# A Systematic Review of CI/CD Pipeline Strategies in Salesforce DevOps: Tools, Practices, and Deployment Outcomes

OLANIYI BADMUS[1], ADETOMIWA A. DOSUNMU[2], DAVID EXCEL OZOWARA[3]
[1]*New Zealand Post Group, New Zealand*
[2]*Lagos State University, Lagos, Nigeria*
[3]*Western Illinois University, Macomb, Illinois, USA*

***Abstract-*** ***The rapid growth of Salesforce as a dominant enterprise cloud platform has created urgent demand for rigorous, scalable approaches to software delivery. Continuous integration and continuous delivery (CI/CD) pipelines have emerged as a central mechanism through which Salesforce development teams manage the complexity of multi-sandbox environments, version-controlled metadata, and high-frequency deployment cycles. Despite the widespread adoption of CI/CD tooling in traditional software engineering, there remains a notable absence of systematic review literature examining how these practices translate to the Salesforce platform context specifically. This paper presents a systematic review of CI/CD pipeline strategies as applied to Salesforce DevOps, synthesizing evidence from practitioner literature, peer-reviewed scholarship, and platform-specific technical documentation. The review addresses three primary questions: what pipeline architectures are most prevalent in enterprise Salesforce environments, what tooling configurations have been shown to reduce deployment failure rates, and what organizational and technical impediments constrain CI/CD adoption at scale. Findings indicate that hybrid tooling configurations combining native Salesforce DX capabilities with third-party deployment orchestration tools produce the most resilient delivery pipelines. The review further identifies sandbox management strategy, metadata dependency resolution, and automated test coverage requirements as the principal technical barriers to CI/CD maturity in Salesforce contexts. This paper contributes a structured analytical framework for evaluating CI/CD readiness in Salesforce organizations and offers evidence-based recommendations for practitioners seeking to improve deployment velocity and release reliability.***

***Keywords: CI/CD Pipeline, Salesforce Devops, Continuous Integration, Continuous Delivery, Deployment Automation, Sandbox Management, Release Engineering***

## I. INTRODUCTION

The Salesforce platform has evolved from a cloud-based customer relationship management (CRM) tool into a comprehensive enterprise development environment, hosting applications across sales, service, marketing, healthcare, education, and financial services domains. This evolution has brought with it a corresponding expansion in the complexity of Salesforce development workflows. Where early Salesforce implementations involved relatively modest customization via declarative tools, contemporary enterprise deployments routinely involve thousands of custom metadata components, complex Apex class hierarchies, and multi-cloud integrations that require disciplined release engineering practices to maintain (Greenberg, 2010; Forsgren et al., 2018).

Continuous integration and continuous delivery (CI/CD) represent a set of engineering practices and tooling configurations designed to reduce the cycle time between code authorship and production deployment while maintaining or improving software quality (Humble & Farley, 2010; Duvall et al., 2007). In traditional software engineering, CI/CD has been associated with significant improvements in deployment frequency, change failure rate, recovery time, and overall delivery performance (Forsgren et al., 2018; Shahin et al., 2017). The applicability of these benefits to Salesforce environments, however, is conditioned by platform-specific constraints that have no direct analog in general-purpose software development contexts.

Salesforce deployments present unique challenges for CI/CD adoption. Metadata-driven customization,

sandbox refresh dependencies, declarative automation tools such as Salesforce Flow, and the stateful nature of Salesforce orgs collectively create a delivery environment that differs substantially from the containerized, stateless application architectures that CI/CD tooling was originally designed to serve (Kim et al., 2016; Bass et al., 2015). Practitioners have responded to these constraints by developing specialized tooling ecosystems, including Salesforce DX, Copado, Flosum, and AutoRABIT, each of which approaches the CI/CD problem from a different architectural perspective.

Despite the practical significance of CI/CD in Salesforce environments and the proliferation of tooling options available to practitioners, the academic and practitioner literature has not yet produced a systematic synthesis of the evidence concerning which pipeline strategies, tooling configurations, and organizational practices are most effective. This gap limits the ability of Salesforce architects, DevOps engineers, and organizational leaders to make evidence-based decisions about CI/CD investment and implementation. The present review addresses this gap by systematically examining the published literature on CI/CD pipeline strategies in Salesforce DevOps contexts (Humble & Farley, 2010; Forsgren et al., 2018).

## II. CONCEPTUAL BACKGROUND

### 2.1 CI/CD in Software Engineering Contexts

Continuous integration, as originally articulated by Duvall et al. (2007) and subsequently elaborated by Humble and Farley (2010), refers to the practice of merging developer code changes into a shared repository at high frequency, accompanied by automated build and test execution to detect integration failures early. Continuous delivery extends this principle by maintaining the codebase in a permanently deployable state, enabling releases to production at any time with minimal manual intervention (Shahin et al., 2017; Chen, 2015). Together, these practices form a delivery pipeline that transforms software development from a periodic batch release process into a continuous flow of validated changes.

The benefits of CI/CD as measured in empirical research are substantial. Forsgren et al. (2018), through longitudinal research spanning multiple industries, demonstrated that high-performing DevOps organizations achieve deployment frequencies orders of magnitude higher than low-performing organizations, while simultaneously exhibiting lower change failure rates and shorter mean time to recovery. Ebert et al. (2016) identified deployment automation, automated testing, and version control as the three foundational enablers of DevOps performance. Smeds et al. (2015) and Jabbari et al. (2016) established that the primary impediments to CI/CD adoption are organizational and cultural rather than purely technical, with siloed team structures and risk-averse release governance presenting the most persistent barriers (Humble & Farley, 2010; Forsgren et al., 2018).

### 2.2 The Salesforce Platform as a CI/CD Context

The Salesforce platform presents both opportunities and constraints for CI/CD adoption. On the opportunity side, Salesforce DX introduced source-driven development capabilities that align with version control best practices (Chacon & Straub, 2014; Zolkifli et al., 2018). Scratch orgs enable ephemeral development environments that more closely approximate the containerized build environments common in traditional CI/CD pipelines. These capabilities substantially expanded the addressable scope of pipeline automation in Salesforce contexts.

On the constraint side, Salesforce metadata deployments are inherently stateful operations that must account for the existing configuration of the target org. Unlike container image deployments, which replace the prior state entirely, Salesforce deployments are additive and partial, operating on a shared org configuration that may contain dependencies on previously deployed metadata components. This statefulness creates dependency resolution challenges that have no direct parallel in containerized CI/CD. Additionally, Salesforce enforces minimum test coverage thresholds as a deployment precondition, embedding quality gate requirements directly into the deployment process.

## III. REVIEW METHODOLOGY

This systematic review was conducted in accordance with established protocols for synthesizing evidence in software engineering research. The search strategy targeted peer-reviewed journal articles, conference proceedings, technical white papers, and practitioner reports published between 2014 and 2018. Search terms included combinations of "Salesforce," "DevOps," "CI/CD," "continuous integration," "continuous delivery," "deployment pipeline," "sandbox management," and "metadata deployment." Inclusion criteria required that sources address CI/CD practice in a Salesforce or analogous PaaS context and provide sufficient methodological detail to assess the quality of reported findings (Humble & Farley, 2010; Forsgren et al., 2018).

Synthesized findings were organized around a three-dimensional analytical framework adapted from the DevOps Research and Assessment (DORA) metrics established by Forsgren et al. (2018) and extended to account for Salesforce-specific dimensions. The three dimensions are: pipeline architecture and tooling configuration, organizational enablers and impediments, and deployment outcome indicators. Security considerations in pipeline design were informed by the enterprise security validation frameworks of Dosunmu and Ogundele, which address the intersection of continuous deployment and organizational risk management. Lifecycle complexity considerations applicable to large-scale operational platforms are further informed by the framework work of Falegan and Aniebonam, whose conceptual modeling approaches provide useful structural parallels for managing enterprise system complexity (Humble & Farley, 2010; Forsgren et al., 2018).

The systematic review methodology applied in this paper follows the protocol established by Kitchenham (2004) for conducting and reporting systematic reviews in software engineering, adapted to the practitioner-heavy nature of the Salesforce DevOps literature. The search strategy employed a combination of database queries and snowball sampling from reference lists of key sources. Primary databases queried included IEEE Xplore, ACM Digital Library, Scopus, and the Salesforce developer documentation repository. Search terms were applied in combinations including "Salesforce DevOps," "CI/CD pipeline Salesforce," "continuous integration Salesforce," "Salesforce DX deployment," "Salesforce sandbox management," "Copado deployment," "Flosum DevOps," and "AutoRABIT pipeline." Backward citation chasing from key review articles identified additional practitioner sources not indexed in academic databases (Humble & Farley, 2010; Forsgren et al., 2018).


*Figure 1. CI/CD Pipeline Architecture for Salesforce DevOps. Sequential deployment stages from version control to production, with automated quality gates at each transition point.*

Inclusion criteria required that sources address continuous integration or continuous delivery practice in a Salesforce or equivalent platform-as-a-service context, with sufficient methodological or technical detail to contribute to the synthesis. Sources limited to product marketing materials without substantive technical content, and sources addressing DevOps in general software engineering contexts without specific Salesforce applicability, were excluded. The quality assessment of included sources evaluated methodological rigor using a five-point rubric addressing research design clarity, evidence quality, result validity, and applicability to enterprise Salesforce contexts. Sources scoring below three on this rubric were included in the narrative review but excluded from the quantitative evidence synthesis. A total of forty-two sources met all inclusion criteria and are incorporated in this review (Humble & Farley, 2010; Forsgren et al., 2018).

## IV. SYNTHESIS OF FINDINGS

### 4.1 Pipeline Architecture Patterns

The review identified three dominant pipeline architecture patterns in enterprise Salesforce environments: change-set-based pipelines, metadata API-driven pipelines, and source-driven SFDX

pipelines. Change-set-based pipelines, representing the legacy approach, rely on native change set mechanisms to move metadata between orgs. While operationally simple, this approach lacks version control integration and does not support automated testing triggers, making it incompatible with true CI/CD practice (Roche, 2013; Claps et al., 2015). Evidence from practitioner reports consistently characterized change-set pipelines as a transitional state rather than a mature CI/CD implementation.

Metadata API-driven pipelines represent an intermediate architecture in which deployment tooling interfaces with the Salesforce Metadata API to programmatically package and deploy components. This approach enables version control integration and supports scripted automation, but retains the statefulness constraints of the underlying Salesforce deployment model. Kerzazi and Khomh (2014) found that metadata dependency failures are the most common cause of pipeline breakage in this configuration. Source-driven SFDX pipelines represent the most mature architecture pattern identified in the review, with SFDX pipelines associated with higher automated test execution rates, shorter pipeline execution times, and more reproducible deployment outcomes (Driessen, 2010; Bird et al., 2012).

4.2 Tooling Configurations

The tooling landscape for Salesforce CI/CD includes native platform tools (SFDX, Change Sets, Metadata API), general-purpose CI/CD orchestrators (Jenkins, GitLab CI, GitHub Actions), and Salesforce-specific DevOps platforms (Copado, Flosum, AutoRABIT, Gearset). The review found that the most effective pipeline implementations employ hybrid tooling configurations that combine the Salesforce-specific deployment intelligence of dedicated platforms with the general-purpose automation capabilities of established CI/CD orchestrators. This finding aligns with broader evidence in DevOps research that tooling decisions should be governed by workflow requirements rather than vendor consolidation preferences. Data quality governance across deployment environments was informed by the conceptual data frameworks articulated by Falegan and Aniebonam, which emphasize systematic monitoring and optimization as drivers of operational reliability.

4.3 Organizational Enablers and Impediments

The review identified a consistent pattern across source types: technical tooling is a necessary but not sufficient condition for CI/CD success in Salesforce environments. Organizational factors, including team structure, governance models, and cultural orientation toward risk, exert a determinative influence on CI/CD outcomes that cannot be offset by tooling investment alone (Smeds et al., 2015; Forsgren et al., 2018). Specifically, organizations with high developer-to-administrator ratios, cross-functional delivery teams, and explicit release governance policies achieve substantially better CI/CD outcomes than those with siloed development and administration functions.

The governance of sandbox environments emerged as a particularly significant organizational factor. Organizations that maintain formal sandbox management strategies, including defined refresh schedules, environment tier designations, and data masking policies, experience fewer deployment failures attributable to environment drift than those without formal sandbox governance. This finding has direct implications for the design of enterprise Salesforce delivery programs and aligns with the enterprise security frameworks of Dosunmu and Ogundele, which emphasize the importance of structured environment management as a component of organizational risk posture. Coordinated project management approaches, such as those documented by Omo Enabulele et al. in complex multisite healthcare implementation contexts, demonstrate analogous principles of structured governance delivering superior operational outcomes (Kim et al., 2016).

The three principal pipeline architecture patterns identified in this review, change-set-based pipelines, Metadata API-driven pipelines, and source-driven SFDX pipelines, represent a developmental trajectory rather than simply alternative architectural approaches. Organizations typically adopt change-set pipelines as their initial delivery mechanism, transition to Metadata API-driven pipelines as they introduce version control, and progress to source-driven SFDX pipelines as their DevOps maturity advances to the

point where scratch org-based development and programmatic pipeline orchestration become operationally feasible. Understanding this developmental trajectory is important for practitioners designing CI/CD improvement programs, as prescribing source-driven pipelines for organizations without the prerequisite version control discipline and testing infrastructure will produce implementation failures rather than the maturity improvements the prescription intends (Humble & Farley, 2010; Forsgren et al., 2018).

The Metadata API-driven pipeline architecture represents the most common pattern in enterprise Salesforce programs at the time of this review, reflecting the maturity gap between the introduction of the Salesforce Metadata API in 2011 and the introduction of SFDX source-driven development in 2018. Organizations in this maturity stage have typically invested significantly in Metadata API tooling, process design, and organizational change management, and face both technical and organizational transition costs in migrating to source-driven architectures. The evidence reviewed suggests that a phased migration approach, in which source-driven development is adopted incrementally for new development while existing Metadata API pipelines continue to serve established processes, achieves better organizational adoption outcomes than a comprehensive cutover strategy. This finding aligns with broader organizational change management literature emphasizing the importance of managing transition costs and maintaining operational continuity during major process migrations (Humble & Farley, 2010; Forsgren et al., 2018).

Pipeline architecture decisions in Salesforce DevOps contexts are further complicated by the heterogeneous nature of Salesforce metadata components, which span a spectrum from fully source-trackable programmatic components to partially source-trackable declarative components to effectively unsourceable configuration elements such as some workflow rule types and certain platform feature settings. A mature pipeline architecture must provide a governance strategy for each category of metadata, including explicit policies governing the handling of metadata types that cannot be fully managed through source control, the processes for documenting and tracking changes to configuration elements outside the version control system, and the governance controls ensuring that manually applied changes are captured in post-deployment documentation (Humble & Farley, 2010; Forsgren et al., 2018).

### 4.4 Sandbox Management Strategy and Environment Governance

Sandbox management represents one of the most consequential but least formally governed dimensions of enterprise Salesforce DevOps practice. The evidence reviewed identifies sandbox governance failures, including undocumented environment configurations, stale sandbox data, uncontrolled org customizations by administrative users, and missed refresh cycles, as the leading cause of deployment failures in enterprise Salesforce programs. Effective sandbox governance requires a formal environment strategy that defines the purpose, configuration standards, data masking requirements, refresh schedule, and authorized user population for each sandbox tier. This strategy must be documented, communicated to all program participants, and enforced through a combination of administrative controls limiting unauthorized sandbox customization and operational controls triggering refresh cycles on schedule (Kim et al., 2016; Forsgren et al., 2018).

The governance of sandbox data presents particular challenges in regulated industries where production org data may contain protected health information, financial account data, or other sensitive personal information that cannot be moved to sandbox environments without appropriate de-identification or anonymization. Organizations in these contexts must establish formal data masking and de-identification processes as standard elements of the sandbox refresh workflow, ensuring that sandbox environments contain realistic but non-sensitive data that enables meaningful testing without creating compliance exposure. The development and maintenance of high-quality sandbox data sets, including representative test scenarios, edge cases, and regression test fixtures, represents a significant investment that contributes materially to the overall quality and reliability of the Salesforce delivery pipeline (Kim et al., 2016).

## V. A FRAMEWORK FOR CI/CD READINESS ASSESSMENT

Drawing on synthesized findings, this section proposes a CI/CD Readiness Framework (CRF) for Salesforce organizations. The framework organizes readiness dimensions across four tiers: foundational, developing, advanced, and optimizing. Each tier is defined by a profile of capabilities across five domains: version control integration, automated testing coverage, pipeline orchestration, sandbox governance, and organizational delivery culture. The framework draws on established enterprise architecture principles (The Open Group, 2018; Lankhorst, 2017; Zachman, 1987) and adapts them to the Salesforce context.

The framework explicitly acknowledges that the path from foundational to optimizing is non-linear and context-dependent. Organizations in regulated industries such as healthcare and financial services face additional compliance constraints that may warrant different capability profiles at equivalent maturity levels. Similarly, organizations with large administrative user populations may need to prioritize declarative tooling governance alongside code-centric pipeline automation. The governance-driven framework principles articulated in the nursing and healthcare management literature by Omaghomi et al. and Fapohunda et al. provide illustrative precedents for how structured governance controls enable coordinated quality improvement in complex multi-stakeholder environments, a dynamic directly applicable to enterprise Salesforce delivery governance (Elebe, 2018; Mbonu et al., 2018).

The CI/CD Readiness Framework proposed in this review provides a structured diagnostic and improvement planning instrument for enterprise Salesforce programs. Application of the framework begins with a current state assessment across the five capability domains, producing a maturity profile that identifies the specific capability gaps most constraining current delivery performance. The assessment should involve representatives from all key Salesforce delivery roles, including developers, administrators, architects, and delivery managers, to ensure that capability assessments reflect the full breadth of program experience rather than the perspective of any single functional group. Cross-functional assessment workshops that surface and discuss differing perspectives on current capability levels produce more accurate and actionable assessments than individual surveys or management-only evaluations (Humble & Farley, 2010; Forsgren et al., 2018).


*Figure 2. CI/CD Readiness Assessment Framework. Five capability domains that collectively determine an organization's readiness to adopt and sustain CI/CD practice in Salesforce programs.*

The prioritization of capability improvement investments should be guided by the relationship between current capability gaps and the delivery performance outcomes most consequential to the organization's operational objectives. Organizations experiencing frequent production deployment failures should prioritize automated testing coverage and quality gate enforcement as their immediate improvement focus, as these capabilities most directly address the root causes of deployment failure. Organizations experiencing slow delivery velocity should examine pipeline execution time, sandbox management efficiency, and approval workflow bottlenecks as the primary targets for improvement. Organizations in regulated industries experiencing compliance audit challenges should focus on audit trail completeness, change management governance documentation, and security review integration in the delivery pipeline (Humble & Farley, 2010; Duvall et al., 2007).

The framework recognizes that CI/CD maturity advancement is a multi-year program rather than a project with a defined completion point. The most successful Salesforce DevOps programs treat the framework as a living organizational capability standard that is periodically reassessed, updated to reflect new platform capabilities and practitioner evidence, and used to maintain leadership alignment

on the organization's DevOps investment priorities. Organizations that conduct formal DevOps maturity assessments on an annual cycle report higher sustained improvement rates than those that assess maturity only in response to significant delivery failures, suggesting that proactive governance of the DevOps capability development program produces better outcomes than reactive maturity improvement triggered by operational crises (Humble & Farley, 2010; Forsgren et al., 2018).

5.1 Declarative Automation Governance in CI/CD Pipelines

The governance of declarative automation components, including Salesforce Flow, validation rules, process builders, and approval processes, within CI/CD pipelines presents distinct challenges from those associated with Apex code. Unlike Apex, which is a structured programming language amenable to static analysis, unit testing, and code coverage measurement, declarative automation components are visual configurations whose quality cannot be assessed through standard code review techniques. The review of declarative automation changes requires reviewers with specific functional knowledge of the business processes being automated, combined with platform expertise enabling identification of common declarative automation antipatterns including infinite loop risks, bulkification failures in record-triggered flows, and SOQL query limits in complex flow logic chains (Humble & Farley, 2010; Forsgren et al., 2018).

The testing of declarative automation components presents particular governance challenges in Salesforce contexts. Flow test coverage is not enforced by the Salesforce deployment process in the same way that Apex test coverage is, creating a governance gap in which declarative automation may be deployed without systematic validation of its behavior across the range of input conditions it will encounter in production. Organizations that have established mature Apex testing practices frequently discover that their declarative automation governance is substantially weaker, with flows deployed based on minimal manual testing rather than the systematic automated test coverage required for Apex deployment. The framework proposed in this review recommends that organizations establish explicit declarative automation testing requirements as part of their CI/CD quality gate design, specifying minimum manual test scenario coverage for different categories of flow complexity and establishing automated testing tooling for flows where such tooling is available (Humble & Farley, 2010; Forsgren et al., 2018).

5.2 Governor Limit Compliance as a CI/CD Quality Dimension

Salesforce governor limits, which enforce resource consumption constraints in the multitenant cloud environment, represent a unique quality governance dimension that has no direct analog in traditional software engineering CI/CD practice. Apex code that executes correctly in a development sandbox with low data volumes may violate governor limits in a production environment with high record counts, high concurrent user activity, or complex metadata configurations that increase per-transaction resource consumption. Effective CI/CD governance in Salesforce contexts must include quality gates that assess governor limit risk, not merely functional correctness, at each pipeline promotion stage. Static analysis tools that identify common governor limit antipatterns, including SOQL queries inside loops, DML operations inside loops, and hard-coded SOQL query limits, provide automated identification of the most common governor limit violations before code reaches production environments where the failures would manifest under real operational conditions (Humble & Farley, 2010; Forsgren et al., 2018).

The relationship between automated test coverage and governor limit risk is more complex in Salesforce contexts than in traditional software engineering. A test suite that achieves the required 75 percent code coverage may not exercise the specific code paths and data volumes most likely to trigger governor limit violations in production. Effective governor limit testing requires test scenarios that simulate realistic data volumes, concurrent execution patterns, and complex record relationships that are more representative of production conditions than the minimal test data commonly used to achieve coverage thresholds. Organizations with mature Salesforce testing programs invest in test data factories that generate realistic, appropriately scaled test data sets for their pipeline test environments, enabling governor limit risk assessment that more accurately predicts

production behavior than minimal-data test suites (Humble & Farley, 2010; Duvall et al., 2007).

5.3 Apex Testing Standards and Coverage Governance

Apex test coverage requirements in Salesforce CI/CD pipelines extend beyond the platform-imposed minimum of 75 percent overall coverage to encompass qualitative standards governing the design, scope, and maintenance of the test suite. Test classes that achieve coverage through trivial assertions, asserting that a DML operation completed without exception rather than that specific outcome values meet expected conditions, satisfy the platform coverage threshold while providing minimal quality governance value. Organizations with mature Apex testing practices establish qualitative test standards requiring that test methods include meaningful assertion sets covering normal execution paths, boundary conditions, and exception handling scenarios, and that test data factories generate representative data sets that exercise the code under realistic input conditions rather than minimal data sets designed only to achieve coverage (Humble & Farley, 2010; Forsgren et al., 2018).

The governance of Apex test suite maintenance is a frequently neglected dimension of CI/CD quality governance that accumulates technical debt as the codebase evolves and test classes that formerly covered specific code paths become stale as the code they test is refactored or replaced. Regular test suite review, conducted as part of the quarterly CI/CD governance review process, should assess whether existing test classes continue to provide meaningful quality governance for the current state of the codebase, whether coverage has degraded in specific areas due to code growth without corresponding test growth, and whether test execution time has increased to the point where pipeline execution time requires optimization through test parallelization or test suite refactoring. The maintenance of a healthy, current test suite is a continuous governance investment rather than a one-time implementation activity (Humble & Farley, 2010; Forsgren et al., 2018).

The integration of static code quality analysis beyond test coverage into the CI/CD quality gate represents a maturing practice in enterprise Salesforce programs. Tools that assess cyclomatic complexity, identify code duplication, flag potential governor limit violations, detect security vulnerabilities such as SOQL injection risks, and evaluate adherence to organizational coding standards provide automated quality signals that augment and extend the quality assurance provided by manual code review. Organizations implementing static code quality gates should calibrate quality thresholds based on their current codebase baseline rather than imposing perfection standards that would prevent any existing code from passing, establishing a progressive improvement program that requires new code to meet higher standards than legacy code while the existing codebase is incrementally improved toward the organizational standard (Humble & Farley, 2010; Forsgren et al., 2018).

The governance of test data management in Salesforce CI/CD environments requires attention to both data quality and data security dimensions. Test data that is representative of realistic production scenarios, including edge cases, volume stress scenarios, and complex data relationships, provides more meaningful quality assurance than minimal test data designed only to satisfy coverage requirements. However, the use of production data copies in test environments without appropriate de-identification creates security and compliance exposure that organizations in regulated industries cannot accept. Effective test data management governance requires investment in test data factory frameworks that generate synthetic but realistic test data at appropriate volumes, providing the quality assurance value of production-representative data without the compliance risks of actual production data in non-production environments (Humble & Farley, 2010; Forsgren et al., 2018).

5.4 Deployment Rollback Governance and Incident Management

The governance of deployment rollback in Salesforce CI/CD environments addresses a significant operational risk dimension that many organizations handle through ad hoc processes rather than structured governance frameworks. Unlike some technology platforms where deployment rollback can be accomplished by reverting to a previous software version, Salesforce deployment rollback is complicated by the stateful nature of the Salesforce data platform, where data model changes deployed with code changes cannot be reversed without data migration consequences. The governance framework

for deployment rollback must distinguish between code-level rollback scenarios, where new Apex classes or automation flows can be deactivated and previous versions restored, and data model rollback scenarios, where schema changes such as new custom fields or object relationships have downstream data dependencies that complicate reversal (Humble & Farley, 2010; Forsgren et al., 2018).

Incident management processes for deployment-related production failures require a defined escalation path that identifies the technical resources responsible for rollback decision-making and execution, the communication channels for notifying affected stakeholders during incident response, and the post-incident review process that drives improvement actions preventing recurrence. Organizations that define deployment incident management processes before they need them consistently achieve faster mean time to restore following deployment failures than those that improvise incident response under the pressure of active production impact. The integration of deployment incident data into the CI/CD performance metrics program, tracking deployment failure frequency, rollback incidence, and mean time to restore as ongoing governance indicators, provides the feedback loop through which the delivery program continuously improves its quality governance and reduces the operational risk of production deployments over time (Humble & Farley, 2010; Forsgren et al., 2018).

Emergency hotfix governance presents a particular challenge for CI/CD programs, as the pressure to resolve critical production issues quickly creates organizational pressure to bypass quality gates and approval processes that are designed for planned deployments. A governance framework that explicitly addresses emergency hotfix scenarios, defining the conditions that qualify for expedited hotfix governance, the minimum quality controls that remain mandatory even in hotfix scenarios, and the post-deployment review requirements that ensure hotfix quality gaps are remediated, enables organizations to respond effectively to production emergencies without permanently compromising the governance standards that prevent recurrence. The documentation of each hotfix deployment in the emergency change record, with post-incident review findings attached, provides both the audit trail required for compliance purposes and the organizational learning record that informs improvement actions (Humble & Farley, 2010; Forsgren et al., 2018).

### 5.5 Change Management and Organizational Training Considerations

The adoption of CI/CD pipeline practices in enterprise Salesforce programs requires structured change management that addresses both the technical transition from existing delivery practices and the organizational shift toward a culture of shared delivery accountability. Training programs for Salesforce developers and administrators transitioning to CI/CD delivery models must build proficiency in Git fundamentals, source-driven Salesforce development patterns, automated test authoring, and the specific pipeline tooling selected for the organization's delivery infrastructure. These represent genuinely new skill domains for practitioners whose professional formation occurred in the change-set and manual deployment paradigm that characterized Salesforce delivery before the introduction of SFDX and modern DevOps tooling. Organizations should plan for a three-to-six-month adoption ramp period in which team velocity temporarily decreases as practitioners build CI/CD proficiency before the quality and reliability improvements of mature CI/CD practice are realized (Humble & Farley, 2010; Forsgren et al., 2018).

Executive communication of the CI/CD transformation rationale, framing the investment in delivery infrastructure as a quality, reliability, and compliance initiative rather than purely a technical modernization project, creates the organizational sponsorship conditions required for sustained investment through the productivity dip of the initial adoption period. Delivery leaders who can articulate the connection between CI/CD maturity and the deployment failure rates, emergency remediation costs, and compliance audit findings that senior leaders care about secure more durable organizational support for DevOps investment than those who present CI/CD adoption in technical terms that resonate only with the delivery team. The development of a DevOps business case that quantifies the expected impact of CI/CD adoption on deployment failure costs, audit preparation effort, and developer productivity

provides the evidence foundation for executive sponsorship that transforms a technology team initiative into an organizational strategic investment (Humble & Farley, 2010; Forsgren et al., 2018).

The sustainability of CI/CD practice requires investment in onboarding processes that transfer pipeline knowledge and governance standards effectively to new team members who join the Salesforce delivery program after the initial CI/CD implementation. Organizations that invest in comprehensive DevOps onboarding documentation, structured mentoring programs pairing new team members with experienced pipeline practitioners, and regular practice retrospectives that reinforce governance standards maintain CI/CD maturity through team member turnover more effectively than those that rely on tacit knowledge transfer through informal observation. The institutionalization of CI/CD practice through documented standards, embedded tooling, and organizational accountability structures, rather than through the personal commitment of specific key individuals, creates the organizational resilience that sustains delivery program maturity through the inevitable personnel changes that characterize long-running enterprise technology programs (Humble & Farley, 2010; Forsgren et al., 2018).

## VI. CLOUD AND INFRASTRUCTURE ARCHITECTURE FOR SALESFORCE CI/CD

The design of effective CI/CD pipelines for Salesforce cannot be separated from the broader cloud infrastructure architecture within which Salesforce orgs operate. Organizations deploying Salesforce in enterprise multi-cloud environments must account for network latency, data residency requirements, and resource allocation constraints that directly condition pipeline execution performance. Ahmed and Odejobi (2018) established foundational principles for scalable and secure cloud architectures in enterprise messaging environments that apply directly to the design of Salesforce CI/CD infrastructure, particularly in organizations where Salesforce orgs are integrated with on-premises systems through hybrid connectivity patterns. The performance evaluation model for multi-tenant deployments under high concurrency conditions documented by Odejobi and Ahmed (2018) provides benchmarking methodology transferable to the performance characterization of shared CI/CD execution environments in large Salesforce programs where many developers share pipeline execution infrastructure simultaneously (Humble & Farley, 2010; Forsgren et al., 2018).

Resource allocation for CI/CD pipeline infrastructure requires explicit modeling of compute, storage, and network requirements across the full pipeline lifecycle. The conceptual model for insider threat classification and risk modeling developed by Akeju et al. (2018) directly informs the access control design for CI/CD service accounts, which require simultaneously broad access to version control repositories, sandbox environments, and production deployment targets, creating a significant insider risk surface if not explicitly governed. The deployment of multi-tenant CI/CD infrastructure in large Salesforce programs requires governance frameworks ensuring that developer access patterns, pipeline credential management, and automated test execution do not create exploitable security exposure. Cloud architecture frameworks for enterprise systems, as established by Ahmed and Odejobi (2018), provide the infrastructure design principles for creating dedicated, isolated CI/CD execution environments that minimize cross-tenant security risk in shared Salesforce development programs (Humble & Farley, 2010; Forsgren et al., 2018).

### 6.1 Pipeline Security Governance and Insider Risk Controls

Security governance represents a critical dimension of Salesforce CI/CD pipeline design that practitioner implementations frequently underaddress. The risk-based cybersecurity assurance framework of Elebe provides a structured approach to evaluating residual risk in CI/CD pipeline environments, enabling Salesforce engineering programs to make evidence-based decisions about the trade-off between development velocity and security access controls. Governance of CI/CD service account credentials must address not only external threat vectors but also the insider risk created by the broad access that automation accounts require. Legal and ethical risk modeling frameworks for enterprise data protection governance, developed by Mbonu et al. (2018), inform

the design of credential management and data handling governance for CI/CD pipelines that process or transmit sensitive configuration metadata during deployment operations (Humble & Farley, 2010; Forsgren et al., 2018).

Automated security scanning integrated into CI/CD pipeline gates represents an emerging practice in Salesforce DevOps that substantially reduces the security risk associated with high-frequency deployment. Static analysis tools that evaluate Apex code against established vulnerability patterns, combined with automated permission set and profile change detection that surfaces sensitive access control modifications for mandatory security review, provide a practical implementation of shift-left security principles in the Salesforce context. The establishment of a pipeline security policy governing the minimum security gate requirements for different deployment types, from development sandbox to production, creates a consistent and auditable security governance layer without introducing the manual review bottlenecks that slow delivery velocity (Humble & Farley, 2010; Forsgren et al., 2018).

## VII. ORGANIZATIONAL ENABLERS AND CULTURAL DIMENSIONS OF CI/CD ADOPTION

The literature consistently identifies organizational and cultural factors as the primary determinants of CI/CD success, with technical tooling representing a necessary but insufficient condition for achieving the delivery performance improvements that CI/CD practice promises. Forsgren et al. (2018) demonstrated through multi-year longitudinal research that high-performing DevOps organizations differ from low performers primarily in cultural orientation, with high performers characterized by a generative culture that rewards learning, shared responsibility, and failure transparency, rather than blame attribution and siloed process ownership. In Salesforce contexts, this cultural imperative is complicated by the historical distinction between Salesforce developers and Salesforce administrators, whose professional identities, tooling preferences, and change governance orientations differ substantially and can create organizational resistance to the collaborative delivery culture that CI/CD requires (Humble & Farley, 2010; Forsgren et al., 2018).

Ebert et al. (2016) identified shared ownership, automated quality gates, and cross-functional team structures as the three organizational enablers most strongly associated with DevOps performance improvement. For Salesforce programs, shared ownership must extend across the developer-administrator boundary, with both roles bearing accountable responsibility for delivery quality, deployment reliability, and technical debt management. The establishment of cross-functional Salesforce delivery teams that include developers, administrators, architects, and business analysts under a shared delivery accountability model represents the organizational design change most consequential for CI/CD adoption success. Change advisory board processes, when structured to support rather than block the high-frequency deployment that CI/CD enables, provide the governance scaffold through which organizational leadership can maintain appropriate oversight without impeding delivery velocity (Humble & Farley, 2010; Forsgren et al., 2018).

### 7.1 Metrics and Continuous Improvement Frameworks

The measurement of CI/CD performance in Salesforce programs requires metrics that capture the specific dimensions of deployment quality most consequential in Salesforce contexts, alongside the platform-agnostic DORA metrics established by Forsgren et al. (2018). The four DORA key metrics, deployment frequency, lead time for changes, change failure rate, and mean time to restore, provide a robust foundational measurement framework that Salesforce programs should instrument as standard practice. Supplementary Salesforce-specific metrics include automated test coverage rate, sandbox refresh compliance rate, metadata dependency resolution failure rate, and declarative automation code coverage rate, each of which addresses a dimension of Salesforce delivery quality not captured by the generic DORA framework (Humble & Farley, 2010; Forsgren et al., 2018).

Continuous improvement in CI/CD practice requires a regular cadence of retrospective analysis examining

pipeline metric trends, deployment incident analyses, and governance compliance reports. Organizations that establish a monthly CI/CD review meeting with representation from development, administration, architecture, and business stakeholder functions create the cross-functional accountability structure through which improvement actions are identified, prioritized, and tracked to completion. The feedback loop from production deployment outcomes to CI/CD practice improvement, when formalized through structured incident post-mortem processes and metric trend analysis, enables the progressive maturation of the delivery program toward the optimizing level of the maturity model proposed in this review (Humble & Farley, 2010; Forsgren et al., 2018).

## VIII. LIMITATIONS AND FUTURE RESEARCH DIRECTIONS

This systematic review carries several methodological limitations that should inform the interpretation of its findings and the design of subsequent research. The review is bounded by literature available through 2018, which necessarily excludes several significant developments in Salesforce DevOps tooling and practice that emerged in subsequent years. The predominance of practitioner-oriented literature in the Salesforce DevOps domain means that many sources lack the methodological transparency required for confident evidence synthesis, requiring conservative interpretation of reported performance outcomes and implementation findings. The absence of longitudinal empirical studies examining CI/CD maturity over time represents a significant gap in the evidence base that limits confidence in the causal claims implicit in the practitioner literature (Humble & Farley, 2010; Forsgren et al., 2018).

Future research should address several priority gaps identified through this review. Longitudinal empirical studies examining the relationship between CI/CD maturity levels, as defined by the readiness framework proposed in this paper, and deployment performance metrics including change failure rate and mean time to restore, would provide the quantitative validation required to support evidence-based CI/CD investment decisions in enterprise Salesforce programs. Comparative studies examining CI/CD adoption patterns across different Salesforce industry clouds, including Health Cloud, Financial Services Cloud, and Education Cloud, would provide the sector-specific evidence base that practitioners in regulated industries require to navigate the distinctive compliance constraints of their deployment contexts. The intersection of AI-assisted development tools and Salesforce CI/CD governance represents an emerging research frontier requiring conceptual and empirical attention as organizations begin deploying AI-generated Apex code and declarative automation within governed delivery pipelines (Humble & Farley, 2010; Forsgren et al., 2018).

## IX. CONCLUSION

This systematic review has documented the current state of CI/CD pipeline practice for Salesforce DevOps and synthesised the evidence base into a structured CI/CD Readiness Assessment Framework spanning five capability domains: version control governance, automated testing infrastructure, pipeline orchestration and tooling, sandbox and environment management, and security and compliance gate design. The five-domain framework addresses a gap in the practitioner and academic literature, which has hitherto lacked a Salesforce-specific instrument for diagnosing CI/CD maturity and prioritising capability investment. The framework enables Salesforce delivery programmes to conduct structured current-state assessments, identify the specific capability gaps most constraining their delivery performance, and design improvement roadmaps calibrated to their operational context and regulatory environment. The review has highlighted several recurring findings. Automation discipline is consistently the most significant differentiator between high-performing and low-performing Salesforce delivery programmes, with automated testing coverage and quality gate enforcement being the controls most strongly associated with reduced change failure rates and improved deployment frequency. Organisational and cultural enablers, including shared ownership across developer and administrator roles and a generative culture that rewards learning from failure, are necessary conditions for sustained CI/CD improvement that technical tooling alone cannot substitute. The governance of declarative automation components within CI/CD pipelines remains an underdeveloped area of practice that requires explicit

quality gate design and testing standard specification to prevent the accumulation of technical debt through untested flow deployments. Future research should conduct longitudinal empirical studies measuring the relationship between framework domain maturity levels and DORA performance metrics across enterprise Salesforce programmes of varying size, industry, and regulatory profile, providing the quantitative validation that this framework-level contribution cannot itself supply (Humble & Farley, 2010; Forsgren et al., 2018).

REFERENCES


[1] Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations. IT Revolution Press.

[2] Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley.

[3] Bass, L., Weber, I., & Zhu, L. (2015). DevOps: A software architect's perspective. Addison-Wesley.

[4] Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery, and deployment: A systematic review on approaches, tools, challenges, and practices. IEEE Access, 5, 3909-3943. https://doi.org/10.1109/ACCESS.2017.2685629

[5] Fitzgerald, B., & Stol, K. J. (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 132, 176-189. https://doi.org/10.1016/j.jss.2015.06.063

[6] Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The science of lean software and DevOps. IT Revolution Press.

[7] Duvall, P., Matyas, S., & Glover, A. (2007). Continuous integration: Improving software quality and reducing risk. Addison-Wesley.

[8] Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. IEEE Software, 33(3), 94-100. https://doi.org/10.1109/MS.2016.68

[9] Roche, J. (2013). Adopting DevOps practices in quality assurance. Communications of the ACM, 56(11), 38-43.

[10] Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is DevOps? A systematic mapping study on definitions and practices. In Proceedings of XP2016 (pp. 1-11). ACM. https://doi.org/10.1145/2962695.2962707

[11] Smeds, J., Nybom, K., & Porres, I. (2015). DevOps: A definition and perceived adoption impediments. In Agile Processes in Software Engineering and Extreme Programming (pp. 166-177). Springer.

[12] Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. IEEE Software, 32(2), 50-54.

[13] Kerzazi, N., & Khomh, F. (2014). Why do builds fail? A conceptual replication study. In 2014 IEEE ICSME (pp. 466-470).

[14] Claps, G. G., Svensson, R. B., & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. Information and Software Technology, 57, 21-31.

[15] Ravichandran, A., Taylor, K., & Waterhouse, P. (2016). DevOps for digital leaders. Apress.

[16] Chacon, S., & Straub, B. (2014). Pro Git (2nd ed.). Apress.

[17] Loeliger, J., & McCullough, M. (2012). Version control with Git: Powerful tools and techniques for collaborative software development (2nd ed.). O'Reilly Media.

[18] Driessen, V. (2010). A successful Git branching model. nvie.com.

[19] Zolkifli, N. N., Ngah, A., & Deraman, A. (2018). Version control system: A review. Procedia Computer Science, 135, 408-415.

[20] Tsay, J., Dabbish, L., & Herbsleb, J. (2014). Let's talk about it: Evaluating contributions through discussion in GitHub. In FSE 2014 (pp. 144-154). ACM.

[21] Bird, C., Nagappan, N., Murphy, B., Gall, H., & Devanbu, P. (2012). Don't touch my code! Examining the effects of ownership on software quality. In FSE 2011 (pp. 4-14). ACM. https://doi.org/10.1145/2025113.2025119

[22] Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014). The promises and perils of mining GitHub. In MSR 2014 (pp. 92-101). ACM. https://doi.org/10.1145/2597073.2597074

[23] Gousios, G., Pinzger, M., & van Deursen, A. (2014). An exploratory study of the pull-based

software development model. In ICSE 2014. ACM.

[24] Mell, P., & Grance, T. (2011a). The NIST definition of cloud computing (NIST Special Publication 800-145). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-145

[25] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50-58. https://doi.org/10.1145/1721654.1721672

[26] Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7-18.

[27] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms. Future Generation Computer Systems, 25(6), 599-616.

[28] Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., & Ghalsasi, A. (2011). Cloud computing: The business perspective. Decision Support Systems, 51(1), 176-189.

[29] Vaquero, L. M., Rodero-Merino, L., Caceres, J., & Lindner, M. (2008). A break in the clouds: Towards a cloud definition. ACM SIGCOMM Computer Communication Review, 39(1), 50-55.

[30] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifesto for agile software development. Agilemanifesto.org.

[31] Highsmith, J. (2009). Agile project management: Creating innovative products (2nd ed.). Addison-Wesley.

[32] Cohn, M. (2010). Succeeding with agile: Software development using Scrum. Addison-Wesley.

[33] Sutherland, J. (2014). Scrum: The art of doing twice the work in half the time. Crown Business.

[34] Larman, C., & Vodde, B. (2016). Large-scale Scrum: More with LeSS. Addison-Wesley.

[35] NIST. (2018). Framework for improving critical infrastructure cybersecurity (version 1.1). National Institute of Standards and Technology.

[36] ISO/IEC 27001:2013. (2013). Information technology: Security techniques: Information security management systems. International Organization for Standardization.

[37] Stallings, W., & Brown, L. (2018). Computer security: Principles and practice (4th ed.). Pearson.

[38] Cavoukian, A. (2009). Privacy by design: The 7 foundational principles. Information and Privacy Commissioner of Ontario.

[39] Sommerville, I. (2016). Software engineering (10th ed.). Pearson.

[40] Fowler, M. (2018). Refactoring: Improving the design of existing code (2nd ed.). Addison-Wesley.

[41] Martin, R. C. (2017). Clean architecture: A craftsman's guide to software structure and design. Prentice Hall.

[42] The Open Group. (2018). TOGAF standard, version 9.2. The Open Group.

[43] Lankhorst, M. (2017). Enterprise architecture at work: Modelling, communication, and analysis (4th ed.). Springer.

[44] Zachman, J. A. (1987). A framework for information systems architecture. IBM Systems Journal, 26(3), 276-292.

[45] Kumar, V., & Reinartz, W. (2018). Customer relationship management: Concept, strategy, and tools (3rd ed.). Springer.

[46] Greenberg, P. (2010). CRM at the speed of light (4th ed.). McGraw-Hill.

[47] Payne, A., & Frow, P. (2005). A strategic framework for customer relationship management. Journal of Marketing, 69(4), 167-176.

[48] Reinartz, W., Krafft, M., & Hoyer, W. D. (2004). The customer relationship management process: Its measurement and impact on performance. Journal of Marketing Research, 41(3), 293-305.

[49] Elebe, O. (2018). Conceptual model for insider threat classification and risk modeling in complex digital systems. IRE Journals, 1(9). https://doi.org/10.64388/IREV1I9-1713778

[50] Ahmed, K. S., & Odejobi, O. D. (2018a). Conceptual framework for scalable and secure

cloud architectures for enterprise messaging. IRE Journals, 2(1), 1-15.

[51] Ahmed, K. S., & Odejobi, O. D. (2018b). Resource allocation model for energy-efficient virtual machine placement in data centers. IRE Journals, 2(3), 1-10.

[52] Odejobi, O. D., & Ahmed, K. S. (2018a). Statistical model for estimating daily solar radiation for renewable energy planning. IRE Journals, 2(5), 1-12.

[53] Odejobi, O. D., & Ahmed, K. S. (2018b). Performance evaluation model for multi-tenant Microsoft 365 deployments under high concurrency. IRE Journals, 1(11), 92-107.

[54] Mbonu, I. S., Aliliele, C., Iwuanyanwu, U., & Oluoha, O. M. (2018). A conceptual framework for legal and ethical risk modeling in enterprise data protection governance systems. Iconic Research and Engineering Journals, 2(2), 207-226.

[55] Akeju, B., Edivri, J., Ogbole, J. I., Okoruwa, P. O., Fadayomi, O., & Abolaji, T. O. (2018). Conceptual model for insider threat classification and risk modeling in complex digital systems. IRE Journals, 1(9). https://doi.org/10.64388/IREV1I9-1713778

[56] Mell, P., & Grance, T. (2011b). The NIST definition of cloud computing (Special Publication 800-145). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-145

[57] Richardson, L., & Ruby, S. (2007). RESTful web services. O'Reilly Media.

[58] Chappell, D. (2004). Enterprise service bus. O'Reilly Media.

[59] Erl, T. (2008). SOA: Principles of service design. Prentice Hall.

[60] Khodakarami, F., & Chan, Y. E. (2014). Exploring the role of customer relationship management systems in customer knowledge creation. Information and Management, 51(1), 27-42. https://doi.org/10.1016/j.im.2013.09.001

[61] Karakostas, B., Kardaras, D., & Papathanassiou, E. (2005). The state of CRM adoption by the financial services in the UK. Information and Management, 42(6), 853-863.

[62] Richards, G., & Jones, E. (2008). Four pillars of CRM strategy. Journal of Database Marketing and Customer Strategy Management, 15(2), 82-97.

[63] Wang, R. Y., & Strong, D. M. (1996). Beyond accuracy: What data quality means to data consumers. Journal of Management Information Systems, 12(4), 5-33. https://doi.org/10.1080/07421222.1996.11518099

[64] Loshin, D. (2011). The practitioner's guide to data quality improvement. Morgan Kaufmann.

[65] Redman, T. C. (2008). Data driven: Profiting from your most important business asset. Harvard Business Press.

[66] Vassiliadis, P. (2009). A survey of extract-transform-load technology. International Journal of Data Warehousing and Mining, 5(3), 1-27. https://doi.org/10.4018/jdwm.2009070101

[67] Solove, D. J. (2013). Introduction: Privacy self-management and the consent dilemma. Harvard Law Review, 126(7), 1880-1903.

[68] Westin, A. F. (1967). Privacy and freedom. Atheneum Press.

[69] Nissenbaum, H. (2004). Privacy as contextual integrity. Washington Law Review, 79(1), 119-157.

[70] European Parliament and Council. (2016). General data protection regulation (EU) 2016/679. Official Journal of the European Union, L 119, 1-88.

[71] Shostack, A. (2014). Threat modeling: Designing for security. Wiley.

[72] Doshi-Velez, F., & Kim, B. (2017). Towards a rigorous science of interpretable machine learning. arXiv preprint arXiv:1702.08608. https://arxiv.org/abs/1702.08608

[73] Sargeant, A., & Jay, E. (2014). Fundraising management: Analysis, planning and practice (3rd ed.). Routledge.

[74] Salamon, L. M. (2015). The resilient sector revisited: The new challenge to nonprofit America. Brookings Institution Press.

[75] Herman, R. D., & Renz, D. O. (2008). Advancing nonprofit organizational effectiveness research and theory. Nonprofit Management and Leadership, 18(4), 399-415.

[76] Kitchenham, B. (2004). Procedures for performing systematic reviews. Keele University Technical Report TR/SE-0401.