# Automated Governance Blueprint

Modernizing risk and compliance to accommodate the rapid evolution and complexity of software

Security Technical Advisory Group

CLOUD NATIVE COMPUTING FOUNDATION

**Automated Governance Reference Architecture**
Andrés Vega in collaboration with David Langhorst, Matthew Flannery, Brandt Keller, Chris Hughes, and Jon Zeolla

Status: Draft
Control Rev 2022120-2

# Table of Contents

[Read This] Notes to contributors on collaboration:

We have established the initial content and framework for the document.

The document is divided into two major sections. The first two-thirds primarily consist of prose, presenting and advocating for a standards-based reference architecture built on open-source components. The final third, which I anticipate will expand to surpass the other sections by getting more into details, is currently dedicated to proposed patterns. It's this latter section where I'd like to direct most of your attention to help in producing the actual design of the architecture. All sections contain seed text at this stage, but again I expect the technical specification section to be more elaborated at a low level and have new components added to it.

For those of you contributing extensive viewpoints and perspectives to the initial sections, the necessary structure is in place. Following the model set in Open and Secure, if you wish to express your thoughts in your unique voice, please do so in commentary boxes located at appropriate points throughout the document. We are seeking politely opinionated insights from experts in our community who have relevant expertise. Depending on their length, we might also consider an appendix of ‘biographical stories' section to further explore each contributor's journey/stance on the subject.

I anticipate a commitment of about 6-12 hours of writing from each contributor to draft their respective parts they take on. As a group, we will then review what we have, identify any gaps, and decide on the next steps, including what to add, revise, or refine.

While we might propose minor edits to others' contributions for the sake of brevity, clarity, or tone, our goal is to let the diverse voices of our community shine through. So, please share your insights and expertise in the relevant sections to help enhance the document.

Specific areas where key contributions are sought include:

- Patterns
- Object model (tentative)
- API specifications
- Normalization of data from various sources
- Formatting of exported data for reporting processes
- Defining all the above constructs as proposed standards (tentative)

# Introduction

Welcome to the Cloud Native Computing Foundation “Automated Governance Reference Architecture”, a transformative framework designed to empower organizations to confidently and easily navigate the complex landscape of compliance and risk management. In today's business environment, characterized by rapid technological advancements and stringent regulatory demands, the integration of compliance with software development is more critical than ever. Though security is traditionally thought of as a blocker, the automation of compliance can in fact be an enabler to innovation rather than a barrier.

Managing the tools your organization uses to implement governance, risk, and compliance (GRC) policies can be a burdensome, cross-organizational task. By consolidating management into a unified interface—providing a "single pane of glass" perspective—we streamline the application of your GRC policies across the board. This centralization liberates your developers, security engineers, and operations personnel to concentrate on tasks critical to business success. This approach not only makes it easier to capture and implement strategic intentions but also guarantees consistent policy enforcement across diverse environments and platforms. Managing these tools collectively, yet distributing enforcement, reveals advantageous features otherwise unattainable with siloed management.

A wide variety of tools are available to ensure software meets certain standards before deployment to production environments. These tools each focus on particular areas of governance, such as change control, automated testing suites, warning flags, adherence to fundamental cryptographic best practices (including appropriate key lengths and avoidance of weak algorithms), code analysis, utilization of approved libraries, and the remediation of known public vulnerabilities. However, these individual tools often operate in isolation, lacking context and integration.

Managing a given set of tools–each designed for specific tasks such as testing, scanning, signing software– is a significant challenge. Ensuring security and compliance controls of those same tools are configured and applied correctly, and reliably recording and reporting their successful application has become an insurmountable operational burden. Ensuring that problems requiring remediation are dealt with correctly and promptly adds an additional, ongoing process burden.

Many organizations have attempted to build a coordination layer for these tools and processes. The required expertise spans several departments – engineering, operations, compliance, legal – and rallying these resources towards a functional system often leads to incomplete solutions or outright failures.

Our mission is to bridge this gap, providing innovative solutions that enable collaboration between compliance and software engineers. This architecture goes beyond being merely an additional compliance tool; it serves as a platform service designed to augment internal

development platforms. Our aim is to accelerate software delivery while upholding the highest standards of governance and compliance.

We envision a future in which regulatory challenges are addressed through efficient, precise, and innovative, thereby improving  the approach to governance in software engineering. Through this document, we detail the specifics of our architecture, illustrating how it achieves improvement of the current practices and processes, and sets a new standard for modern governance practices.

# Summary

Technology organizations are increasingly challenged to maintain compliance and manage risks within their software development processes. The emergence of rigorous regulatory standards and the potential for severe consequences, including criminal conviction, substantial fines and reputational damage, necessitate a strategic approach to governance in software engineering. The promise of operational efficiency and agility in software development, championed by platform engineering and cloud native technologies, is often hampered by the challenges of meeting governance, regulatory, and compliance requirements.

Traditionally, organizations have relied on written plans, questionnaires, and the Institute of Internal Auditors's Three Lines of Defense model to address governance, compliance, and risk management. While this approach may have sufficed for static systems, it falls short in the face of modern, complex, and distributed software environments.

Acknowledging these challenges, the need arises for a transformative solution that aligns compliance with business innovation, ensuring that regulatory mandates become an accelerating force that raises the bar in software  rather than an impedance.

In response to this need, we seek to facilitate where business governance, regulatory compliance, and innovation collaborate through a shared platform. Our vision is to bring compliance and risk management closer to architecture, making it an integral and seamless part of the software development journey. We envisage compliance engineering as a conduit to accelerate software delivery by shortening the loop to build secure, reliable, and thoroughly tested software. This approach not only safeguards against regulatory pitfalls but also empowers organizations to leverage compliance as an advantage..

This document presents a comprehensive reference architecture designed to automate engineering software governance, addressing the pressing needs of organizations to navigate and conform to raised organizational standards and evolving regulatory landscapes. Our architecture offers a solution that not only ensures compliance with current frameworks and US standards such as SOX, FedRAMP, PMA, and Australian standards such as ISM and PSPF, but also provides the agility to adapt to future regulatory changes.

# Background and Motivation

In today's era, where software systems are crucial to organizational operations, the importance of efficient governance in software engineering cannot be overstated. Companies, especially those experiencing rapid growth and operating in the cloud, face unique challenges when going public or interacting with regulated industries. Despite their innovative capabilities, they encounter the tension of needing to innovate while also facing stringent scrutiny from regulators, shareholders, and the public to avoid missteps and meet compliance and security standards. This situation highlights the complexities of scaling, maintaining software quality, and adhering to compliance, emphasizing the critical need for effective software governance.

This blueprint aims to encapsulate the essence of “well-governed” software development platforms, translating its theoretical underpinnings into a practical, automated framework for software governance. Recognizing the repetitive history of project failures and the often overlooked issues with organizational structure at the interface between business and software development, our architecture seeks to bridge these gaps through automation. Key themes such as alignment of interests and incentives, shared understanding of business value, and risk mitigation are integrated into a cohesive, automated governance model and methodology.

The core objective of this architecture is to provide organizations with a blueprint for implementing automated governance processes. These processes are designed not only to be compliant with regulatory standards like the Sarbanes-Oxley Act (SOX), oversight from the Office of the Comptroller of the Currency (OCC), Federal Information Security Modernization Act (FISMA) accreditations, Food and Drug Administration (FDA) Premarket Approval (PMA), and the Executive Order 14028 on “Improving the Nation's Cybersecurity”, but also to be agile enough to adapt to evolving landscapes of enterprise architectures, software development methodologies, and organizational structures. **By automating governance processes, we aim to enhance transparency, accountability, and efficiency. Automation helps streamline compliance, reduce manual oversight, accelerate decision-making processes, and thereby reduce the risks associated with software development and deployment. This ensures that these processes contribute optimally to the organization's overall performance and value creation.**

Effective governance significantly eases the burden of demonstrating standards adherence. This domain uniquely unites stakeholders from diverse functions, including developers, executive management, trust & safety departments, and legal teams. When misalignments occur, the responsibility often falls on developers to address the deviations. Our architecture empowers these developers with clear context and crisp requirements, enabling them to correct necessary deviations and achieve alignment with overarching business goals.

In the following sections, we will delve into the specifics of the proposed blueprint, detailing its components, processes, and the mechanisms through which it aligns recommended software engineering practices with overarching business goals. This approach is designed to actualize

the vision of modern governance, where automated processes facilitate seamless collaboration, ensure compliance, and enable swift adaptation to regulatory changes and business needs.

# The Forcing Function of Mandates: The Executive Order 14028, NIST SSDF (SP800-218), and CISA Self-Attestation

On May 12, 2021, the Biden Administration issued Executive Order 14028, aiming to bolster the cybersecurity framework of the United States. A key focus of this order, outlined in Section 4, is the enhancement of software supply chain security. In response, particularly to Section 4. (e) of the EO, NIST developed the "Secure Software Development Framework" (SSDF) (SP800-218). This framework is designed to assist government entities in countering software supply chain threats. Detailed in this special publication are a series of practices intended for integration within the Software Development Lifecycle (SDLC). The NIST SSDF categorizes these secure software development practices into four distinct groups.

- Prepare the organization (PO)
- Protect the Software (PS)
- Produce Well-Secured Software (PW)
- Respond to Vulnerabilities (RV)

There are 42 specific requirements needed to achieve compliance with the NIST framework. Each of these practices is underpinned by specific tasks that need to be demonstrably managed by all organizations governed by federal regulations, including but not limited to banks regulated by the SEC and the Federal Reserve, as well as entities in various other industries.

A more immediate and pressing priority arises from the Cybersecurity & Infrastructure Agency (CISA)'s recent call for feedback on their proposed Secure Software Self-Attestation Form. This form outlines a foundational set of requirements for government agencies, derived as a subset from the 42 items listed in the SSDF.

As the CISA attestation transitions from its draft phase to enforcement, which could occur imminently, organizations will face a tight timeline to prepare their attestations. Regardless of the approach Chief Information Security Officers (CISOs) choose for their organization's attestation, parties must begin collecting these for critical software within three months after the official finalization of CISA's self-attestation form.

When the secure software self-attestation form is finalized, organizations aiming to sell or continue selling software for governmental use will be obligated to certify their adherence to the SSDF principles. As stated in the form: "*This self-attestation form identifies the minimum secure software development requirements a software producer must meet, and attest to meeting, before their software subject to the requirements of M-22-18 may be used by Federal agencies.*

*This form is used by software producers to attest that the software they produce was developed in conformity with specified secure software development practices."*

- Secure development and build environments are utilized for the software.
- The software producer diligently ensures the integrity of source code supply chains.
- In a committed effort, the software producer uses automated tools or equivalent processes to safeguard source code supply chains.
- Provenance data is meticulously maintained for both internal and external code integrated into the software.
- Security vulnerabilities are proactively checked using automated tools or similar methodologies.

These represent the overarching categories of requirements, firmly rooted in the NIST SP 800-218, Secure Software Development Framework. Consequently, adherence to the self-attestation's objectives necessitates compliance with the SSDF's stipulations.

The responsibility for software security is shifting from users to providers. It explicitly places the responsibility on software suppliers to the federal government to implement the appropriate security measures as outlined in the NIST's SSDF framework.

The self-attestation form emphasizes the necessity of "reasonable" actions, especially in the realms of documenting and minimizing the use of risky software products in development and build environments, and establishing a comprehensive process. This process involves taking reasonable steps to ensure the security of third-party components and to effectively manage any vulnerabilities associated with them.

Regulations and liability are increasingly dictating how software is developed and shipped. It's no longer enough to simply test code thoroughly, quickly address vulnerabilities, and maintain a log of changes to production environments. These actions, along with others, must now be extensively documented, creating an abundance of evidence to reduce liability and meet regulatory demands. Compliance has become a pivotal point where engineering, security, quality, and risk management intersect in virtually all sizable organizations engaged in software creation.

Understanding regulatory rules is one aspect; comprehending which ones hold the potential to impact not only one's professional standing but also the financial and operational viability of a business is a far more complex issue. The consequences of these decisions can profoundly influence an organization's financial health and continuity, with ramifications extending to both personal careers and the existential stability of the business.

In the past, suppliers often routinely consented to a Software Development Lifecycle (SDL) policy. These policies represent more than mere words; they constitute an unspoken pact between the company's leadership and the end-users of the software. Commonly, within these documents, there is a statement implying that the development process is safeguarded through

a series of checks and balances. Yet, in many cases, this tends to be a broad statement of governance principles rather than an accurate depiction of the everyday practices and actual controls in place.

An opportunity lies in the perception and validation of self-attestation forms. This presents a significant opportunity to establish a framework, supported by verifiable proofs, which validates that all necessary requirements have been effectively implemented.

# Organizational Structures Typically in Place

The effective implementation of governance, particularly in the context of software development, typically involves a hierarchy of organizational structures, each playing a distinct role in the process:

1. **Board Level: Strategic Investment Management**

At the highest tier, the board focuses on strategic investment management. This involves overseeing the broader strategic implications of software development initiatives and ensuring alignment with the organization's long-term goals.

2. **Executive Level: Business Case Scrutiny and Requirements Management**

The executive level is responsible for scrutinizing business cases and managing requirements. This includes evaluating the viability of software projects and ensuring that they meet the business objectives and operational needs of the organization.

3. **Group Level: Technical Authority**

At the group level, technical authority is exercised. This involves making decisions on the technical aspects of software development, such as choosing the right technologies and methodologies to meet project goals effectively.

4. **Operational Level 1: Monitoring Execution, Risk, and Compliance**

The operational level has a dual role. Firstly, it involves monitoring the execution of key decisions, ensuring that projects are progressing as planned and are measured. Secondly, it focuses on managing risk and compliance, ensuring that all activities adhere to regulatory standards and organizational policies.

5. **Operational Level 2: Design Review and Architecture Compliance**

Another critical function at the operational level is to conduct design reviews and ensure architecture compliance. This entails assessing software designs for their effectiveness and efficiency and ensuring they comply with established architectural standards and best practices.

Each of these levels plays a vital role in creating a cohesive and effective governance structure, ensuring that software development aligns with both strategic and operational objectives.

# Target audience: Platform Engineers and DevSecOps Teams

Within the domain of software engineering governance, a diverse array of teams and roles converge to forge a path toward effective governance, compliance, and risk management. At the strategic level, management sets the overarching direction, while specialized teams like Change Management Boards, Legal, and Trust & Safety focus on their respective areas of expertise - from controlling change implementations to oversight of policy controls. Despite these varied roles, a common thread binds these teams: the collective aim to accelerate time-to-market, steer clear of adverse publicity, and align closely with business objectives. Yet, amidst this collaborative effort, the brunt of the responsibility often rests with those directly involved in crafting the software - where the tangible actions meet the strategic intentions.

In this dynamic, a significant transformation is taking shape in the domain of compliance and risk management. The rising discipline of "compliance engineering" emerges as a vital and distinct practice, catalyzed by the recognition that traditional approaches to governance are increasingly insufficient to tackle the complexities and rapid pace of modern software development.

This paradigm shift has led organizations to recalibrate their focus, increasingly tasking software engineers for embedding and automating compliance controls within their development processes. Compliance engineers at the convergence of platform engineering and DevSecOps, therefore, become key players in this new arena, transforming regulatory mandates into practical, code-based solutions that integrate seamlessly with the software delivery pipeline. Their role transcends the conventional boundaries of merely generating compliance reports; it involves actively ensuring that every facet of software development resonates with the stringent regulatory standards and requirements.

## Platform Engineers

**Role and Challenges:** Platform Engineers are pivotal in developing and maintaining the technical infrastructure that supports software development. They often grapple with integrating compliance and governance requirements into a dynamic and scalable infrastructure.

Inspired by SRE principles, Platform Engineers can leverage governance automation to transform traditional governance processes. This approach involves automating manual,

repetitive governance tasks, focusing on strategic, high-value outcomes, and easing the process by which operators maintain a compliant environment.

**Benefits for Platform Engineering:** This reference architecture offers Platform Engineers a structured framework to embed compliance and governance seamlessly into the infrastructure. It provides tools and methodologies to automate governance processes, ensuring infrastructure scalability and reliability while adhering to regulatory standards. Through improved efficiency in governance-oriented tasks, reduced direct and indirect costs of manual governance, and enhanced ability to scale governance processes with organizational growth.

## DevSecOps Teams

**Role and Challenges:** DevSecOps Teams are at the forefront of integrating security, development, and operations. Their challenge lies in balancing rapid development cycles with stringent security and compliance requirements.

DevSecOps teams can integrate governance automation principles to make governance an enabler of DevOps, rather than an obstacle. This includes automating compliance checks and risk controls, allowing for faster and more secure software delivery.

**Benefits:** Through reduced toil in governance processes, clear definition of governance roles and responsibilities, and enhanced security and compliance posture through automated control verification, integrating security and governance throughout the software development life cycle. It enables these teams to maintain a high pace of innovation and deployment without compromising on security or compliance. The architecture facilitates real-time risk assessment and compliance monitoring, making governance a part of the continuous integration/continuous deployment (CI/CD) pipeline.

## Common Benefits for All Parties:

**Streamlined Compliance:** Automated tools and processes reduce the manual burden of compliance, allowing both Platform Engineers and DevSecOps teams to focus on innovation and development.

**Enhanced Collaboration:** The architecture fosters improved collaboration between cross-functional teams to accelerate change approval that often hinders delivery, ensuring that compliance and governance are integrated into every aspect of the software development and deployment process.

**Adaptive Framework:** Given the ever-evolving nature of technology and regulations, the architecture is designed to be flexible and adaptable, enabling Platform Engineers and DevSecOps teams to swiftly respond to new challenges and changes in the regulatory landscape.

**Adopting an Engineering Mindset for Governance**: Both groups can benefit from approaching governance with an engineering mindset, identifying and implementing solutions that address the root causes of governance and delivery toil.

**Real-Time Governance Measurement:** Utilizing concepts like Governance Level Indicators (GLIs) and Governance Level Objectives (GLOs) for real-time governance measurement, similar to SLIs and SLOs in SRE.

Commentary Box: Comparing Compliance Engineering to the Transition from Infosec to Security Engineering

The emergence of compliance engineering bears striking similarities to the transformation witnessed in the field of information security (infosec) in recent years. Much like traditional roles evolving into specialized security engineering positions, the compliance engineering domain represents a parallel evolution within compliance and risk management.

While traditional GRC tools have primarily focused on helping organizations prepare retrospective reports and document compliance activities, they often fall short in equipping compliance engineers with the tools and capabilities required for proactive control implementation. In contrast, compliance engineering is forward-looking, emphasizing the embedding of compliance controls directly into the software development lifecycle. This shift not only enhances efficiency and accuracy but also ensures that compliance is not a mere checkbox exercise but an integral part of the software delivery process.

As organizations continue to grapple with the need for both speed and compliance, compliance engineering bridges the gap between these seemingly conflicting objectives. It empowers software engineers to embrace compliance as a design principle rather than an afterthought, ultimately leading to more secure and compliant software systems.

# Benefits and Outcomes of Automated Governance

**Comprehensive Compliance Assessment:** Automated Governance empowers organizations to collect and assess the configuration and state of their systems and infrastructure against the most stringent control frameworks, regulatory standards. This comprehensive evaluation ensures that every aspect of your technology ecosystem aligns with the highest compliance requirements, reducing the risk of non-compliance.

**Real-Time Alerting for Remediation**: The proposed reference architecture provides real-time alerting capabilities, instantly notifying your teams of any discrepancies or deviations from compliance standards. This proactive approach allows you to identify and address potential compliance issues before they escalate, minimizing the associated risks and penalties.

**Out-of-Compliance Remediation:** Automated Governance goes beyond mere detection by offering automated remediation workflows. When discrepancies are identified, the proposed system can automatically initiate corrective actions such as patching in the event of identified vulnerabilities and roll back to last known state, ensuring rapid and consistent remediation. This automation not only reduces the burden on your teams but also accelerates the process of returning to compliance.

**Attainment of Safe Harbor:** In the unfortunate event of a material breach or compliance incident, Automated Governance is crucial in helping your organization attain safe harbor status. By consistently monitoring and demonstrating a commitment to compliance, you can leverage this evidence to mitigate the impact of breaches, potentially reducing regulatory fines and legal liabilities.

**Enhanced Regulatory Confidence:** The ability to collect, evaluate, and remediate compliance issues in real time instills confidence in regulatory bodies. Regulators are more likely to view your organization favorably when they see proactive, automated compliance measures in place. This can lead to smoother regulatory audits and interactions.

**Operational Efficiency:** By automating compliance checks and remediation, Automated Governance significantly improves operational efficiency. Your teams can focus on strategic tasks rather than manual compliance, leading to cost savings and streamlined operations.

**Reduced Compliance Costs:** The proactive, automated nature of our solution reduces the overall cost of compliance. Organizations can allocate fewer resources to manual compliance tasks, resulting in lower operational expenses and more predictable compliance budgets.

**Faster Time-to-Market:** Accelerating compliance and remediation processes means faster time-to-market for your products and services. You can deploy new software releases, features, and updates with confidence, knowing that governance is being actively managed and maintained.

Commentary Box: Organizational-Wide Benefits

While many of these benefits may appear to benefit only a subset of roles at an organization, the reduction of friction throughout an organization has superlinear returns. By reducing misunderstandings and confusion, projects see fewer delays, and processes are able to become more independent, improving their efficiency and improving the ability for smaller teams to maintain complete ownership and control over their responsible systems. This reduction allows team members to improve their recall of project context alongside the status of initiatives from inception and completion.

Additionally, institutional observability becomes attainable, improving the ability for Executives and Boards to make well-informed decisions regarding strategic investment and requirements management.

Finally, by automating policy and governance, companies can reduce or remove the need for certain recurring meetings such as change approval boards and architecture committees replacing them with as-needed collaborations and asynchronous information sharing.

All of this allows for the expedited adoption and experimentation with new processes and products by technical and non-technical users alike.

# Why Automated Governance Matters to Smaller or Less Regulated Organizations

**Reducing Development and Maintenance Burdens**

Automated Governance abstracts system interfaces into well-defined APIs based on open standards, significantly easing the definition, development, and maintenance of proofs from both the compliance and software development views.

Its platform-agnostic nature allows deployment across various environments, offering flexibility and potential cost savings in platform technology changes.

**Enhancing Developer Efficiency**

Automated evidence creation and consistent policy evaluation streamline processes, freeing developers from manual tasks and trial/error approaches, allows for focus on their primary goal: delivering high-quality software efficiently.

**Expertise Pool**

Addressing the scarcity of expertise in regulated system management is a major challenge. Automated Governance enables organizations to tap into specialized security, compliance, and development knowledge without incurring high costs.

Automated Governance not only streamlines the process of integrating compliance into development but also significantly decreases the onboarding time for new developers. By providing a clear and automated framework for governance, it reduces the learning curve associated with understanding and implementing regulatory requirements. This efficiency directly translates into faster time-to-market, as developers can prioritize core software development over navigating complex compliance procedures. Additionally, this approach inherently reduces risk by ensuring consistent and accurate adherence to compliance standards, making it easier to maintain regulatory integrity in a fast-paced development environment.

**Interoperability in Cross-Organizational Tech Environments**

Automated Governance establishes a standards-based framework for exchanging governance evidence, and facilitating technology integration, especially in scenarios like organizational mergers or third-party risk assessments.

Moreover, it fosters a common comprehension of what defines appropriate evidence, thereby empowering organizations to confidently select and integrate new technology systems, assured by the minimized need for manual oversight.

Commentary Box: The Misconception of One-Size-Fits-All in Governance

Many organizations fall into the trap of a one-size-fits-all approach to governance. Automated Governance challenges this notion, offering a flexible framework that can be tailored to diverse organizational needs, emphasizing that effective governance is not uniform but adaptable.

Commentary Box: The Challenge of Custom Solutions

In automated governance, numerous highly regulated organizations have pursued their own custom solutions, encountering substantial obstacles. The primary issue is the lack of alignment with established standards, causing significant onboarding and scalability challenges.

Custom solutions often fall short due to their detachment from industry standards, making them hard to integrate with existing systems and adjust to organizational growth or changes in regulations.

The outcome is clear: many of these bespoke systems are inadequate, failing to efficiently onboard users or scale, leading to inefficiencies and potential compliance risks.

# Differentiation between Automated Governance and Traditional Compliance Software

In distinguishing Automated Governance from traditional compliance software, our solution is particularly adept at handling software that undergoes frequent changes. This adaptability is crucial in environments characterized by rapid development and deployment cycles, such as those involving Kubernetes clusters, serverless functions, and AI/ML systems.  Our Automated

Governance framework is designed to ensure compliance and governance continuity amidst these high development velocity and delivery scenarios, where traditional compliance tools, often tailored for less fluid environments and workflows, may struggle to keep pace.

Additionally, our solution goes beyond the traditional scope of compliance frameworks or policy engines. It is a comprehensive platform that integrates compliance requirements with the software delivery lifecycle. This integration enables teams to assert and attest their compliance status in real time, thereby saving time and reducing manual effort. It enhances the efficacy of audits while significantly reducing the associated manual toil. By removing governance, risk, and compliance (GRC) as a bottleneck, it accelerates software delivery. Moreover, this platform provides a holistic approach to achieving common protection goals, offering a more comprehensive solution to governance and compliance challenges.

The US Department of Defense in a recent publication titled 'DevSecOps Continuous Authority to Operate Evaluation Criteria' outlines some of the capabilities and areas of assessment needed to work towards cATO (Continuous Authority to Operate) in a heavily regulated environment. For readers unfamiliar with the US Department of Defense DevSecOps Reference Design, cATO is a modern approach to how current compliance aligns to a RMF (Risk Management Framework) by introducing an authorization process that supports the development and delivery of software in an Agile, DevSecOps enabled ICT (Information and Communication Technology) environment. From a Defense perspective, the reasons why such a modern approach to Governance, Risk and Compliance become obvious - In order to meet the immediate needs of the warfighter, the ability to provide software that is secure, delivered rapidly, and is continuously improved is crucial.

# Use Case Examples

**Example 1: Frequent Deployments and Compliance Readiness**

A fintech startup, facing the challenge of deploying new features multiple times a day, integrates Automated Governance into their CI/CD pipeline. This architecture ensures that with each deployment, compliance checks are automatically performed, aligning with industry-specific regulatory standards. For instance, when a new payment feature is pushed, the system instantly verifies its compliance with PCI DSS standards. This seamless integration allows the organization to maintain a rapid deployment schedule without compromising on compliance, addressing the challenges described by teams deploying software frequently. Finally, from the perspective of a PCI QSA the audit process is by consequence streamlined.

**Example 2: Scaling Compliance in a High Growth Tech Company**

A rapidly growing SaaS provider implements Automated Governance to handle the expanding scope of its compliance needs. As the company enters new markets and faces diverse

regulatory environments, the architecture scales its compliance checks and controls accordingly. This scalability ensures that as the company grows, its software remains compliant with varying international data protection laws, such as GDPR in Europe and CCPA in California, thereby supporting its global expansion.

**Example 3: Transition from Manual to Semi-Automated Compliance**

An e-commerce company, historically reliant on manual compliance processes, adopts Automated Governance to transition towards a semi-automated system. The new architecture automates routine compliance tasks, like data privacy checks and audit trail management, while allowing for manual oversight in complex decision-making scenarios. This hybrid approach streamlines their compliance operations, reducing the time and resources spent on manual checks, and marks a significant step in their digital transformation journey.

**Example 4: Addressing Specific Compliance Challenges**

An online healthcare platform faces stringent HIPAA compliance requirements. Automated Governance is employed to meticulously manage patient data handling, ensuring encryption and access controls are in place and auditable. Each data transaction is logged and checked against HIPAA standards automatically. This system not only helps the company avoid potential violations but also builds trust with its users, ensuring their sensitive health information is handled with the utmost care and compliance.

# Maturity measurement and enabling Automated Governance

Readers familiar with papers such as the CNCF Secure Software Factory Reference Architecture, US DoD Enterprise DevSecOps Reference Architecture and similar may be asking the question: “How do I get from a foundational DevOps or DevSecOps capability to Secure Software Factory and/or Automated Governance”

Whilst frameworks such as SLSA exist, we have prepared a maturity scale that can assist an organization in determining the types of controls and processes to have in place..

## Software Supply Chain Security - Signing

| Maturity | Description |
|---|---|
| Level 0 | Software artifacts are not digitally signed and/or no auditability exists. |
| Level 1 | Basic code signing with self managed keys and manual processes? |
| Level 2 | Decentralized code signing and enhanced key sec? |
| Level 3 | Automated signing of software artifacts within CI/CD processes, automated verification of digital signatures, a policy engine provides automated decision making on deployments of artifacts, keyless signing, attestations are created and contained alongside deployment artifacts in secure artifact repositories (including for example signed SBOM, signed Vulnerability Analysis results i.e. trivy) |

## Evidence & Attestations

| Maturity | Description |
|---|---|
| Level 0 | No evidence is captured. |
| Level 1 | Evidence is captured on an ad-hoc or manual basis (i.e. as part of an audit or assessment) |
| Level 2 | Evidence is automatically captured and attestations are created yet with little action performed on evidence or attestations |
| Level 3 | Evidence is captured for all relevant security controls and is clearly mapped to a policy document. Captured evidence has attestations which are well formed and include all listed attributes within Reference Architecture Technical Specifications |

| | |
|---|---|
| | (i.e. Event Declaration, Asset Identification, etc). Documentation exists for attestation creation and verification process, automated processes exist for capturing evidence and creating attestations, automated cryptographic verification of attestations where possible (i.e. a Quality Gate) as well as ad hoc verification processes available. |

## Control Validation

| Maturity | Description |
|---|---|
| Level 0 | No validation or attestation of security controls |
| Level 1 | |
| Level 2 | |
| Level 3 | Test coverage is measured and continuously validated through automated processes, control validation is entirely automated.<br><br>Policies are validated autonomously and automatically, and security controls are evaluated continuously for policy compliance. |

## GRC & Policy Enforcement (Maybe just ‘GRC Engineering’ ? )

| Maturity | Description |
|---|---|
| Level 0 | Policies are outdated, not accounting for modern application delivery or security practices and are enforced through manual or ad-hoc processes (i.e audits / assessments) |
| Level 1 | Reporting on compliance is partially automated |
| Level 2 | |
| Level 3 | Policies and System Security Plans (SSPs) are well formed, continuously optimized, account for modern application security and application delivery and compliance of policies is automated. Policies can be automatically created (i.e. OSCAL) and testing of security controls for compliance is automated. Reporting on compliance fully automated. |

# Integration with Existing Systems

The Automated Governance architecture is designed to integrate seamlessly with a range of existing software development tools, ensuring a smooth transition and minimal disruption to current operations. This integration is critical for organizations seeking to enhance their compliance and audit readiness, particularly in fast-paced DevOps environments.

## Strategies for Seamless Integration and Transition

**Compatibility with Common Tools:** The architecture is compatible with widely-used development and monitoring tools, allowing for easy integration into existing workflows.

**Modular Design:** Its modular design facilitates incremental adoption, where organizations can start small and scale up, integrating more components as needed.

## Enhancing DevOps Practices for Compliance and Audit Readiness

**Frequent Deployments:** The architecture supports frequent deployments by automating compliance checks, ensuring that every release meets the required standards without slowing down the development process.

**Managing Short-lived Services:** It provides tools to track and audit short-lived services, ensuring compliance even in ephemeral environments.

**Blurred Lines Between Development and Operations:** Automated Governance bridges the gap between development and operations, providing clear visibility and control over the entire software lifecycle, from code development to deployment and operations.

By aligning with current DevOps practices, Automated Governance ensures that compliance and audit processes are embedded within the development pipeline, rather than being external checkpoints. This integration not only maintains the agility and speed of DevOps but also enhances the overall security and compliance posture of the organization.

| Commentary Box: The Unseen Cost of Manual Compliance |
| --- |

The hidden costs of manual compliance – from slowed innovation to increased human error – can significantly impact an organization's agility and growth. Automated Governance not only mitigates these costs but also turns compliance into a strategic asset.

[TO DO] Add metrics

# Risk Management Details

Automated Governance plays a crucial role in enhancing risk management strategies, particularly in environments characterized by frequent deployments and dynamic services. The architecture is tailored to address the unique challenges posed by such fast-paced development cycles while ensuring ongoing security and compliance.

## Strategies for Security and Compliance in Dynamic Environments

**Automated Compliance Checks:** During frequent deployments, the architecture automatically performs compliance and security checks, ensuring that each release adheres to the latest standards.

**Per Commit:** The architecture incorporates real-time monitoring tools that actively assess the security posture with every commit and pull request. This ensures continuous evaluation and rapid identification of potential vulnerabilities, allowing for immediate notification and maintaining robust security throughout the development lifecycle.

## Approach to Risk Identification, Assessment, and Mitigation

**Comprehensive Risk Identification:** The architecture includes integration advanced scanning and analysis tools to identify a wide range of risks, from data breaches to compliance violations.

**Continuous Risk Assessment:** It employs continuous risk assessment techniques, providing ongoing insights into the security and compliance status of the software.

**Proactive Risk Mitigation:** Upon identifying risks, the architecture initiates automated mitigation processes, such as patching vulnerabilities or updating configurations, to address issues promptly.

## Methodologies for Continuous Risk Assessment and Real-Time Monitoring

**Real-Time Alerts and Reporting:** It offers real-time alerts and detailed reports, enabling quick response to emerging threats and comprehensive risk management oversight.

Adopting these strategies and methodologies, Automated Governance ensures that risk management becomes an integral, efficient element of the development cycle, enhancing security across all operations.

# Fostering Technological Advancement in Software Development

Automated Governance is not just about ensuring compliance and security; it's also a catalyst for innovation in software development. By integrating cutting-edge technologies and methodologies, this architecture reshapes how organizations attain high standards in the dynamic context of progressive delivery environments.

**Incorporating Advanced Technologies for Streamlined Compliance**

The architecture streamlines compliance checks, shifting away from traditional manual processes. This transformation is key in fast-paced development settings where agility is paramount.

Transparency logs and digital signing are employed to ensure the integrity and traceability of compliance records.

**Seamless Integration of Compliance into Continuous Delivery**

Embedding audit readiness directly into the continuous delivery process, the architecture allows development teams to focus on innovation without being bogged down by compliance concerns.

The CI/CD pipeline integrates real-time compliance monitoring and automated audit trails, balancing swift deployment and regulatory compliance.

**Encouraging Innovation While Maintaining Compliance**

Emphasizing the use of containerization and microservices, the architecture empowers teams to explore new features and technologies while safeguarding the system's overall compliance.

With features like automated rollback and feature toggling, the architecture enables rapid testing and deployment, fostering an environment of agile development and innovation

**Promoting Creative Solutions and Rapid Development**

The architecture's adaptable nature allows for quick responses to changes, enabling the deployment of new features with automated compliance management running in parallel.

By minimizing the compliance workload for developers, the architecture nurtures a culture of innovation, giving them the freedom to focus on developing groundbreaking software solutions.

Automated Governance redefines the balance between compliance and development, facilitating innovative software development within the framework of stringent regulatory standards.

# Advancement of Pipeline Design: Control Points for Decentralized Decision-Making

In May 2018, Capital One shared insights in a blog post titled "Focusing on the DevOps Pipeline," where they outlined their approach to software delivery. Their philosophy emphasized the delivery of software that is not only high in quality, with minimal defects and full compliance, but also thoroughly tested to ensure functionality across all aspects. They stressed the importance of swift delivery, but not at the expense of quality.

The post also introduced the concept of "gates" or "control points" in the development pipeline, essential in maintaining standards throughout the software development lifecycle. Capital One's specific implementation included 16 guiding principles for their pipelines, covering aspects from source code version control and code coverage to vulnerability scans and automated rollbacks.

These control points, serving as both metadata and evidence, document crucial actions taken throughout the software's development, production, and promotion stages. This ensures that each software iteration meets rigorous standards at every stage of the pipeline, with each control point serving as both a checkpoint and a record of compliance and quality assurance action. They are strategically performed at every step of the continuous integration process and are meticulously recorded in build logs or artifact creation logs. This methodical capture of pipeline activities or tasks through control points paves the way for a shift towards decentralized decision-making, moving away from the centralized inspection models predominant in many organizations.

# Reference Architecture Technical Specifications

The system aims to provide an automated governance framework focusing on ensuring the integrity of assets and the security of running applications. It does by ascertaining trust within an organization's infrastructure and delivery pipeline.

Functional Requirements:

1. Automated Governance: The system should automatically track governance metrics and compliance data throughout the software delivery pipeline and its underlying Infrastructure.
2. Integrity Checks: The system should validate the integrity of code, configurations, and data at each control point of the pipeline and all aspects of the underlying infrastructure.
3. Traceability: Must record all changes and actions in the software delivery process.
4. Security Measures: The system should integrate security checks and monitoring features to ensure the security of deployed applications and infrastructure.

Non-Functional Requirements:

1. Scalability: The system should be scalable to accommodate a growing number of projects and applications.
2. Usability: The system should have a user-friendly interface for easy navigation and data interpretation.
3. Compliance: The system itself should comply with relevant industry standards and regulations.

The system will be designed based on a modular, microservices-based architecture, in a memory-safe language, leveraging cloud native solutions for scalability and resilience.

Integrations with third-party event and data sources will be implemented with a plug-in (also referred to as adaptors) architecture of data-source specific agents for both collection and analysis.

Collected data will be normalized and stored in immutable datastores for integrity and non-repudiation.

For each control point the model identifies a set of inputs, outputs, actors, and the actions that can occur at that point.

Next, the model identifies a set of risks that can be attributed to the control point. Finally, based on the identified risks, a set of controls are chosen to mitigate the risks and attest to the input, output, actors, and actions involved.

# Constructing and Storing Evidence in the form Attestations

**What is an Attestation?**

Attestations play a pivotal role in Automated Governance. They serve as critical records that detail the occurrences within the chain, offering transparency and accountability. The following attributes that well formed attestations should exhibit:

1. **Event Declaration:** Attestations clearly state the specific event that occurred, outlining the action or series of actions that took place.

2. **Asset Identification:** They identify the asset(s) involved, providing clarity to what the event pertained to, be it software components, tools, or processes.

3. **Timestamp and Context:** Attestations include precise timestamps, offering a contextual timeframe of when the event occurred. This temporal metadata is crucial for tracking and auditing purposes.

4. **Condition Description:** The conditions or circumstances that led to the event are described, offering insights into how the event transpired.

5. **Event Outcome:** Details on the result or output of the event are provided, giving a clear picture of the event's consequences or end state.

6. **Policy Compliance Check:** The role of attestations in assessing policy compliance can vary. Generally, an attestation itself does not directly indicate whether an event has met predefined policy criteria. Instead, it provides the necessary data that can be used to determine compliance. In specific instances, such as with a verification summary as outlined in the SLSA framework, the attestation may directly reflect the outcome of a policy verification process. In these cases, the attestation can signify a pass or fail status. However, in most scenarios, attestations serve as a record of events or conditions, which are then evaluated against the set policy criteria to determine compliance.
7. **Verifiability:** They furnish essential information necessary to reproduce the event's outcome, ensuring that the data can be independently verified for accuracy and authenticity.

8. **Identity of attestor:** Any individual or entity can create an attestation. In order to enable consumers to extend trust to the attestations, the attestation’s creator needs to be strongly (i.e. cryptographically) linked to the content.

## Event Driven Streaming Architecture

Building on tthe 2019 “DevOps Automated Governance Reference Architecture”, the event-driven streaming architecture within the Automated Governance Reference Architecture utilizes a message queue (for instance Kafka) to establish a robust and responsive system for managing and processing governance-related events. This architecture is designed to capture,

store, and analyze data in real-time, ensuring the integrity of assets and the security of running applications through the proposed automated governance framework..

In this architecture, events are organized into streams using keys that create logical groupings. These keys are pivotal in correlating related events, facilitating efficient data management and retrieval. For example, build identifiers and various event types, such as pull request identifiers, vulnerability findings, and test results, serve as keys. This method allows for the precise tracking of changes and incidents, enhancing the governance and compliance processes.

A stream processor plays a critical role in this setup by orchestrating the flow of data from multiple streams. It constructs an attestation model by aggregating and correlating events, enabling comprehensive monitoring and evaluation of compliance with policies and controls. This approach not only supports the enforcement of policy as code using OPA (Open Policy Agent) and Rego but also ensures that digital attestations of compliance are accurately generated and signed, leveraging tools like Sigstore for digital signing.

# Infrastructure auditing

Infrastructure risks are both periodically audited for compliance as well as audited when certain automated events occur (GitOps, for example)

- Data protection
- Confidential computing
- Infrastructure access controls for enforcement of least privilege

## Development infrastructure

Integration examples: GitHub, GitLab, GCP, Azure, Kubernetes, physical hardware

| Risks | Controls |
| --- | --- |
| 1. Unauthorized access to source<br>2. Unauthorized access to development environments and data | 1. Source control permissions are correct<br>2. IAM/RBAC configuration |

## Deployment infrastructure

Integration examples: AWS, GCP, Azure, Kubernetes, physical hardware

| Risks | Controls |
| --- | --- |
| 1. Unauthorized account access<br>2. Unintended account authorization<br>3. Unauthorized network access<br>4. Ephemeral credential expiration<br>5. PKI configuration | 1. IAM/RBAC configuration<br>2. IAM/RBAC configuration<br>3. Networking rules applied correctly<br>4. Confirm that expired credentials are invalid<br>5. Validate certificates are being properly issued |

### Deployment State

Integration examples: AWS, GCP, Azure, Kubernetes, physical hardware

| Risks | Controls |
| --- | --- |
| 1. Software improperly deployed<br>2. Old versions of software deployed<br>3. Malicious activity on deployed system | 1. Check state of running systems<br>2. Check signatures of running applications<br>3. Monitor activity at the host level |

## Software delivery pipeline auditing

### Source Code Repository and Code Quality

Integration examples: GitHub, GitLab, Clang, Coverity, SonarQube

| Risks | Controls |
| --- | --- |
| 1. Unapproved changes<br>2. Untested changes<br>3. Unapproved dependency<br>4. Information (secrets) leakage<br>5. Low quality code | 1. Peer review<br>2. Test coverage<br>3. Approved dependencies<br>4. Scan for information leaks<br>5. Code analysis / linting |

### Build and Dependencies

Integration examples: GitHub, GitLab, Chainguard, Snyk, Blackduck, Sonatype

| Risks | Controls |
| --- | --- |
| 1. Inaccurate, unapproved build configuration<br>2. Build information is missing, modified, or inconsistent<br>3. Unapproved dependency or version<br>4. Improper licensing of dependencies<br>5. Dependencies may have known security issues<br>6. Build output is untested<br>7. Unapproved versions of dependencies being used | 1. Build configuration in source control and peered review<br>2. Immutable build configuration<br>3. Linting<br>4. Upstream approved dependency store<br>5. Approved dependencies<br>6. License check<br>7. Unit test succeeded<br>8. Linting<br>9. Static analysis<br>10. Vulnerability check<br>11. Reachability analysis |

## Artifacts

Integration examples: Artifactory, Docker Trusted Registry, AWS CodeArtifact

| Risks | Controls |
| --- | --- |
| 1. Unknown and potentially dangerous artifacts<br>2. Artifacts may not have proper licensing<br>3. Artifacts may have known security issues<br>4. Unapproved versions of artifacts being used<br>5. Unsigned artifacts | 1. Download only from approved internal or external sources<br>2. License check<br>3. Vulnerability check<br>4. Version check<br>5. Signature check |

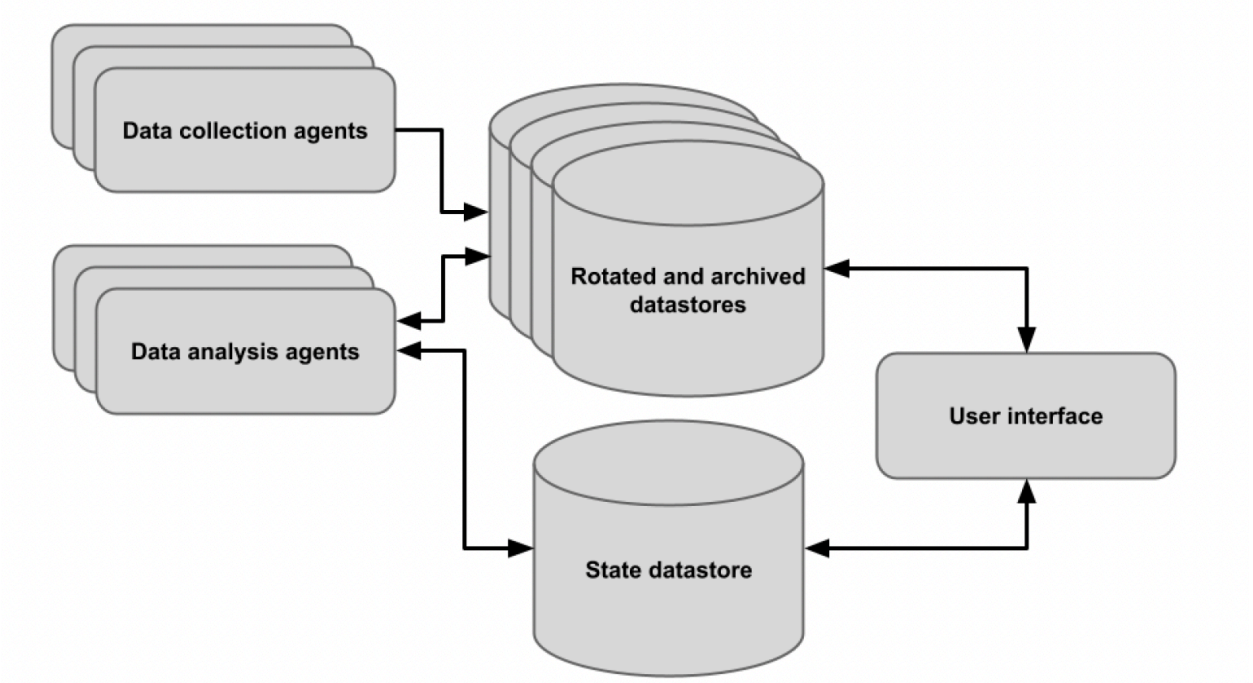
## Deployment

Integration examples:

| Risks | Controls |
| --- | --- |
| 1. Deployment of software from untrusted source<br>2. Unsigned artifacts | 1. Only from trusted source<br>2. Signature check<br>3. Prod/dev environment check |

| | |
|---|---|
| 3. Improper configuration or data for environment | |

# System Architecture




# Symbols and Icons

- **Checkmark (✓):** To indicate a passing state.
- **Exclamation Mark (!):** To signal a warning or needs attention.
- **Cross (✗):** For a failing state.

# Leveraging Emerging Standards: Secure Controls Framework and OSCAL

The Secure Controls Framework (SCF) is a comprehensive cybersecurity framework that provides a collection of cybersecurity best practices, controls, and guidelines aimed at helping organizations design, implement, and manage a robust cybersecurity program. Developed to be technology-agnostic and industry-neutral, SCF covers a broad range of cybersecurity topics and is designed to align with existing standards, regulations, and best practices, making it a versatile tool for compliance and risk management. The framework offers a structured methodology for identifying and prioritizing control requirements, thereby simplifying the complex landscape of regulatory mandates and industry standards.

The Open Security Controls Assessment Language (OSCAL) is an initiative by the National Institute of Standards and Technology (NIST) aimed at standardizing the documentation, assessment, and continuous monitoring of security controls in information systems. OSCAL provides a standardized set of XML, JSON, and YAML formats that enable the clear and consistent expression of security controls, their implementation, and assessment procedures. The goal of OSCAL is to facilitate more efficient and automated security assessments, reduce errors and inconsistencies, and improve the speed at which new regulations can be implemented into an organization's cybersecurity framework.

While both SCF and OSCAL aim to improve cybersecurity postures, they serve slightly different purposes and can actually complement each other. SCF is more focused on providing a comprehensive set of cybersecurity controls and best practices that organizations can adopt, whereas OSCAL is designed to standardize the way these controls are documented, assessed, and continuously monitored. SCF provides the "what" — the controls that should be considered — and OSCAL provides the "how" — a standardized format for describing, implementing, and assessing these controls. Organizations could potentially use SCF to identify which controls are most relevant to their operations and then use OSCAL to document and assess the implementation of those controls in a standardized manner.

# Future Directions

## Automated Reasoning and Formal Proofs

Semantic-based automated reasoning and formal verification methods offer powerful tools for evaluating the compliance controls in development platforms and application environments. By leveraging formal languages to describe both the regulatory requirements and the system configurations, semantic-based reasoning can systematically analyze if the rules and settings within a system aligned with specification..

Automated reasoning algorithms can then compare this formal representation against the

system's actual configurations to identify any discrepancies or gaps in compliance. Formal verification methods take this a step further by not just identifying if the system is compliant at a given moment, but also by ensuring that it will remain compliant under various conditions and scenarios. Using mathematical models to represent the system's behavior, formal verification provides exhaustive proof that the system will or will not meet certain criteria.

For instance, it can prove that data will always be encrypted when transferred externally, thereby satisfying a specific regulatory requirement. These methods offer a high degree of assurance that the regulatory intent not only matches the current system configuration but will continue to do so in the future, thereby minimizing the risk of non-compliance.

## AI and LLMs

AI and Large Language Models (LLMs) like GPT-4 have a range of potential applications in Automated Governance by interrogating the system in natural language:

**Generating Control Mappings:** One of the most labor-intensive aspects of governance is mapping controls to specific regulatory requirements. LLMs could assist by automatically generating these mappings based on the text of the regulation and the available controls within the system. With continual updates and training, the model can adapt to changes in regulations, ensuring that the control mappings remain accurate over time.

**Expression of Logical Object Models:** LLMs can be trained to understand the intricacies of a system's logical object model. This means they could assist in automatically generating or updating these models, ensuring they are both complete and compliant with any relevant regulations. This would be particularly valuable in complex systems where manual modeling is prone to errors or omissions.

**Writing TLA+ Policies:** Temporal Logic of Actions (TLA+) is a formal specification language used to describe systems. LLMs could potentially be trained to write TLA+ policies based on simpler, natural language requirements. This would make it easier to create formal specifications for complex governance systems, thereby facilitating formal verification methods.

**Reasoning about Threat Models and Data Flows:** LLMs can analyze text-based threat models or data flow diagrams and then reason about their impact on compliance. For example, if a new data flow introduces a potential for non-compliance with a particular regulatory standard, the model could flag this for further review.

**Automated Explanation and Justification:** Formal reasoning often involves intricate logical proofs that are hard for non-experts to understand. LLMs can act as a bridge between the formal methods and human operators, offering natural language explanations and justifications for the reasoning process.

**Code Reviews for Compliance:** Given a sufficiently detailed understanding of regulatory intent and coding best practices, LLMs can be used to review code and configuration files, highlighting areas that may be non-compliant.

**Automated Queries and Checks:** LLMs could be integrated into the system to perform ongoing compliance checks, querying the system's status and comparing it to the formal requirements to ensure ongoing compliance.

**Technical to Legal Translation:** Translation of data exports evidencing low level procedures and implemented requirements mapped back into compliance parlance for ease of interpretation by non-technical individuals.

### Quantizing of Risk(?) [TO DO]

# Conclusion and Call to Action

As we reach the culmination of our exploration of the Automated Governance Reference Architecture, it is clear that this innovative framework is not just a compliance solution, but a strategic enabler for organizations navigating the complex landscape of modern software development.

## Embracing Automated Software Governance

The architecture we have presented is a testament to the potential of integrating compliance seamlessly into the software development lifecycle. It embodies a forward-thinking approach that aligns with the rapid pace of technological advancement and ever-evolving regulatory demands.

## Call to Action: Transforming Compliance into a Strategic Advantage

We urge organizations, especially those in high-growth and dynamic sectors, to consider the adoption of this architecture. It's an opportunity to transform compliance from a cumbersome necessity into a strategic advantage.

For developers, project managers, auditors, and IT leaders, this architecture offers a pathway to enhance efficiency, security, and innovation within your software development processes.

## Moving Forward with Automated Governance

We invite you to explore the possibilities that Automated Governance opens for your organization. Whether it's reducing the burden of compliance, accelerating time-to-market, or simply building more secure and reliable software, this architecture is a key to unlocking these benefits.

Take the first step in this journey by reviewing your current compliance processes and considering how Automated Governance can integrate into and enhance these systems.

# Appendixes

## Common Terminology and Definitions

**Automated Governance:** The use of automation to manage and enforce governance policies in software development processes.

**cATO:** continuous Authority to Operate integrates with an organization's Risk Management Framework (RMF). cATO is a modernized approach that facilitates continuous compliance and automated security checks within the RMF structure. This continuous process ensures that software meets security standards more efficiently, streamlining the RMF steps for quicker deployment and updates in IT environments, while maintaining the rigorous security assessment required by the RMF.

**DevOps**: A set of practices that combines software development (Dev) and IT operations (Ops), aimed at shortening the systems development life cycle and providing continuous delivery with high software quality.

**Continuous Integration (CI)**: The practice of automating the integration of code changes from multiple contributors into a single software project.

**Continuous Deployment (CD):** A software engineering approach in which software functionalities are delivered frequently through automated deployments.

**Control Points:** Specific checkpoints in a software development pipeline where compliance and quality standards are assessed.

**Compliance Engineering:** The discipline of embedding compliance requirements directly into the software development lifecycle.

**Gates:** In a software development pipeline, points of evaluation or decision-making where certain criteria must be met before the process continues.

**Attestations:** Records or evidence in a software development process that certain criteria or standards have been met.

**Metadata:** Data that provides information about other data, often used for management and control in software environments.

# References:


Ross, J. W., & Weill, P. (2002). Six IT decisions your IT people shouldn't make. Harvard Business Review. https://hbr.org/2002/11/six-it-decisions-your-it-people-shouldnt-make

Finkelstein, A. (2009). Software Engineering Governance. University of Oregon. https://www.cs.uoregon.edu/events/icse2009/images/postConf/TB-Governance-ICSE09.pdf

Pal, T. (2018). Focusing on the DevOps Pipeline. Medium. https://medium.com/capital-one-tech/focusing-on-the-devops-pipeline-topo-pal-833d15edf0bd

Nygard, M., Magill, S., Guckenheimer, S., & Willis, J. (2019). DevOps Automated Governance Reference Architecture. IT Revolution. https://itrevolution.com/product/devops-automated-governance-reference-architecture/

Fred, A. M. (2018). Compliance and Audit Readiness: The DevOps Killer?. IT Revolution. https://itrevolution.com/articles/compliance-audit-readiness/

Magill, S., Edenzon, M., Bantu, R., & Betz, C. (2023). Reinventing Software Asset Inventory. IT Revolution. https://itrevolution.com/product/reinventing-software-asset-inventory/

Bensing, B. (2022). Governance Engineering. IT Revolution. https://itrevolution.com/articles/governance-engineering/

Platt, M. (2022). Why GRC needs SRE: Integrating Security Where Reliability is Managed [Video] https://www.youtube.com/watch?v=jVPIBwdoZMs

Thomson, J., & Laing, D. (2019). Extending the Error Budget Model to Security and Feature Freshness [Video] https://www.youtube.com/watch?v=lXPmh9Ap114