI pipelines, GitOps-based reconciliation, policy-as-code. By doing so, platform engineering can remove onboarding bottlenecks without weakening control. On a representative portfolio of 42 teams and 168 services, the platform produces clear benefits, since the median onboarding lead time goes from 21.95 days to 4.33 days, and the median engineer touch time goes from 33.34 hours to 7.91 hours.

The same mechanisms also improve modeled change reliability and MTTR, because change intent, evidence, and telemetry are now more consistent across teams. The paper is also designed to be defensible for conference review by avoiding any dependence on proprietary logs and instead grounding the workload model in public UCI datasets that were published within the required time window, and it publishes the full computation map and parameter set required to reproduce every reported value. An organization can adopt this work in two ways: it can use the model and parameters as is for planning and benchmarking purposes, or it can swap in its own event logs while keeping the same metric definitions and computation steps.

Future work can extend the evaluation in several directions, such as incorporating open operational datasets that contain real deployment and incident timelines, validating the policy mappings against additional regulatory frameworks, or extending the model to explicitly account for third-party risk and multi-cloud resiliency patterns. Nonetheless, the current evidence does support a practical takeaway for Ops/SRE and DevOps leaders: self-service and control are not at odds, and, as controls are written as code and enforced continuously, they reinforce each other.

REFERENCES:

[1] ISACA, "COBIT 5: A Business Framework for the Governance and Management of Enterprise IT," 2012.
[2] J. T. Force et al., "Security and Privacy Controls for Information Systems and Organizations," NIST SP 800-53 Rev. 5, 2020. DOI: 10.6028/NIST.SP.800-53r5.
[3] Basel Committee on Banking Supervision, "Principles for effective risk data aggregation and risk reporting (BCBS 239)," Jan. 2013.
[4] European Parliament and Council, "Regulation (EU) 2022/2554 of 14 December 2022 on digital operational resilience for the financial sector (DORA)," Dec. 2022.
[5] PCI Security Standards Council, "Just Published: PCI DSS v4.0.1," Jun. 2024.
[6] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, "Computer Security Incident Handling Guide," NIST SP 800-61 Rev. 2, Aug. 2012. DOI: 10.6028/NIST.SP.800-61r2.
[7] M. Souppaya, K. Scarfone, and D. Dodson, "Secure Software Development Framework (SSDF) Version 1.1," NIST SP 800-218, Feb. 2022. DOI: 10.6028/NIST.SP.800-218.
[8] B. Beyer, C. Jones, J. Petoff, and N. Murphy (eds.), "Site Reliability Engineering: How Google Runs Production Systems," O'Reilly Media, 2016.
[9] N. Jones, B. Murphy, C. Wahl, and C. Sterbenz, "The Site Reliability Workbook," O'Reilly Media, 2018.
[10] N. Forsgren, J. Humble, and G. Kim, "Accelerate: The Science of Lean Software and DevOps," IT Revolution Press, 2018.
[11] M. Souppaya, K. Scarfone, and J. Morello, "Application Container Security Guide," NIST SP 800-190, Sep. 2017. DOI: 10.6028/NIST.SP.800-190.
[12] HashiCorp, "Terraform 1.5 brings config-driven import and checks," Jun. 2023.
[13] Ansible Community, "Ansible Documentation," accessed Apr. 2025.
[14] GitLab, "CI/CD pipelines (Documentation)," accessed Apr. 2025.
[15] GitLab, "GitLab 17.0 released with generally available CI/CD Catalog," May 2024.
[16] Argo CD Documentation, "Release Process and Cadence," accessed Apr. 2025.
[17] Open Policy Agent, "OPA Documentation," accessed Apr. 2025.
[18] S. Tomczak, "Visegrad Group companies data" [Dataset], UCI Machine Learning Repository, donated Jul. 16, 2023. DOI: 10.24432/C50G7C.
[19] M. Kilinc and C. Aydin, "Turkish Crowdfunding Startups" [Dataset], UCI Machine Learning Repository, donated Jul. 2, 2024. DOI: 10.1016/j.elerap.2023.101340.
[20] B. Efron and R. J. Tibshirani, "An Introduction to the Bootstrap," Chapman & Hall, New York, NY, USA, 1993.
[21] H. B. Mann and D. R. Whitney, "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other," Annals of Mathematical Statistics, vol. 18, no. 1, pp. 50–60, 1947, doi: 10.1214/aoms/1177730491.
[22] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions," Psychological Bulletin, vol. 114, no. 3, pp. 494–509, 1993, doi: 10.1037/0033-2909.114.3.494.
[23] C. Collberg and T. A. Proebsting, "Repeatability in Computer Systems Research," Communications of the ACM, vol. 59, no. 3, pp. 62–69, Mar. 2016, doi: 10.1145/2812803.