

# Runbook Engineering and SOP Design in High-Availability Environments: A Playbook for DevOps Teams

Harish Govinda Gowda  
Devops Engineer, Promates Technologies LLC

**Abstract-** In modern high-availability environments, runbooks and Standard Operating Procedures (SOPs) serve as foundational tools for maintaining system reliability, enabling rapid incident response, and ensuring compliance. As organizations scale their DevOps and Site Reliability Engineering (SRE) practices, the need for structured, version-controlled, and automation-ready documentation becomes increasingly urgent. This article explores the principles and practices of runbook engineering and SOP design, offering a practical playbook for DevOps teams operating in complex, cloud-native infrastructures. Through real-world case studies and forward-looking strategies, it highlights how well-designed documentation not only reduces mean time to resolution (MTTR) but also empowers teams to automate responses, facilitate onboarding, and meet regulatory requirements. With insights into intelligent triggers, governance models, and AI-driven operational tooling, this guide aims to elevate runbooks and SOPs from static artifacts to dynamic, self-healing components of platform resilience.

**Index Terms-** Runbook engineering, SOP design, DevOps, Site Reliability Engineering, MTTR reduction.

## I. INTRODUCTION

In high-availability, modern digital environments, systems must be designed not only to perform well but also to recover swiftly from disruptions. As organizations increasingly embrace DevOps and Site Reliability Engineering (SRE) practices, the role of runbook engineering and Standard Operating Procedure (SOP) design becomes critical to maintaining operational excellence.

Runbooks serve as structured guides to address specific incidents, while SOPs define routine workflows and compliance-bound processes. Together, they form the backbone of system reliability, especially when operating at scale or under pressure.

The demand for resilient, fast-responding systems has reshaped how DevOps and SRE teams manage infrastructure, deploy changes, and handle incidents. Traditional IT models, reliant on tribal knowledge and ad hoc response, are no longer viable. Instead, engineering teams must codify operational knowledge into actionable, reusable formats that enable consistent, predictable results. High availability (HA) environments, in particular, require precision, speed, and traceability—goals that well-designed runbooks and SOPs help achieve.

As organizations adopt container orchestration, hybrid cloud, and service mesh architectures, the scope and complexity of operational tasks have grown. Failures may cascade across multiple systems and geographies, making clear documentation a necessity for coordinated incident response. Furthermore, compliance standards such as SOC 2, ISO 27001, and HIPAA require documented processes for routine operations and incident handling. Runbooks and SOPs not only support uptime—they also reinforce auditability and security.

This article explores the evolving discipline of runbook engineering and SOP design through the lens of high-availability systems. It delves into principles, practices, and automation strategies that modern DevOps teams use to turn documentation into real-time operational tools. From dynamic execution to governance, this guide presents a playbook for teams aiming to scale reliability while reducing human error, onboarding friction, and MTTR (mean time to resolution).

By embedding structured documentation into day-to-day operations, teams can transform chaos into consistency, and reactive firefighting into proactive resilience. Whether you're designing your first runbook or evolving an enterprise-grade SOP library, the insights ahead will help you build systems that are not only functional—but fault-tolerant, compliant, and future-ready.

## II. DEFINING RUNBOOKS AND SOPS IN DEVOPS

In DevOps and SRE practices, clear definitions of runbooks and Standard Operating Procedures (SOPs) are essential for operational clarity and consistency. Though the terms are sometimes used interchangeably, they serve distinct but complementary roles in the reliability ecosystem.

Runbooks are tactical documents that provide detailed, step-by-step instructions for resolving specific incidents or performing immediate operational tasks. They are usually designed for rapid execution by on-call engineers during outages, degraded performance, or alerts. A typical runbook might outline how to restart a failing Kubernetes pod, failover a database instance, or remediate a CPU spike. The emphasis is on speed, repeatability, and accuracy—especially under pressure.

SOPs, on the other hand, are broader in scope. They codify routine or compliance-driven processes that must be followed consistently over time, such as patch management, change control, or access reviews. SOPs often span multiple systems and teams, include decision points, and are used not just for emergency response but also for proactive operations. While a runbook might be followed in minutes during an incident, an SOP might be followed over hours or days as part of a scheduled process.

Both types of documents are integral to operational readiness, and both must be maintained rigorously. In high-availability environments, where downtime can have serious financial or reputational consequences, the accuracy of these documents is mission-critical. Errors, omissions, or outdated steps can amplify the impact of an incident, whereas a well-maintained runbook or SOP can contain it.

Documentation format is also a key consideration. Runbooks are increasingly written in structured formats such as Markdown, YAML, or JSON, making them both human-readable and automation-ready. SOPs, meanwhile, often live in platforms like Confluence, ServiceNow, or Git repositories with version control and workflow approval mechanisms. Some organizations adopt hybrid formats, embedding runbooks within SOPs or linking them contextually.

Lastly, both runbooks and SOPs benefit from ownership and continuous improvement. Assigning maintainers, incorporating feedback from postmortems, and reviewing documents as part of release or sprint cycles ensures they evolve alongside the systems they describe. In a world where services are distributed, dynamic, and tightly regulated, effective documentation is not just a best practice—it is a cornerstone of resilience.

## III. DEFINING RUNBOOKS AND SOPS: PURPOSE AND DIFFERENCES

In high-availability (HA) environments, where uptime and consistency are critical, having clearly defined operational documentation is essential. Two foundational types of such documentation are runbooks and Standard Operating Procedures (SOPs). While the terms are sometimes used interchangeably, they serve distinct purposes in a well-structured DevOps environment and understanding the difference can significantly enhance team efficiency, incident response, and service reliability.

A runbook is a detailed, task-focused document used by engineers to execute specific technical operations—particularly in response to alerts or incidents. These include step-by-step instructions for activities like restarting a service, scaling an application, handling a known failure scenario, or validating the state of a deployment. Runbooks are action-oriented, often tightly scoped, and intended for use during active troubleshooting or operational events. Their primary function is to provide a repeatable, clear pathway for resolving known issues or executing predefined workflows, usually with minimal context switching.

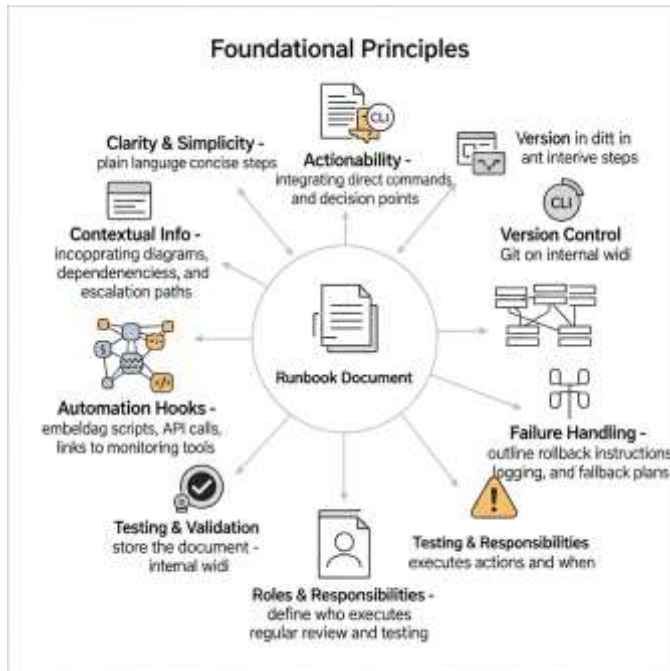
On the other hand, an SOP (Standard Operating Procedure) is a broader, more formalized document that outlines standard, recurring operations or compliance-related tasks in a systematic way. SOPs are often tied to business rules, audit requirements, or governance needs, covering things like change management approvals, patching policies, backup validation processes, or environment provisioning guidelines. They may not be invoked during a high-severity incident, but they govern how operational processes are performed to ensure consistency, quality, and accountability across teams and time zones.

While runbooks are typically more technical and immediate, SOPs serve a strategic role in defining how operations are carried out in a compliant, secure, and repeatable fashion. SOPs may reference one or more runbooks as part of their workflows, creating a layered structure of documentation that allows both flexibility and control.

It's also important to consider the audience. Runbooks are primarily used by on-call engineers, SREs, or operations staff responding to real-time conditions. SOPs, by contrast, are also read by compliance teams, auditors, or business stakeholders who need visibility into operational maturity. In effective DevOps teams, both runbooks and SOPs are living documents that evolve with systems, technologies, and organizational priorities, but they are designed and governed differently. Clarifying their roles early on avoids redundancy, ensures

faster incident response, and improves overall operational resilience.

#### IV. RUNBOOK DESIGN PRINCIPLES FOR DEVOPS TEAMS



The core principles of designing effective runbooks for DevOps and SRE teams

Designing effective runbooks requires more than just capturing instructions in a document. In high-availability environments, runbooks must be engineered for speed, clarity, repeatability, and usability under pressure. DevOps teams rely on these guides during system failures, degraded performance, or time-sensitive operations—so a poorly written or outdated runbook can significantly increase Mean Time to Resolution (MTTR) and even contribute to cascading outages. Good runbooks are modular, easy to follow, and designed to align with automation and monitoring systems.

One of the most critical principles is actionability. A runbook should offer immediate, actionable steps. It should start with a clear trigger condition—often defined by an alert or system signal—followed by verification checks, remediation steps, and validation procedures. Each action should include commands, expected outputs, and next steps depending on results. Ambiguity is dangerous in high-pressure situations, so every instruction must be unambiguous and concise.

Another principle is minimalism with context. Engineers reading a runbook during a 2 AM incident don't need a lecture—they need fast results. However, some minimal context (like the service scope, risks, and escalation

conditions) should be included at the top to help orient the reader. Use diagrams or flowcharts where appropriate to visualize decision paths or dependencies, especially when multiple outcomes or conditions exist.

Modularity helps scale runbooks across teams and services. Rather than writing one monolithic document for every scenario, break common procedures into reusable blocks. For example, a “Restart Service” or “Check Database Health” module can be referenced by multiple runbooks. This modularity supports consistency and easier maintenance when systems evolve.

Integration with automation tools is also a modern best practice. Runbooks that can trigger scripts, playbooks (e.g., Ansible), or pipelines (e.g., Jenkins, GitLab CI) enhance response speed and reduce manual error. Tools like StackStorm, Rundeck, or PagerDuty’s Runbook Automation can link monitoring alerts to automated or semi-automated runbook execution. Include links to scripts, templates, or even API endpoints within the runbook.

Finally, every runbook should define escalation paths and owner contacts. If automation fails or manual steps are unclear, the responder must know whom to contact. Embedding Slack channels, Jira links, or team rotation schedules ensures continuity and avoids dead ends. Well-designed runbooks are a cornerstone of operational excellence and, when maintained correctly, dramatically increase reliability and confidence during critical events.

#### V. SOP DEVELOPMENT AND ALIGNMENT WITH SLAS/SLOS

Standard Operating Procedures (SOPs) play a foundational role in ensuring repeatable, compliant, and measurable operations within high-availability (HA) systems. Unlike runbooks, which typically serve immediate and technical incident response use cases, SOPs are built around broader workflows and long-term service assurance. Effective SOPs are designed to align closely with Service-Level Agreements (SLAs) and Service-Level Objectives (SLOs), helping to translate business expectations into consistent operational execution.

The first step in SOP development is requirements gathering. This means collaborating with multiple stakeholders—DevOps teams, SREs, compliance officers, security leads, and service owners—to understand what processes must exist to meet business and regulatory requirements. For example, if an SLA mandates 99.99% uptime with no more than 5 minutes of downtime per month, SOPs must define clear procedures for activities like patching, release rollback, failover testing, and capacity scaling to prevent SLA breaches.

Each SOP should begin with a purpose and scope section, clearly outlining what the document covers, when it should be used, and which teams or services it applies to. Following that, the SOP should define prerequisites (such as user permissions or required tools), dependencies, and step-by-step procedures. SOPs are not just technical documents—they are also compliance artifacts, so version control, change history, approval signatures, and audit logs are essential components. Integration with documentation platforms such as Confluence, Git, or ServiceNow helps enforce governance and accessibility.

To keep SOPs relevant, teams must establish maintenance and review cycles, such as quarterly audits or after-action reviews following major incidents or releases. These updates ensure that evolving systems and SLAs remain aligned with procedures on paper. Some organizations implement workflow diagrams or BPMN (Business Process Model and Notation) models to improve clarity, especially when the SOP involves decision points or coordination between multiple roles.

Effective SOPs also improve onboarding and knowledge sharing, reducing the dependency on tribal knowledge and enabling junior engineers to contribute meaningfully. When aligned properly with SLOs, SOPs become instruments of reliability—they define how to meet performance and availability targets in a predictable, defensible manner. They also help organizations satisfy ISO, SOC, or HIPAA audit requirements by proving that core processes are not only defined but are actively followed and enforced.

## VI. AUTOMATION-READY RUNBOOKS AND DYNAMIC EXECUTION

Modern DevOps and Site Reliability Engineering (SRE) practices are increasingly focused on automation as a critical enabler of speed, consistency, and resilience. As systems scale in complexity and demand, traditional static runbooks no longer suffice. Today's high-availability environments require automation-ready runbooks—structured documents that can trigger workflows, integrate with CI/CD pipelines, and dynamically adapt to system state. These runbooks function as both human-readable instructions and machine-consumable playbooks.

The first step in automation readiness is to standardize the format and structure of runbooks. Using templates with clearly defined fields (e.g., triggers, preconditions, command blocks, rollback steps, success/failure criteria) allows tools and scripts to parse them for dynamic execution. Markdown, YAML, or JSON-based documentation is especially useful for this purpose, as it supports both readability and programmatic integration.

Next, automation-ready runbooks must be tightly integrated with observability and monitoring systems. When a monitoring tool like Prometheus or Datadog detects a threshold breach, it should be able to trigger a specific runbook or a linked automation task. This can be achieved using orchestration platforms like StackStorm, Rundeck, or even ChatOps bots that allow responders to run predefined tasks directly from Slack or Teams. Automation is most effective when it's both proactive and reactive—handling known issues automatically, and guiding engineers through unknown ones with interactive runbooks.

Dynamic execution is enhanced through Infrastructure as Code (IaC) and configuration management tools. For instance, if a runbook step requires increasing pod limits in a Kubernetes deployment, it can directly invoke a Helm upgrade or a Terraform apply. This eliminates the lag between diagnosis and remediation. Additionally, parameterized runbooks allow the same document to be reused across environments by accepting variables like region, service name, or deployment stage.

One emerging practice is the use of self-updating runbooks, where documentation is generated or updated in real-time based on system changes, Git commits, or incident postmortems. This ensures operational instructions remain in sync with infrastructure. Furthermore, access control and audit trails are crucial—every automated action should be logged, traceable, and reversible.

In essence, automation-ready runbooks transform operational knowledge into executable frameworks. They reduce toil, lower the risk of manual error, and enable rapid, scalable response to incidents. In the future, as AI and ML become more integrated into DevOps pipelines, these runbooks will evolve further—becoming intelligent, self-optimizing agents that drive autonomous operations.

## VII. CASE STUDY: BUILDING A SCALABLE RUNBOOK LIBRARY FOR A CLOUD-NATIVE PLATFORM

A leading global SaaS company operating a multi-region, cloud-native platform encountered persistent operational bottlenecks due to fragmented runbook documentation and inconsistent incident response practices. With services deployed across AWS, GCP, and Kubernetes clusters, and on-call rotations involving geographically distributed teams, the organization struggled with issues such as high mean time to resolution (MTTR), poor knowledge transfer, and frequent escalations during incidents. Recognizing these challenges, the company embarked on a structured initiative to build a centralized, scalable runbook library tailored for high-availability environments.

The first step involved a comprehensive audit of existing runbooks, which were scattered across wikis, internal Git repos, and shared folders. Many documents were outdated, lacked consistent formatting, or required tribal knowledge not recorded anywhere. To solve this, the team created a unified runbook repository using markdown files stored in Git, version-controlled and organized by service domain. A tagging system was introduced to categorize runbooks by severity, function (e.g., networking, database, CI/CD), and environment (e.g., staging, production, canary).

To ensure uniformity, the team introduced runbook templates with defined sections such as trigger condition, verification steps, remediation, rollback, owner, and escalation path. These templates enforced clarity and allowed for programmatic parsing by automation tools. The DevOps team integrated these runbooks with their incident management tool (PagerDuty), linking specific alerts to context-aware runbook entries. As a result, when an alert fired—for example, "Redis memory usage critical"—the system automatically presented the corresponding runbook to the on-call engineer, complete with commands and rollback instructions.

One of the key success factors was stakeholder engagement. Engineers from different product and infrastructure teams participated in quarterly runbook review sessions. These sessions helped refine content based on recent incidents and evolving architecture. Additionally, automated tests and linters ensured runbook quality and flagged broken links or deprecated commands. Integration with Slack enabled command execution from chat via bots, further improving incident response efficiency.

The outcomes were measurable: a 35% reduction in MTTR, fewer escalations to senior engineers, and increased confidence among newer team members. The runbook library also supported compliance audits by demonstrating operational readiness and consistency. Ultimately, the case study underscored the value of treating documentation as a living, versioned, and integrated component of platform reliability.

## VIII. COMMON PITFALLS AND HOW TO AVOID THEM

While building and maintaining runbooks and SOPs is essential for operational excellence, many organizations fall into predictable traps that can render even well-intentioned documentation ineffective. Recognizing and avoiding these pitfalls is critical for ensuring that these assets actually deliver value during high-stress, high-stakes operational events.

One common pitfall is overdocumentation, where teams attempt to cover every possible scenario in exhaustive detail.

This often results in bloated documents that are difficult to navigate in real time. Instead, runbooks should be concise and task-focused, aimed at resolving a specific issue or guiding a known workflow. Supporting information and edge-case discussions can be offloaded to linked documents or wikis to preserve clarity.

Conversely, some teams suffer from underdocumentation, where runbooks omit key steps, assumptions, or validation checks. This is especially dangerous in high-availability systems where small missteps can cascade into broader failures. For example, a runbook that says "restart the service" without specifying which container, namespace, or flags to use introduces unnecessary ambiguity. To avoid this, use structured templates that require inputs such as environment, service name, expected outcomes, and verification commands. Another major issue is lack of testing and review. It's not uncommon for runbooks to contain outdated steps due to system changes, decommissioned services, or renamed endpoints. Runbooks should be tested regularly—during chaos engineering drills, during blameless postmortems, or in simulated incidents. Embedding review cycles into sprint planning or using automation to flag stale entries helps ensure that documentation stays accurate and relevant.

A frequently overlooked pitfall is ignoring the user experience. Runbooks written by experts for experts can alienate junior engineers or new team members. Including context, clear headings, estimated time per step, and links to deeper documentation helps make them more accessible. Similarly, using consistent formatting, bulleted lists, and diagrams can significantly improve readability.

Finally, many organizations fail to assign ownership to individual runbooks or SOPs. Without clear accountability, documentation quickly becomes stale and untrusted. Every runbook should have a designated owner responsible for updates and quality control. When documentation is treated as code—with pull requests, peer reviews, and versioning—it becomes a first-class citizen in the reliability ecosystem.

## IX. BEST PRACTICES AND GOVERNANCE MODELS

Implementing effective runbook and SOP frameworks at scale requires not just technical documentation, but a structured approach to governance and best practices. As high-availability environments grow in complexity, so does the need for disciplined documentation management. Without clear ownership, review processes, and quality standards, even well-written runbooks can quickly become obsolete or untrustworthy.

A foundational best practice is to assign clear ownership of each runbook and SOP. Every document should have a designated owner or maintainer—typically the team or individual most familiar with the relevant service or system. This person is responsible for ensuring the documentation remains accurate, current, and actionable. When runbooks are treated as code, with pull requests, reviews, and version control (e.g., in Git), ownership becomes transparent and auditable.

Quality control is another essential pillar of governance. This includes the use of standardized templates to ensure consistency across documents, linting tools to flag missing sections or outdated commands, and automated validation scripts to test critical steps. Peer review processes should be enforced for every major runbook change, especially for workflows that impact production. Organizations can also adopt a documentation maturity model to assess and improve their SOP and runbook practices over time.

Searchability and discoverability play a key role in how useful documentation is during incidents. Metadata tagging—such as by service name, severity level, or functional category—allows engineers to quickly find the right runbook. Integration with search engines, incident management tools, and communication platforms (e.g., linking runbooks in Slack, PagerDuty, or Jira) ensures runbooks are accessible exactly when and where they're needed.

Governance models should also define review cadences and retirement cycles. Just as code has technical debt, documentation can accumulate operational debt. Establishing quarterly reviews or tying reviews to release cycles helps keep documentation fresh. Runbooks that haven't been used in six months or more should be flagged for review, especially if tied to deprecated systems.

Cross-team collaboration is equally important. Platform teams, SREs, application developers, and security engineers should jointly contribute to and review runbooks, especially those tied to shared infrastructure or business-critical services. Training programs and onboarding sessions can include walkthroughs of common runbooks and SOPs to help new team members build familiarity and confidence.

#### **Future Trends: AI-Driven Runbooks and Intelligent SOPs**

As DevOps, SRE, and platform engineering continue to mature, the future of runbook and SOP design is increasingly being shaped by artificial intelligence (AI), machine learning (ML), and automation-first paradigms. The next evolution of operational documentation goes beyond static procedures—ushering in intelligent, context-aware, and self-improving systems that transform how teams respond to incidents and manage complex environments.

One major trend is the emergence of AI-driven runbooks, which use historical incident data, telemetry, and real-time metrics to recommend or even execute operational actions. These systems learn from past responses—how certain issues were resolved, what worked, what failed—and generate suggestions or automated remediations based on similar patterns. For example, if a database latency alert appears similar to a prior incident, an intelligent runbook system could propose the same resolution steps or pre-fill a checklist for the engineer.

Natural Language Processing (NLP) is also enabling conversational interfaces to operational knowledge. Engineers can query systems using everyday language—e.g., “What should I do if Kafka has high consumer lag?”—and receive context-specific guidance drawn from a dynamic knowledge graph of SOPs, logs, and system state. Integration with ChatOps platforms like Slack or Microsoft Teams further bridges human-machine interaction, making intelligent SOP access seamless during incident response.

Another key innovation is self-healing automation triggered by intelligent runbooks. Instead of waiting for a human to execute remediation steps, AI systems can automatically detect, verify, and act on anomalies—such as restarting failed pods, scaling services, or routing traffic—based on predefined rules or adaptive behavior. These actions are logged and auditable, preserving governance while accelerating recovery.

Dynamic runbooks are also becoming self-updating. They can ingest updates from infrastructure-as-code repositories, postmortem analyses, or change records, keeping their instructions in sync with the live system state. As environments become more ephemeral and declarative (e.g., with Kubernetes, Terraform), documentation must evolve just as rapidly.

Finally, we are beginning to see autonomous operations copilots—AI agents that assist SREs by running diagnostics, guiding decisions, and even initiating repairs autonomously within defined guardrails. These copilots don't replace engineers but augment their decision-making, allowing teams to scale operations without proportionally scaling headcount. The future of operational readiness lies in intelligent documentation that is living, learning, and deeply integrated into the systems it describes—enabling not just faster response, but truly proactive and predictive reliability.

## **X. CONCLUSION**

In today's era of distributed systems and cloud-native architectures, where uptime, resilience, and agility are paramount, the importance of well-engineered runbooks and SOPs cannot be overstated. These tools are not just operational documents—they are living assets that

encapsulate institutional knowledge, guide rapid response, and reduce cognitive load during high-pressure events. For DevOps and Site Reliability Engineering (SRE) teams operating in high-availability environments, structured and actionable documentation serves as a critical bridge between design intent and operational reality.

Throughout this article, we have examined the strategic role of runbooks and SOPs in driving consistent, predictable, and compliant responses across complex infrastructures. From designing templates aligned with SLAs and SLOs to implementing automation-ready, dynamic runbooks that integrate with alerting systems, the modern playbook has evolved into a dynamic, interactive layer of reliability engineering. Case studies and practical guidance have shown that organizations who treat their documentation with the same rigor as code—applying version control, peer review, testing, and ownership—are better equipped to scale operations while maintaining control, visibility, and compliance.

We've also explored the potential of integrating intelligent triggers, observability, and AI into documentation practices, transforming static instructions into adaptive, self-healing workflows.

As runbooks become more machine-consumable and interconnected with incident management tools, the boundary between human and automated action continues to blur. The future of runbook engineering lies in context-aware, data-driven systems that learn, evolve, and optimize over time.

However, the journey to effective documentation is not without challenges. Many organizations struggle with outdated or overly verbose runbooks, lack of governance, and disconnected silos of operational knowledge. By following best practices in governance, maintaining continuous feedback loops, and embracing collaborative authoring models, teams can avoid these common pitfalls and build a culture of operational excellence.

In closing, runbooks and SOPs are no longer just post-incident references—they are proactive enablers of system reliability, on-call readiness, and organizational maturity. When embedded into the development and deployment lifecycle, they serve not only as first responders during outages but also as architectural guardians that safeguard against systemic failure.

As platforms grow in complexity, the ability to scale operational intelligence through structured, intelligent documentation becomes a competitive advantage—and a cornerstone of modern DevOps.

## REFERENCES

1. Riyani, N.P., Widaningrum, S., & Lalu, H. (2019). Perancangan Standar Operating Procedure (Sop) Audit Internal Sesuai Iso 9001:2015 ( Klausul 9.2), Iso 14001:2015 (Klausul 9.2) Dan Iso 19011:2018 Dengan Mempertimbangkan Resiko Menggunakan Metode Business Process Improvement Di Pt. Telehouse Engineering.
2. Mulyandari, Y.P., Widaningrum, S., & Lalu, H. (2019). Perancangan Standard Operating Procedure (Sop) Pengendalian Informasi Terdokumentasi Sesuai Dengan Iso 9001:2015 Dan Iso 14001:2015 Klausul 7.5 Dengan Mempertimbangkan Risiko Menggunakan Metode Business Process Improvement Di Pt.Telehouse Engineering.
3. Wahib, M. (2016). DESIGN OF A MINIATURIZED SOP INTEGRATED GNSS RF FRONT-END MODULE.
4. Roza Albareta, A., & Mursanto, P. (2019). Design of Standard Operating Procedure for Requirement Engineering in Software Development: Case Study Data Processing Integration Subdirectorat Statistics Indonesia. *Journal of Physics: Conference Series*, 1175.
5. Sembiring, N., Panjaitan, N., & Angelita, S. (2018). Design of preventive maintenance system using the reliability engineering and maintenance value stream mapping methods in PT. XYZ. *IOP Conference Series: Materials Science and Engineering*, 309.
6. Jibril, F., Yasser, B.M., Abdulwahed, M., Hasna, M.O., Benammar, M.A., & Ghani, S. (2015). Contributions of a Competition-Based Complex Engineering Design Experience to Leadership Development in Engineering Students.
7. Muhlisin, I., Darmawan, I., & Hediyanto, U.U. (2018). Analisis Dan Perancangan Standar Operasional Prosedur (sop) Service Operation Menggunakan Iso 20000 Dan Itilv3 Dengan Metodologi Pdca (plan, Do, Check, Act) Pada Unit Kerja Sistem Informasi Bagian It Support Pt Len Industri (persero).
8. Bin, J. (2012). Application of SOP in quality management system of medical equipment.
9. Lukei, M., Hassan, B., Dumitrescu, R., Sigges, T., & Derksen, V. (2016). Requirement analysis of inspection equipment for integrative mechatronic product and production system development: Model-based systems engineering approach. 2016 Annual IEEE Systems Conference (SysCon), 1-7.
10. Bowles, J.E. (1988). FOUNDATION ANALYSIS AND DESIGN. FOURTH EDITION.
11. Kluge, A., Silbert, M.R., Wiemers, U.S., Frank, B., & Wolf, O.T. (2019). Retention of a standard operating procedure under the influence of social stress and refresher training in a simulated process control task. *Ergonomics*, 62, 361 - 375.

12. Lennartson, B., Bengtsson, K., Yuan, C., Andersson, K., Fabian, M., Falkman, P., & Åkesson, K. (2010). Sequence Planning for Integrated Product, Process and Automation Design. *IEEE Transactions on Automation Science and Engineering*, 7, 791-802.