harness

# Data Governance Best Practices for Software Delivery

# Contents

In the rapidly evolving landscape of modern software development, governance has emerged as a crucial practice for successful software delivery. Far from being a mere set of restrictive rules, effective governance provides the framework that enables organizations to balance innovation with control, ensuring consistent quality while managing risks in an increasingly complex digital environment.

Software delivery governance encompasses comprehensive frameworks, policies, and controls that guide how organizations develop, test, and deploy software. This structured approach ensures that development efforts align with business objectives while maintaining necessary controls for security, quality, and compliance. By establishing clear guidelines and automated controls, governance enables teams to move quickly while staying within appropriate boundaries.

The strategic value of governance cannot be overstated in today’s competitive landscape. It provides organizations with the ability to standardize practices across development teams, protect valuable assets and data, and enable rapid, reliable software delivery. This standardization becomes increasingly crucial as organizations scale their development efforts and face growing complexity in the technologies they employ.

In an era where security breaches and compliance violations can result in significant financial and reputational damage, governance provides mechanisms for identifying and mitigating risks throughout the software delivery lifecycle. Through systematic approaches to risk assessment and mitigation, organizations can protect themselves while maintaining their ability to innovate.

Teams that practice strong governance reap numerous benefits, including reduced risks and costs, improved quality and reliability, faster time to market, and increased stakeholder confidence. These benefits compound over time as governance practices mature and become embedded in organizational culture.

Looking toward the future, governance must continue to adapt to emerging technologies and methodologies. As cloud-native architectures, artificial intelligence, and increased automation become more prevalent, governance frameworks must evolve to address new challenges while enabling organizations to leverage new opportunities.

# Key Governance Regulations

As you can see, organizations must navigate a complex web of regulatory requirements while maintaining efficient software delivery practices. Understanding these regulations and their common threads is crucial for implementing effective governance frameworks that ensure compliance without hindering innovation. Let’s get an overview of the major regulations organizations must comply with.

- The General Data Protection Regulation (GDPR) stands as one of the most comprehensive and influential regulations globally. Introduced by the European Union, GDPR has set new standards for data protection and privacy, requiring organizations to implement strict controls over personal data processing, storage, and transfer. Its influence extends far beyond European borders, effectively establishing a global benchmark for privacy protection.
- The Health Insurance Portability and Accountability Act (HIPAA) governs healthcare data protection in the United States. It mandates stringent controls over patient health information, requiring healthcare organizations and their technology partners to implement robust security measures, maintain detailed audit trails, and carefully manage access to sensitive health data.
- The Sarbanes-Oxley Act (SOX) focuses on financial reporting accuracy and corporate governance. While primarily targeting financial systems, its requirements for control and transparency have broad implications for software delivery, particularly in systems that handle or influence financial data. SOX compliance demands robust change management processes and comprehensive audit capabilities.
- The Payment Card Industry Data Security Standard (PCI DSS) provides specific requirements for protecting payment card data. This global standard affects any organization handling credit card information, requiring particular attention to encryption, security testing, and access control.

# Common Requirements Across Regulations

Modern software delivery regulations, whether from GDPR, SOX, HIPAA, or industry-specific frameworks, share fundamental requirements despite their different origins and focuses. These common threads reflect universal principles of security, accountability, and risk management that organizations must address in their software delivery practices.

Across regulatory frameworks, one consistent requirement stands out: the need for comprehensive documentation of the software delivery lifecycle. Organizations must maintain detailed records of all system changes, including who made them, when they were made, and why they were necessary. This documentation must demonstrate clear links between business requirements, technical implementations, and deployed changes. Version control systems must capture not only code changes but also configuration modifications, deployment procedures, and testing results.

Every major regulation emphasizes the importance of robust access management. Organizations must implement role-based access control (RBAC) systems that restrict code access, deployment capabilities, and production environment modifications to authorized personnel only. Multi-factor authentication becomes mandatory for sensitive operations, while periodic access reviews ensure that permissions remain current and appropriate. Systems must maintain detailed logs of all access attempts, successful or otherwise.

Regulations universally mandate the protection of sensitive data throughout the software delivery pipeline. This includes encryption requirements for data both in transit and at rest, secure key management practices, and data minimization principles. Organizations must implement mechanisms to identify and protect personally identifiable information (PII) and other sensitive data types, ensuring they are handled according to applicable privacy laws and industry standards.

# Common Control Objectives

In the realm of software governance, control objectives serve as fundamental guidelines that help organizations maintain oversight and ensure quality throughout their software delivery lifecycle. These objectives form the backbone of effective governance frameworks, providing structure and direction for development teams while protecting organizational interests. Let's take a closer look at some of the key areas organizations need to focus on in order to execute a comprehensive governance framework.

- **Access control** stands as a primary objective in software governance, ensuring that only authorized individuals can access specific systems, code repositories, and deployment environments. This fundamental control extends beyond simple username and password protection, encompassing role-based access control, multi-factor authentication, and regular access reviews to maintain security integrity.
- **Quality assurance** emerges as a vital control objective, establishing standards and processes to maintain consistent software quality. This encompasses code review requirements, testing protocols, and performance benchmarks that must be met before software can progress through development stages to production deployment.
- **Compliance and regulatory adherence** represent increasingly important control objectives, ensuring that software development and deployment practices meet relevant regulatory requirements. This includes maintaining proper documentation, implementing required security measures, and ensuring appropriate data handling practices.
- **Documentation control** serves as a fundamental objective, ensuring that all aspects of software development and deployment are properly documented. This includes technical specifications, user manuals, deployment procedures, and incident response plans, providing essential reference materials for current operations and future maintenance.
- **Performance monitoring and optimization** represent ongoing control objectives, ensuring that software systems meet performance requirements and operate efficiently. This includes establishing performance benchmarks, monitoring system metrics, and implementing optimization processes when needed.
- **Incident management** emerges as a crucial objective, establishing processes for identifying, responding to, and resolving software-related incidents. This includes defining escalation procedures, maintaining incident logs, and conducting post-incident reviews to prevent future occurrences.

- **Vendor management** serves as an important control objective when third-party software or services are involved. This includes evaluating vendor security practices, ensuring service level agreements are met, and maintaining appropriate contractual controls.
- **Business continuity** represents a strategic control objective, ensuring that software systems can continue operating during disruptions. This includes disaster recovery planning, backup procedures, and failover testing to maintain system availability.
- **Risk management** stands as an overarching control objective, focusing on identifying, assessing, and mitigating risks throughout the software lifecycle. This includes regular risk assessments, implementing appropriate controls, and maintaining risk registers.

These control objectives work together to create a comprehensive governance framework that enables organizations to maintain control while fostering innovation and efficiency. By implementing these objectives thoughtfully and systematically, organizations can build robust software governance practices that protect their interests while enabling successful software delivery.

The effectiveness of these control objectives relies heavily on their proper implementation and regular review. Organizations must continually assess and adjust their control objectives to ensure they remain relevant and effective in an evolving technological landscape while supporting business objectives and maintaining necessary protections.

As software systems continue to grow in complexity and importance, these control objectives become increasingly crucial for maintaining effective governance. They provide the structure needed to ensure quality, security, and compliance while enabling organizations to deliver software efficiently and reliably.

# Achieving Strong Governance For Software Delivery

## Objective 1: Maintain Separation of Duties Between Application Code and Production

Instituting separation of duties throughout the software development lifecycle is an impactful security risk mitigation realized through role-based access controls (RBAC). It essentially involves clearly defining and limiting what duties or functions a given stakeholder can execute. For example, individuals or development teams responsible for writing code shouldn't be the ones to deploy it and manage it in a live production environment. Separation of duties would prevent any accidental or malicious code changes from being introduced into production without proper review and oversight.

## Objective 2: Prevent Untested Code From Reaching Production

Ideally, all code should undergo thorough functional and security testing prior to being deployed in production, a policy which greatly reduces the risks of bugs or security vulnerabilities finding their way to live applications.

Unit testing is frequently mandated and should be enforced via governance. Any code containing business logic and domain objects should undergo unit testing. These are typically classes in the codebase referencing actions and entities a user can perform and manage through the system. Ideally, they do not depend on third-party libraries or external frameworks and therefore they are easier to test at the unit level.

## Objective 3: Prevent Vulnerable Code From Reaching Production

Application security testing (AST) can be mandated as well and enforced in the pipeline, ensuring that all code be scanned for the purpose of uncovering known vulnerabilities, thus strengthening the application's security posture. Two types of security scans that should be run on code are SAST (Static Application Security Testing) and SCA (Software Composition Analysis). SAST is applied early in the SLDC, prior to code being compiled.

SAST scanners should be run on code on a regular basis, such as during periodic builds, at each code check-in, or during a code release. SCA tools are used to identify open source software within a code base, for the purpose of evaluating security, license compliance and overall code quality.

A recommended approach to ensuring that vulnerable code isn't promoted to the next development phase is to create security gates that test the code and policies that fail a pipeline if code is found to contain a high level of severity.

## Objective 4: Enforce Granular Access Controls Throughout The SDLC

Role-based Access Control (as mentioned above) is essential for enforcing clear separation of duties among software development stakeholders, and is key to the overall secure operation of the underlying development platform. While static entitlements are a good start, a granular RBAC is a superior approach to access control that enables tailored access control and governance that meets diverse organizational needs.

Granular access controls would enforce permissions based on specific applications, services, environments, triggers (conditions to execute pipelines or workflows), individual pipelines and deployments. For example, you might want to let a DevOps team manage deployment pipelines (create/update/delete/read) but only allow developers to deploy and view them. In addition, you might want to grant read-only access to execs so they can get visibility across your deployment pipelines.

## Objective 5: Ensure Rollback And Recovery Plans Are In Place

A software rollback is a safety net in the world of software development. As new features and improvements get introduced into an application, a rollback serves as a critical recovery strategy. It allows developers to revert to an earlier, stable version if a new release creates unexpected problems.

## Objective 6: Maintain Detailed Audit Trails

Establishing strong governance also requires audit trails that capture events in detail. The audit trail shows the date and time of the event, the user who made the change, actions taken, resources affected, corresponding organization, project, or module, and an event summary that shows the change.

Audit trails are highly useful for reviewing new user creation dates to support external audit requests, as well as to validate when new releases are deployed to production to remediate pending library vulnerabilities. Below are some related compliance standards and frameworks that state the requirement of logs and audit trails:

### AICPA Trust Services Criteria

CC4.1 - COSO Principle 16: The entity selects, develops, and performs ongoing and/or separate evaluations to ascertain whether the components of internal control are present and functioning.

CC7.2 - The entity monitors system components and the operation of those components for anomalies that are indicative of malicious acts, natural disasters, and errors affecting the entity's ability to meet its objectives; anomalies are analyzed to determine whether they represent security events.

ISO 27001:2013

A12.4.1 - Event logs recording user activities, exceptions, faults and information security events should be produced, kept and regularly reviewed.

A12.4.3 - System administrator and system operator activities should be logged and the logs protected and regularly reviewed.

ISO 27001:2022

8.15 - Logs that record activities, exceptions, faults and other relevant events should be produced, protected, stored and analyzed.

# Governance in the Age of AI

As you can see, organizations must build their CI/CD governance on a solid foundation of well-defined policies. These should encompass not only traditional software development concerns but also the unique challenges presented by AI systems. Development teams need clear guidance on compliance requirements, from data privacy regulations to industry-specific standards. Security protocols must be explicitly outlined, covering everything from code access controls to deployment safeguards. Perhaps most critically, teams require detailed ethical guidelines for AI development, ensuring their systems remain fair, transparent, and accountable.

These systems serve as vigilant guardians of the pipeline, continuously monitoring for potential issues. Static code analysis tools scan for security vulnerabilities, while specialized AI auditing systems examine models for bias and unexpected behaviors. Performance monitoring tools track system behavior in production, providing early warning of potential issues. This automated oversight ensures consistent enforcement of standards while freeing human reviewers to focus on more nuanced aspects of governance.

Every modification, whether to traditional code or AI models, must be tracked with precision. This includes capturing not only what changed but why it changed, who approved it, and what testing validated the change. Advanced version control systems should maintain detailed histories of model training data, parameters, and performance metrics. Code review processes must be enhanced to account for AI-specific concerns, such as model drift and data quality.

Effective CI/CD pipeline governance in the AI era requires a delicate balance between rigorous control and development agility. By establishing clear policies, leveraging automation, and maintaining robust change management processes, organizations can build trustworthy, efficient pipelines that support innovation while managing risk.

Ironically, AI can also be used to enforce policies. AI systems excel at enforcing complex policy requirements consistently across large organizations. These systems can automatically check code commits against security policies, ensure proper access controls are maintained, and verify that all necessary approvals are obtained before changes are implemented. Machine learning models can understand the context of changes, making intelligent decisions about whether modifications comply with governance requirements rather than simply applying rigid rules.

Perhaps the most valuable aspect of AI in governance is its ability to learn and adapt over time. AI systems can analyze the effectiveness of governance policies, identifying areas where rules may be too strict or too lenient based on real-world outcomes. These systems can recommend policy adjustments to optimize the balance between security, compliance, and development velocity. Through continuous learning, AI governance systems become more effective at distinguishing between genuine risks and false positives, reducing the burden on development teams while maintaining robust protection.

The integration of AI into software delivery governance represents a fundamental shift in how organizations approach compliance and risk management. As AI systems continue to evolve, they will enable even more sophisticated approaches to governance, helping organizations maintain high standards of security and compliance while accelerating innovation. The key to success lies in thoughtfully implementing these AI capabilities while maintaining human oversight of critical governance decisions.

# How a Change Advisory Board Can Help

A change advisory board (CAB) plays a pivotal role in modern software governance, serving as a critical control point for managing and overseeing changes to software systems and infrastructure. This governing body brings together key stakeholders from various departments to evaluate, authorize, and monitor changes, ensuring they align with organizational objectives while minimizing potential risks.

The primary purpose of a CAB extends beyond simple change approval. It serves as a strategic forum where proposed changes are thoroughly evaluated from multiple perspectives, including technical feasibility, business impact, security implications, and resource requirements. This comprehensive evaluation helps organizations avoid costly mistakes and ensure changes contribute positively to business objectives.

Risk management stands as a fundamental reason for implementing a CAB. By bringing together experts from different domains, organizations can better identify potential risks associated with proposed changes. This collective expertise helps anticipate and mitigate issues that might otherwise be overlooked, reducing the likelihood of service disruptions or security vulnerabilities.

Compliance requirements often necessitate formal change management processes, and the CAB serves as a key component in meeting these requirements. By maintaining detailed records of change evaluations, approvals, and implementations, organizations can demonstrate due diligence and compliance with various regulatory standards.

The CAB also serves as a vital feedback loop in the organization’s continuous improvement efforts. Through regular review of change outcomes and lessons learned, organizations can refine their processes and improve their ability to manage changes effectively over time.

# Conclusion

The fact that strong governance is essential to software delivery becomes clearer as organizations face increasing complexity and risk in their digital initiatives. By implementing effective governance frameworks, organizations position themselves for sustainable success, managing risks while enabling innovation and growth in an increasingly complex digital landscape.

Through thoughtful implementation of governance practices, organizations can achieve the delicate balance between control and agility, ensuring consistent quality while maintaining the speed necessary for competitive advantage in today's market. This balanced approach enables sustainable growth while protecting organizational assets and stakeholder interests.

harness

The AI-Native Software Delivery Platform™

**Follow us on**

𝕏 /harnessio

in /harnessinc

**Contact us on**

www.harness.io