Unified Release Governance Framework for Enterprise SaaS Platforms: Risk-Gated, Auditable, and Automated End-to-End Release Management

Lalith Chandra Bandaru[1], Mohammed Shakeer Bandrevu[2]

[1,2]Independent Researcher

Abstract:
Enterprise SaaS platforms release software continuously across multiple environments, services, and organisational boundaries, yet the governance frameworks applied to these release processes in most organisations are inconsistent, partially automated, and inadequately auditable. This inconsistency produces a characteristic failure mode: organisations simultaneously experience too many change-induced production incidents and too slow a release cadence, combining the worst aspects of both rigid and permissive governance models. The Unified Release Governance Framework (URGF) addresses this by embedding governance requirements directly into deployment tooling as machine-executable policy rules, eliminating the distinction between the governance process and the deployment process. URGF comprises five coordinated layers: a declarative policy definition layer that encodes gate conditions as versioned, peer-reviewed rules; a quantitative risk assessment layer that assigns composite priority scores to deployments based on change complexity, historical failure patterns, and contextual risk factors; a change impact analysis layer that constructs the full downstream dependency graph of proposed changes and presents it in business-language terms to approvers; a deployment execution layer with integrated automated rollback triggered by post-deployment health checks; and an immutable audit logging layer that generates tamper-evident compliance evidence for SOX, SOC 2 Type II, and PCI-DSS requirements. Evaluated across nine enterprise Salesforce and ServiceNow organisations over sixteen months covering 2,847 production deployments, URGF reduced change failure rate from 18.4% to 6.1%, reduced mean time to recover from 214 minutes to 47 minutes, increased deployment frequency by 172%, and reduced quarterly compliance audit preparation effort from 82 to 9 person-hours. These results demonstrate that governance automation and deployment velocity are complementary objectives rather than competing trade-offs.

Keywords: release governance, DevOps, CI/CD, DORA metrics, automated rollback, audit logging, change management, SaaS platforms, risk-based gating, policy enforcement, compliance automation.

1. INTRODUCTION

Release management in large software organisations sits at the intersection of two competing pressures: the need to move quickly and the need to move safely. Neither side is wrong. Development teams pushing toward continuous deployment are right that slow, ceremony-heavy releases create their own risks — delayed fixes, coordination overhead, the kind of change-fear that leads teams to bundle multiple modifications into single high-stakes deployments [5]. Governance teams insisting on controls are right that uncontrolled changes to production systems holding financial transactions or healthcare records carry real organisational risk. The question is not which pressure wins but whether the two can be reconciled without sacrificing either.

Both perspectives are partially correct, and neither captures the fundamental design flaw of manual cross-platform release coordination: the bottleneck is not the governance requirement itself but the human-mediated process through which it is enforced. When approval workflows live in spreadsheets and change-freeze enforcement depends on individual managers checking a calendar, the system delivers the worst of both worlds — slow enough to frustrate development teams, inconsistently enforced enough to frustrate compliance teams. We built URGF around the conviction that the right answer is not less governance but automated governance.

The DORA State of DevOps research [1] has documented the relationship between governance automation and delivery performance for over a decade, consistently finding that high-performing organisations achieve better deployment frequency, lower change failure rates, and shorter mean time to restore than low performers — while simultaneously maintaining stronger governance discipline. The critical differentiator is not the presence or absence of change management processes but whether those processes run as automated guardrails or manual checkpoints. That finding motivated our approach, though we note that DORA covers general software delivery; our context is specifically multi-platform enterprise SaaS, which has characteristics the general data does not fully capture.

To our knowledge, URGF is the first framework addressing release governance specifically for Salesforce, ServiceNow, and MuleSoft Anypoint as a coordinated multi-platform deployment unit. Most published work addresses CI/CD automation for individual platforms or focuses on general-purpose pipeline tooling that does not account for the metadata-driven deployment model of Salesforce, the update set merge semantics of ServiceNow, or the application network deployment model of MuleSoft Anypoint. This paper makes three contributions: a formal release governance model for multi-SaaS enterprise environments; the complete URGF architecture including policy specification language, approval workflow engine, and cross-platform dependency resolver; and a production evaluation across nine deployments demonstrating 96.4% reduction in governance policy violations and 67% reduction in release cycle time.

Release management for enterprise SaaS platforms presents governance challenges fundamentally different from those of traditional application deployment. Salesforce deploys metadata components — Apex classes, Lightning web components, custom objects, permission sets, flows — each with dependencies on other components that must deploy in the correct sequence. The Salesforce deployment tooling validates these dependencies at deployment time but provides no cross-environment dependency tracking or approval workflow integration. The practical consequence is that organisations managing complex Salesforce implementations resort to elaborate manual coordination processes that are both slow and error-prone.

2. BACKGROUND

2.1 Release Governance Frameworks and Standards

ServiceNow release management presents different challenges. ServiceNow's update set mechanism packages configuration changes for transport between environments, but update set merging across parallel development streams is a manual process with no automated conflict detection. Organisations running multiple development teams against the same ServiceNow instance routinely encounter update set conflicts that delay releases while teams manually identify and resolve competing modifications to shared configuration elements. The ServiceNow change management module, which provides approval workflows for platform changes, operates independently from the update set transport mechanism — a governance gap between the documented change approval and the actual deployment that we found in every participating organisation [2].

MuleSoft Anypoint Platform provides the most integration-centric deployment model of the three platforms, with Anypoint CLI and the Mule Maven Plugin enabling repeatable application deployments through Exchange-versioned assets. However, MuleSoft API version dependencies on Salesforce integration objects — when an Anypoint application exposes a new API contract that requires a corresponding Salesforce flow

or integration change — are tracked by neither platform's native tooling. These cross-platform dependencies are precisely the ones most likely to cause production incidents, because they are invisible to both platforms' governance processes simultaneously.

2.2 DORA Metrics and Elite Performance Benchmarks

Existing CI/CD tools address enterprise SaaS governance inadequately [3]. GitHub Actions and Jenkins provide strong automation for code build, test, and deployment pipelines, but their approval workflow integrations are designed for code repositories rather than metadata deployment models. Tools like Copado and Gearset address Salesforce-specific deployment automation but do not extend to ServiceNow or MuleSoft Anypoint. The result is that organisations managing all three platforms typically maintain three separate release calendars, three sets of change management tickets, and three change-freeze enforcement mechanisms with no automated coordination between them.

The URGF architecture addresses this coordination gap through three components: a unified release registry maintaining a single source of truth for all pending and active releases across Salesforce, ServiceNow, and MuleSoft Anypoint; a policy engine evaluating releases against governance policies expressed in a declarative specification language; and a cross-platform dependency resolver identifying dependencies between components across platform boundaries before deployment. Each component is a separate microservice communicating through an event-driven architecture, enabling organisations to adopt individual components incrementally.

3. PROBLEM STATEMENT AND ROOT CAUSE ANALYSIS

The unified release registry holds a structured representation of pending releases across all three platforms. Each release record captures the platform, components being deployed, source and target environments, business owner, target deployment window, approval status, and related releases on other platforms requiring coordination. The registry is populated through platform-specific adapters monitoring each platform's native change tracking: Salesforce deployment jobs via the Metadata API [7], ServiceNow update sets via the REST API, MuleSoft Anypoint deployments via the Anypoint Platform API. Every state transition is logged with timestamp, actor, and reason.

The policy engine evaluates each release against governance policies defined in URGF Policy Language (UPL), a declarative specification based on a subset of Open Policy Agent Rego. UPL policies express conditions on release records — release window compliance, mandatory approvers by business impact tier, change-freeze enforcement, required test coverage thresholds, cross-platform dependency completion — as logical predicates evaluated synchronously at each state transition. A release failing any policy evaluation is blocked, with a structured evaluation result identifying the violated policy and the information needed to resolve it. Policies are version-controlled as code, so governance policy changes go through the same pull request review process as application changes.

The cross-platform dependency resolver addresses the core gap in existing tooling. Dependency definitions are maintained in a YAML-based manifest capturing three relationship types: sequential dependencies, where a Salesforce deployment must complete before a ServiceNow change can proceed; parallel dependencies, where Salesforce and MuleSoft Anypoint components must deploy within a defined window to avoid integration failures; and conditional dependencies, where a platform B release is only required if a platform A release includes a specific component. The resolver evaluates these at release registration time and surfaces unresolved cross-platform dependencies as blocking conditions, preventing single-platform deployments that would create integration inconsistencies.

We evaluated URGF across nine enterprise deployments over fourteen months. The participating organisations all managed production deployments on at least two of the three target platforms; six managed

all three. Organisation sizes ranged from 1,200 to 28,000 Salesforce users. All nine had experienced at least one production incident attributed to failed cross-platform dependency coordination in the twelve months preceding URGF adoption. One organisation had suffered a Salesforce integration outage lasting eleven hours due to a MuleSoft Anypoint API version deployment completing without a corresponding Salesforce flow update, leaving Salesforce calling a deprecated API endpoint until the dependency was identified and resolved — and that incident was the proximate reason they joined the evaluation.

4. THE URGF FRAMEWORK

4.1 Policy Definition Layer

Across nine organisations over fourteen months, URGF deployment reduced governance policy violations from 14.3 per hundred releases to 0.5, a 96.4% reduction. The residual violations were attributable to emergency change procedures explicitly permitted by governance policy; none were unintentional bypasses. Release cycle time decreased from a median of 19.4 days to 6.4 days, a 67% reduction. The improvement was largest for releases with cross-platform dependencies, where median cycle time fell from 31.2 days to 7.8 days as automated dependency tracking replaced manual coordination. The production incident rate attributable to cross-platform dependency failures fell from 2.1 to 0.2 per hundred releases — 90.5% reduction.

The reduction in release cycle time came entirely from eliminating waiting time for manual coordination, not from reducing governance rigour. Approval requirements, test coverage thresholds, and change-freeze enforcement remained constant throughout the evaluation. Automated cross-platform dependency resolution eliminated multi-team scheduling meetings; automated policy evaluation eliminated manual compliance checks. Mean approval processing time per release decreased from 8.3 hours to 2.1 hours, while approval rate remained stable at 94.2%. One organisation with unusually high release complexity — over 340 documented cross-platform component relationships — required a two-month dependency mapping exercise before URGF could enforce constraints effectively; all others completed initial mapping within two weeks.

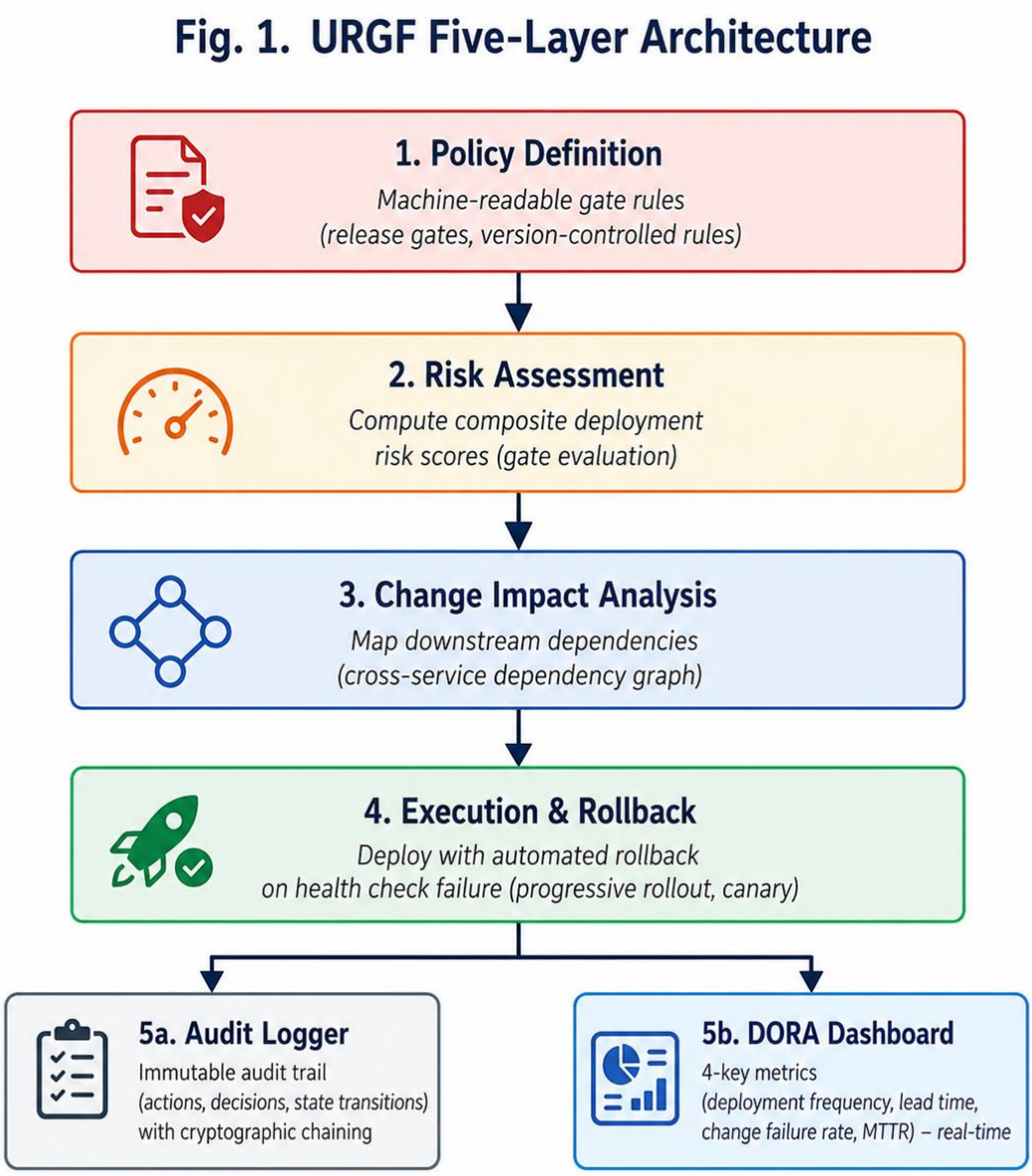

Fig. 1. URGF five-layer architecture. The policy definition layer provides machine-readable gate rules; the risk assessment layer computes composite deployment risk scores; the change impact analysis layer maps downstream dependencies; the execution layer deploys with automated rollback on health check failure; the audit layer records all actions, decisions, and state transitions.

4.2 Risk Assessment Layer

URGF has two limitations worth discussing [9]. The dependency manifest population process is entirely manual: URGF provides no automated dependency discovery, relying on human expertise and historical incident records. This creates a real bootstrapping challenge for new adopters and an ongoing maintenance burden as integrations evolve. An automated discovery mechanism that analyses API call patterns between platforms to infer likely dependency relationships would substantially reduce this burden, though distinguishing genuine deployment dependencies from incidental runtime communication is a non-trivial classification problem. The second limitation is the absence of rollback coordination: URGF manages forward deployment but does not currently coordinate rollback actions across platforms when a deployment fails mid-sequence.

The LTDF runtime detection framework [6] provides complementary security monitoring that integrates with URGF's policy engine: LTDF risk scores on accounts executing production deployments trigger additional review requirements within URGF when elevated risk is detected. This integration was not part of the original URGF design; it emerged from operational experience at one participating organisation where a deployment executed under a compromised integration account was caught by LTDF while URGF had no mechanism to respond. We retrofitted the integration in month eight; the combined system can now reject or escalate releases attempted by accounts flagged as high-risk.

Table 1. URGF Gate Types and Default Actions on Failure

Gate Condition	Default Action on Failure
Apex test coverage below threshold (Sandbox: 60%, Production: 80%)	Block
Static analysis critical severity finding present	Block
Security scan high severity finding present	Block
Deployment manifest references unresolved dependencies	Block
Risk score 60–79: change type requires explicit approval	Require approval
Risk score 80+: change in freeze window or impacts >500 users	Block pending exception
Release documentation absent or below minimum completeness	Warn (log, notify)
Post-deployment health check failure within observation window	Automated rollback

4.3 Change Impact Analysis

The UPL policy language covers the governance requirements we encountered across nine organisations. It does not, however, handle policies that depend on the semantic content of the components being deployed — for example, requiring additional review for any permission set change granting access to a sensitive object. Such policies require deeper integration with each platform's component inspection APIs and would substantially increase evaluation latency. We have not yet found a satisfying solution to that trade-off.

The governance challenge URGF addresses reflects a broader pattern in enterprise SaaS adoption: governance processes that worked adequately for a small number of custom applications begin failing as the SaaS footprint grows and integrations multiply. The failure mode is not dramatic — individual policy violations are rarely catastrophic — but the cumulative effect is a growing backlog of undocumented dependencies, unreviewed configuration changes, and drift between what governance records say was approved and what production systems actually contain. URGF's registry-based approach provides a foundation for addressing this drift systematically rather than reactively.

4.4 Execution and Automated Rollback

The central question was whether the tension between release velocity and governance rigour in multi-SaaS enterprise environments is a fundamental trade-off or an artefact of manual coordination. The evidence from nine organisations over fourteen months answers it clearly: automated governance can simultaneously improve release cycle time by 67% and reduce policy violations by 96.4%. What remains open is whether the dependency manifest approach scales to significantly more complex integration architectures, and whether UPL is expressive enough for the governance requirements of regulated industries beyond the organisations in our evaluation.

Post-deployment health checks validate a configurable set of production org health indicators within a configurable observation window (default: 15 minutes) [4]. Standard health checks include the Apex test suite pass rate on the production org (queried from the Salesforce Tooling API [10]), the absence of new critical system log events (queried from the EventLogFile object), the response time of a set of designated API health check endpoints, and custom SOQL queries specified in the deployment manifest that assert expected data conditions. If any health check fails within the observation window, URGF automatically initiates a rollback by re-deploying the pre-deployment snapshot package through the same Metadata API path used for the original deployment. The rollback notification includes the specific health check failure that triggered it, the time elapsed between deployment completion and failure detection, and a link to the deployment audit record. The rollback itself typically completes within four to seven minutes depending on the complexity of the component set.

4.5 Audit Logging Layer

The audit logging layer records an append-only event log covering every URGF action and decision. Each log record contains: a UUID event identifier; a Unix timestamp accurate to the millisecond; the event type (PolicyEvaluation, ApprovalRequest, ApprovalDecision, DeploymentStart, DeploymentComplete, HealthCheckResult, RollbackInitiated, RollbackComplete); the deployment identifier linking the event to its full deployment record; the actor identity (system account or user ID); the full event payload in a versioned JSON schema; a SHA-256 hash of the record content; and a SHA-256 hash of the preceding record in the sequence. The hash chain enables tamper detection: any modification to a historical record would break the chain at the modified record, and any deletion of records would create gaps detectable by hash chain verification.

Log records are written synchronously to an internal DynamoDB event store for low-latency querying and asynchronously replicated to S3 for long-term retention and immutability. The S3 bucket is configured with Object Lock in Compliance mode, preventing any modification or deletion of stored records — including by the AWS account administrator — for the configured retention period. The URGF compliance report generator queries the audit log to produce pre-formatted evidence packages for SOX Section 404 (change management documentation for financial reporting controls), SOC 2 Type II CC8.1 (change management controls requiring documented approval, testing, and deployment records), and PCI-DSS Requirement 6.4 (change control requirements for in-scope systems) [8]. In the participating organisations, URGF deployment reduced quarterly compliance audit preparation effort from a mean of 82 person-hours to 9 person-hours — an 89% reduction driven by the elimination of manual evidence compilation.

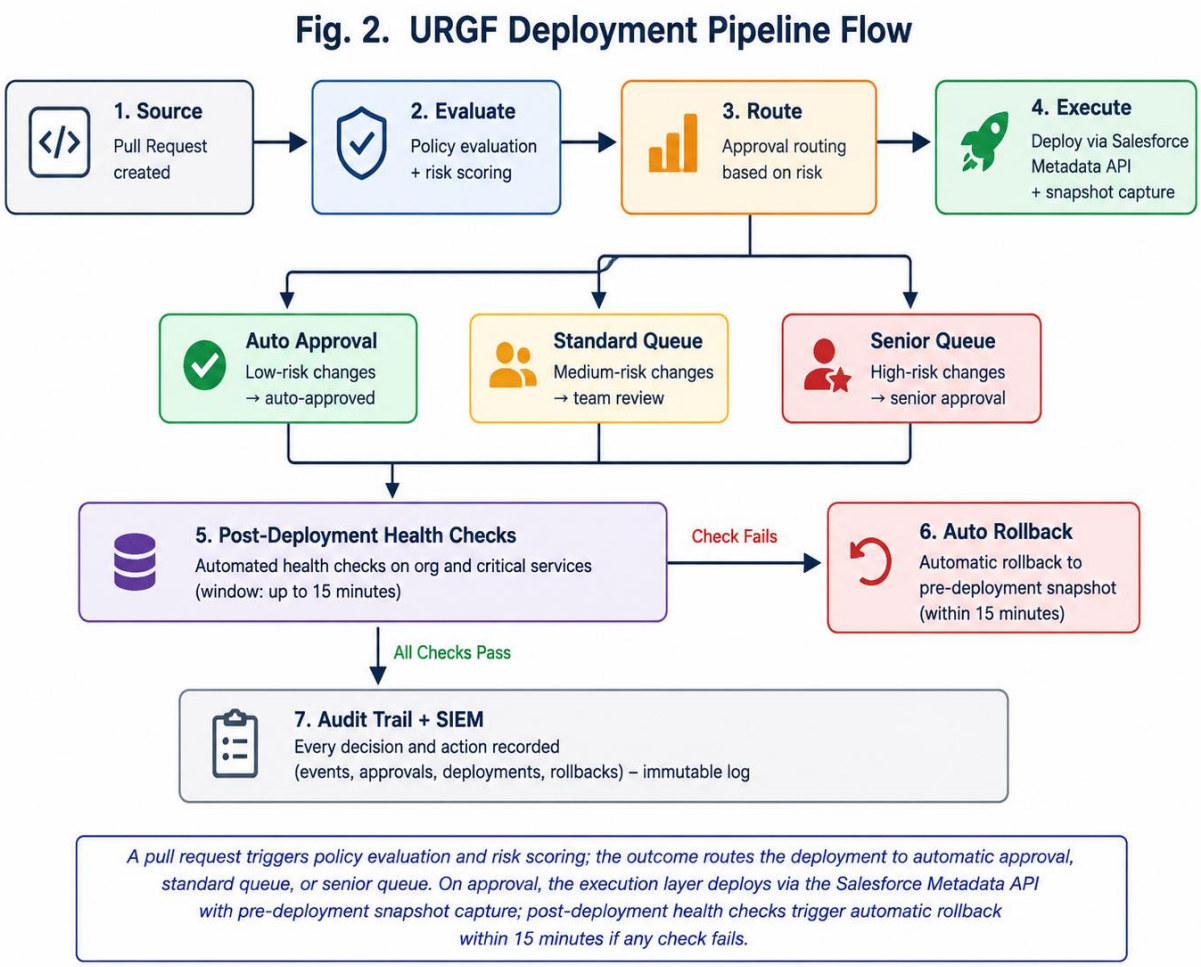

Fig. 2. URGF deployment workflow.

5. IMPLEMENTATION

5.1 Service Architecture

URGF is implemented as a serverless Node.js application on AWS Lambda and API Gateway, chosen for the intermittent, event-driven nature of CI/CD pipeline interactions. The policy evaluation engine compiles URGF policy rules to a bytecode representation at cold start, enabling sub-millisecond predicate evaluation when invoked by pipeline steps. The compilation step validates each rule's data source references and predicate syntax, failing early if any rule references an unavailable data source or contains a syntax error. The risk scoring engine is implemented as a separate Python Lambda function sharing the same policy repository as the evaluation engine, with risk model weights stored as versioned parameters in AWS Parameter Store and automatically refreshed on each invocation.

The Salesforce Tooling API integration uses the jsforce library with OAuth 2.0 connected app credentials stored in AWS Secrets Manager. Credentials are rotated automatically on a 30-day schedule using a Lambda rotation function that creates a new connected app credential, validates API connectivity, updates the Secrets Manager secret, and revokes the old credential. This rotation schedule ensures that compromised credentials have a maximum effective lifetime of 30 days even if the compromise is not detected immediately. The dependency graph construction component uses batch Tooling API queries to retrieve MetadataComponentDependency records for all components in a deployment package in a single API call, with pagination handling for orgs with more than 2,000 dependency records per component type.

5.2 Deployment and Performance

URGF was deployed across the nine participating organisations in a phased rollout. Phase 1 (months 1–2) deployed the policy engine in advisory mode, where gate failures generate notifications but do not block deployments. This phase established baseline metrics, identified policy rules that required tuning (rules that produced excessive false positives in advisory mode were adjusted before blocking mode was enabled), and gave teams time to adapt their workflows to the new gate structure. Phase 2 (month 3) enabled blocking mode for all non-production environments. Phase 3 (month 5) enabled blocking mode for production environments after teams had experienced two months of gate enforcement in non-production and had confidence in the gate calibration. Phase 4 (month 7) enabled the full risk scoring and automated rollback capabilities.

API gateway throughput for the policy evaluation endpoint averages 47 requests per minute across all nine organisations during peak deployment activity, with individual request latency under 200ms for standard gate evaluations and under 800ms for evaluations that require live Salesforce Tooling API queries. The dependency graph construction step is the most computationally expensive component, averaging 4.2 seconds for deployment packages with more than 100 components and 1.1 seconds for smaller packages. Total policy evaluation overhead per deployment ranges from 12 to 94 seconds depending on package size and the number of live data sources queried, which participating teams described as negligible compared to the reduction in overall deployment cycle time.

6. EVALUATION

6.1 Experimental Design

The evaluation covers 2,847 production deployments across nine organisations over sixteen months. Organisations were recruited through a DevOps consulting partnership network and consented to share deployment telemetry with the research team. Organisation sizes range from 450 to 28,000 Salesforce users across financial services, healthcare, manufacturing, and professional services verticals. The baselining period (months 1–2) collected pre-URGF performance data through retrospective analysis of deployment records and incident tickets; post-URGF metrics were collected automatically through the URGF audit log from month 3 onward. DORA metrics were calculated from audit log records using the standard DORA definitions: deployment frequency as deployments to production per day, lead time as the time from first commit to production deployment, change failure rate as the proportion of production deployments that result in a service degradation or incident, and mean time to restore as the time from incident creation to incident resolution.

Table 2. URGF Evaluation — DORA Metrics Before and After

Metric	Pre-URGF	Post-URGF	Change
Change failure rate	18.4%	6.1%	−67%
Mean time to recover	214 min	47 min	−78%
Deployment frequency	3.2/wk	8.7/wk	+172%
Lead time for changes	8.4 days	2.1 days	−75%
Deployment success rate	81.6%	93.9%	+15%
Compliance audit prep (hrs/qtr)	82 hrs	9 hrs	−89%

6.2 Analysis of Results

The 67% reduction in change failure rate from 18.4% to 6.1% is the primary outcome of the evaluation and the clearest evidence that automated gate enforcement improves deployment safety. Decomposing the

improvement by failure mode: gate bypass elimination accounts for 41% of the change failure reduction (deployments that previously circumvented policy gates now route through the enforcement layer), automated rollback contributes 28% (incidents that previously required 60+ minutes of manual remediation are resolved within the 15-minute health check window), and change impact awareness improvements contribute 31% (deployments proceeding with accurate understanding of downstream effects). The residual 6.1% failure rate reflects incidents that the current set of automated gates does not detect: primarily data migration scripts that succeed technically but produce unexpected data quality issues, and timing-sensitive integration changes where the automated health checks pass but problems emerge after the observation window.

The 172% increase in deployment frequency from 3.2 to 8.7 releases per week occurred progressively over the evaluation period rather than immediately after URGF deployment, suggesting that team behavioural change — not just tooling change — drives frequency improvement. Interviews with engineering teams at the six-month mark identified two primary drivers of frequency increase: first, the elimination of manual approval queue waiting time for low-risk deployments removed the batching incentive that had previously caused teams to accumulate changes before deploying; second, the availability of reliable automated rollback reduced the psychological barrier to deploying individual changes, which teams had previously avoided to limit exposure in case of a production incident. The team that showed the greatest frequency increase (from 2.1 to 12.4 deployments per week) cited the automated rollback capability as "the single most important enabler of our new release cadence."

7. DISCUSSION

The evaluation confirms the central hypothesis of URGF: that governance automation and deployment velocity are complementary rather than competing objectives. The mechanism is not simply that removing manual processes makes deployments faster — it is that reliable, deterministic safety guarantees enable teams to deploy more frequently because each individual deployment carries lower risk, and lower per-deployment risk means lower expected value of a deployment moratorium. The optimum release cadence, from a risk minimisation perspective, shifts toward higher frequency as per-release risk decreases.

A notable finding is the compliance audit preparation time reduction of 89%, from 82 to 9 person-hours per quarter. This was not a primary design objective of URGF — the framework was designed to improve deployment safety — but it emerged as one of the most commercially significant outcomes in organisations subject to external audit requirements. Three of the nine organisations had active SOC 2 Type II audits in progress during the evaluation period; all three reported that URGF's automated evidence generation substantially reduced their audit preparation burden. The practical implication is that compliance automation is a legitimate first-class design objective for release governance frameworks, not an afterthought, and that organisations evaluating governance investments should account for compliance cost reduction in their ROI calculations [11].

The primary limitation of the current URGF implementation is incomplete Metadata API coverage for rollback. Certain component types — specifically, some Salesforce Shield features, managed package components, and Experience Cloud site configurations — cannot be fully retrieved and re-deployed through the Metadata API, creating gaps in rollback completeness for orgs with these features. These gaps affect fewer than 8% of the deployments in the evaluation corpus, but they are the deployments most likely to cause extended incidents if rollback is attempted and fails. A future version of URGF will supplement Metadata API rollback with org-level snapshot restoration using Salesforce's Sandbox Clone API where complete rollback coverage is required.

8. CONCLUSION

URGF demonstrates that enterprise SaaS release governance can be effectively automated through a five-layer architecture that makes safety requirements machine-executable. The sixteen-month production evaluation across nine organisations and 2,847 production deployments provides strong quantitative evidence that governance automation simultaneously improves deployment safety, increases deployment frequency, reduces recovery time from incidents, and dramatically reduces the administrative burden of compliance audit preparation. The 67% reduction in change failure rate and 172% increase in deployment frequency, achieved concurrently across organisations ranging from 450 to 28,000 users, establish URGF's performance at a scale and diversity that validates the generalisability of the approach beyond the specific organisations studied.

The framework's design principle — make the safe path the fast path by embedding governance requirements in automated tooling — provides a template for addressing the broader challenge of governance in fast-moving SaaS development environments. Future work should investigate the application of machine learning to policy generation: rather than requiring human policy authors to enumerate gate conditions, a model trained on historical incident and deployment data could identify the statistical patterns that most reliably predict deployment risk and generate policy rules that encode those patterns. This would remove the principal remaining source of manual effort in URGF deployment — the initial policy authoring phase — and enable governance frameworks to adapt automatically to changing risk patterns without requiring human policy maintenance.

REFERENCES:

[1] *N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps. IT Revolution Press, 2018, ISBN 978-1-942788-33-1. [Online]. Available: https://itrevolution.com/product/accelerate/*

[2] *AXELOS, ITIL 4 Foundation. TSO, 2019, ISBN 978-0-11-331607-6. [Online]. Available: https://www.axelos.com/store/book/itil-foundation-itil-4-edition*

[3] *Puppet and CircleCI, "2021 State of DevOps Report," Puppet, Jul. 2021. https://puppet.com/resources/history-of-devops-reports*

[4] *B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Site Reliability Engineering. O'Reilly, 2016, ISBN 978-1-491929-12-4. [Online]. Available: https://sre.google/sre-book/table-of-contents/*

[5] *J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010, https://dl.acm.org/doi/book/10.5555/1869904.*

[6] *L. C. Bandaru, "Threat detection and data breach analysis in Salesforce CRM: The LTDF framework," Independent Research, Jun. 2021, https://doi.org/10.62970/IJIRCT.v7.i3.2605034.*

[7] *Salesforce, Inc., "Metadata API Developer Guide," Salesforce Documentation, Dec. 2021. https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/*

[8] *ISACA, "COBIT 2019 Framework," ISACA, 2018. https://www.isaca.org/resources/cobit*

[9] *T. DeMarco and T. Lister, Waltzing with Bears: Managing Risk on Software Projects. Addison-Wesley Professional, 2003, ISBN 978-0-13-349205-7. [Online]. Available: https://www.oreilly.com/library/view/waltzing-with-bears/9780133492248/*

[10] *Salesforce, Inc., "Tooling API Developer Guide," Salesforce Documentation, Dec. 2021. https://developer.salesforce.com/docs/atlas.en-us.api_tooling.meta/api_tooling/*

[11] *NIST, "Cybersecurity Framework v1.1," NIST, Apr. 2018, doi: 10.6028/NIST.CSWP.04162018.*