The Future Depends on Enabling Sustainable Analytics Governance

How teams maintain structure in analytics through automated quality enforcement

Executive Summary

It is a common misconception that investing in an analytics platform immediately produces better decisions. Analytics software is necessary, but the organizational commitment doesn't end there. Many businesses fail to optimize the benefits of their analytics investments. Organizations that succeed with analytics succeed because they recognize the delivery of analytics as a complex process. They adopt the engineering disciplines required to operate analytics at scale.

This paper argues that the moment analytics is developed internally, it becomes a software system under continuous change. Dashboards, data models, and pipelines behave like software artifacts. They must be designed, developed, versioned, tested, governed, deployed, and monitored. When organizations treat analytics as reporting rather than engineering, predictable failure modes emerge – fragility at scale, loss of trust in numbers, uncontrolled change, and mounting technical debt.

The core thesis is simple: **analytics programs fail because systems lack structure.**

Drawing on fifty years of software engineering practice, this paper shows how analytics follows the same evolutionary path and why o**rganizations must adopt engineering frameworks such as SDLC, DevOps, testing discipline, governance, and agile delivery** to achieve reliability and trust. Analytics Engineering must not be a phrase that sounds foreign. Analytics can and must learn from the structured success of software development discipline.

Within this framework, QSDA Pro plays a specific and necessary role. It is not intended to replace analytics tools or governance processes. Instead, it strengthens the engineering system by providing continuous validation across the build, test, and monitor stages of the analytics lifecycle – where regression, drift, and silent breakage most often occur. The insights produced by QSDA Pro extend beyond testing, supporting documentation, governance evidence, and operational confidence.

This paper proceeds in three parts:

- First, we examine the origins of analytics and how we arrived at where we are. It explains why analytics teams unintentionally became software teams without engineering discipline.
- Second, it describes the failure modes that result from treating analytics as craft, with dashboard jockeys as the artisans of our day.
- Third, it presents a unified engineering model for analytics and demonstrates how QSDA Pro fits within a disciplined system.

The conclusion is not that organizations must become software companies, but that by taking on analytics, they have become one by default. Businesses must **operate like a software development company** if analytics is to be trusted, scalable, and sustainable.

Contents

The Classic Management Model

For most of the twentieth century, management theorists delivered a consistent message to business leaders: stay true to your core mission. Organizations were advised to focus on what they did best and avoid drifting into unfamiliar territory. Historical analyses of U.S. corporate structure show that prior to the widespread adoption of the multidivisional form in the 1950s and 1960s, most large firms operated focused, single-business organizations rather than diversified conglomerates (Chandler, 1998). Focus was strategic.

Peter Drucker emphasized this discipline. He argued that effectiveness required concentration rather than diffusion of effort (Drucker, 1954). Michael Porter later framed competitive strategy as a deliberate choice about what not to do. Prahalad and Hamel formalized the idea through “core competencies,” defining the capabilities that deserved sustained investment (Porter, 1996). The message was consistent: organizations succeed by aligning resources tightly to mission and resisting distractions.

This classical model assumed a boundary between the business and the tools that supported it. Accounting systems, operational reports, and management information systems existed to serve the mission. Data was simply a byproduct of operations.

That assumption that business strategy must adhere to a core mission no longer holds.

Using Data as a Strategic Asset

For decades, boardrooms discussed this as: Mission. Vision. Tactics. Execution. Data followed, as the rear-view mirror of early BI reports. It documented what had already happened. But once data was recognized as an asset, the flow reversed. To extract value from data, organizations began adopting capabilities that had not previously been part of their operating models.

For decades, boardrooms discussed this as: Mission. Vision. Tactics. Execution. Data followed, as the rear-view mirror of early BI reports. It documented what had already happened. But once data was recognized as an asset, the flow reversed. To extract value from data, organizations began adopting capabilities that had not previously been part of their operating models.

Data and the analytics it required needed more than eyes on. It required systems.

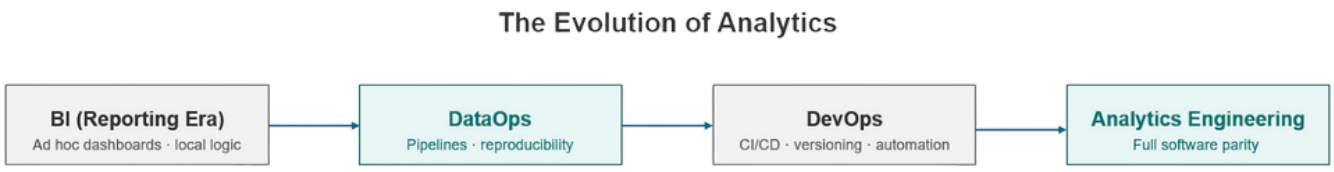
To harness data at scale, businesses began standing up new functions. BI teams, analytics groups, and data units. They performed activities historically associated with software development: requirements gathering, modeling, development, testing, deployment, support, and maintenance. To make analytics reliable, they adopted behaviors and frameworks drawn directly from software engineering: SDLC thinking, Agile delivery, DevOps automation, version control, and testing.

By the late twentieth century, data, once thrown overboard as flotsam, had moved below deck and become the engine itself.

Using BI Teams as Software Teams

As analytics matured, its artifacts – reports, dashboards, semantic models, and data pipelines – became no different than those in software development in terms of complexity, risk, and strategic importance. When numbers drive financial decisions, regulatory reporting, operational performance, or clinical outcomes, reliability and auditability become critical. The rigor of software engineering becomes unavoidable.

When data emerged as a strategic asset, most organizations did not respond by hiring software engineers. Instead, they assigned responsibility to people who already understood the business. These were Business Analysts, typically in finance or operations. Finance led the way. Analysts were already comfortable with spreadsheets, formulas, and structured data. They were natural candidates to build reports, metrics, and dashboards.



As BI evolves toward analytics engineering, dashboards and pipelines become software artifacts requiring versioning, testing, lifecycle management, and governed change control.

These analysts knew the business. They understood the data. What they lacked was any orientation to the engineering disciplines required to manage complex systems.

They approached Business Intelligence as an extension of reporting. They had little exposure to SDLC concepts, version control, DevOps practices, formal modeling techniques, metadata management, governance structures, or testing theory. As a result, BI teams and the Centers of Excellence that followed became accidental software teams.

They inherited software development responsibilities without inheriting the frameworks needed to manage them.

At first, this worked remarkably well. With a small number of pioneers, analytics functions grew quickly. Early dashboards delivered immediate value. Executives were impressed. Innovation flourished. But growth eventually plateaued. Systems became owned by too few individuals. (Put in your ticket; get your report back in a week). Definitions fragmented. Dashboards multiplied. Logic diverged. Knowledge concentrated in silos. Technical debt accumulated across personal pipelines, duplicated metrics, and undocumented business rules.

Analytics became software faster than organizations adopted software engineering. The system became fragile. Consistency eroded. Trust declined. Scaling became difficult. The departure of a single analyst could destabilize entire reporting ecosystems.

The challenge has been reconciling self-service analytics with a central, disciplined approach. Much has been written about the fact that analytics needs governance. Even self-service. The question is how that discipline is operationalized. The following sections examine the failure modes of craft-based analytics and the engineering frameworks that address them. Later, we discuss how QSDA Pro is one solution that addresses these disciplines where analytics systems are most vulnerable.

The Core Failure

Initially, the disconnect between software-like processes and the lack of software discipline was thought to be a people problem. At its root, however, the failure is simple: **analytics is treated as a craft when it must be treated as engineering**. It's a process problem.

Craft Analytics

Craft analytics emerges naturally in early analytics programs. A small number of tech-savvy power users – some companies call these “citizen developers” – with strong domain knowledge, produce dashboards, metrics, and models through personal technique. Success depends on intuition, experience, and familiarity with the business rather than on shared process (McConnell, 2004).

In these environments:

- Business rules live in analysts’ heads rather than in documented specifications.
- Metric definitions evolve informally through conversation and repetition.
- Logic is embedded directly into dashboards instead of shared models.
- Testing is visual and manual: “Does this look right?”
- Changes are made quickly, with limited understanding of the downstream impact.

The outputs are often impressive. Dashboards look polished. Questions get answered. Executives see value quickly. Early success reinforces the belief that analytics is primarily a matter of individual skill rather than system design. This is the trap.

The Limits of Scalability

Craft works locally but fails structurally. Because the process lives in the individual, the organization inherits the individual’s limitations along with their strengths.

When analytics remains craft-based:

- Throughput is constrained by individual bandwidth.
- Knowledge concentrates in a small number of indispensable people.
- Logic cannot be reliably reused or extended.
- Consistency depends on who built the artifact.
- Staff turnover threatens continuity and trust.

Growth exposes the fault lines. As dashboards multiply and data sources change, teams spend more time reconciling numbers, fixing breakage, and explaining discrepancies than delivering insight. No two reports can be trusted to have the same logic. What once felt fast becomes brittle. What once was flexible becomes fragile.

This is not a failure of talent. It is the predictable outcome of relying on personal technique. **Analytics programs rarely fail because teams lack talent; they fail because organizations treat analytics as reporting rather than engineering** (McConnell, 2004).

Discipline Deferred

As fragility increases, organizations often respond by reframing the problem as a hiring challenge. Two strategies emerge.

1. The first is the **domain-expert model**: hire people who deeply understand the business and teach them analytics tooling. These individuals excel at interpretation and context but are rarely equipped to design resilient systems or manage lifecycle complexity.

2. The second is the **technologist model**: hire technically strong developers and teach them the business. These individuals bring structure and rigor but struggle to encode domain logic without constant translation.

Both approaches fail for the same reason. They attempt to solve a **systemic problem with individual capability**.

Talent scouts tried to find the elusive candidate who embodied domain fluency, analytics skills, and engineering expertise. It didn't matter if these individuals exist; building an analytics program around them is neither scalable, sustainable, nor smart. Mature systems do not depend on exceptional people; they rely on **ordinary people supported by dependable processes**.

Analytics fails at scale not because organizations lack the right people, but because they lack the right **engineering discipline**. That discipline leads directly to the next problem.

Technical Debt

The failure of craft-based analytics does not surface immediately. It's subtle because there are successes. The water builds up slowly behind the dam. The quiet accumulation of shortcuts, assumptions, and undocumented decisions. Over time, these decisions harden into a structural burden that analytics teams must carry forward. The dam bursts when the light of an upgrade or staffing change shines its light on it. Software engineering has a name for this burden: **technical debt**.

> In analytics, technical debt is not a metaphor. It is the operating condition.

The Face of Technical Debt

Pressman defines technical debt as the cost incurred when teams optimize early speed at the expense of long-term maintainability (Pressman & Maxim, 2020). Technical debt in analytics forms wherever speed substitutes for structure. The "git 'er done" mentality. It is rarely intentional. It emerges from reasonable local decisions made without a governing system.

Pressman defines technical debt as the cost incurred when teams optimize early speed at the expense of long-term maintainability (Pressman & Maxim, 2020). Technical debt in analytics forms wherever speed substitutes for structure. The “git ‘er done” mentality. It is rarely intentional. It emerges from reasonable local decisions made without a governing system.

Common manifestations include:

- Multiple, slightly different definitions of the same KPI across dashboards.
- Copied logic instead of a single shared source.
- Hard-coded business rules embedded directly in visualizations.
- Dashboards without versioning, lineage, or rollback paths.
- Manual regression testing performed ad hoc.
- Data transformations that exist only in personal pipelines.
- Proofs of concept that become permanent.
- Requirements? We don’t have time for that.

Individually, each shortcut seems harmless. Collectively, they form a complicated, interconnected web no one really understands. Changes in one place ripple unpredictably across the system. Teams respond by moving cautiously, adding manual checks, and avoiding refactoring altogether. Upgrades are postponed.

Debt is invisible until it becomes unavoidable.

Business Consequences

Technical debt is often framed as a technical concern. In analytics, its impact is business-facing.

Speed declines. Each new request takes longer because analysts must navigate fragile logic and fear breaking existing work. Delivery slows not because teams are inefficient, but because the system is unstable.

Trust erodes. Executives receive conflicting numbers from different dashboards. Meetings shift from decision-making to reconciliation. Confidence in analytics declines, even when the underlying data is sound.

Cost escalates. Analyst time is consumed by rework, manual testing, and troubleshooting. The organization pays more than once for the same insight because logic is not reused.

Risk increases. Upstream changes – schema updates, new data sources, revised definitions – cause unexpected breakage. In regulated environments, this creates audit exposure and compliance risk.

Talent becomes a bottleneck. A small number of individuals become indispensable because they alone understand how things work. Their availability, or lack thereof, becomes an operational risk.

These outcomes are the predictable effects of unmanaged technical debt in a system that has outgrown craft.

Velocity without Direction

In analytics, speed is not inherently beneficial. Velocity without direction ensures only that technical debt accumulates faster. Agile practices are often introduced to improve speed and responsiveness in analytics. When supported by an engineering discipline, Agile works. When adopted without it, Agile accelerates debt (McConnell, 2009).

In analytics environments lacking structure:

- Backlogs become holding areas for deferred quality and system-wide rework.
- Acceptance criteria focus on visuals, not correctness or impact. Do the colors match our branding?
- “Done” means delivered, not validated.
- Refactoring is postponed indefinitely because it feels risky. It actually *is* risky.
- Sprints optimize for deliverables rather than system health.

Every sprint that ships untested logic or undocumented changes adds compound interest to technical debt. Speed increases temporarily, but fragility increases exponentially.

This is not a failure of Agile. **Agile assumes the presence of engineering discipline** – testing, version control, automation, continuous integration, and shared standards. Without those foundations, Agile becomes a delivery shortcut rather than a delivery system. The conclusion is consistent across industries and decades:

When engineering is missing, speed multiplies debt.

As analytics becomes more central to operations, that debt becomes a business liability rather than a technical inconvenience. There is a cost to carrying debt on the books.

At this point, the pattern is clear. Craft-based analytics does not fail because teams lack intelligence or effort – it fails because the system itself cannot absorb growth. The grass-roots system cannot scale. Technical debt accumulates faster than it can be paid down, speed erodes trust, and agility amplifies fragility. These outcomes are predictable. They are the inevitable result of treating a software problem as a reporting exercise.

The only durable remedy is not better tools or better people, but **engineering discipline** – the same set of practices that transformed software development from a fragile craft into a reliable industrial capability.

Analytics Requires Engineering Discipline

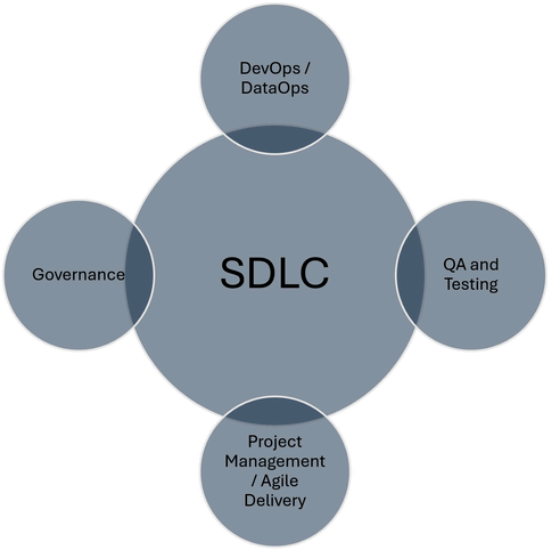
The failure modes described so far – scale, change, trust – do not stem from analytics being uniquely difficult. They stem from analytics being treated as something other than what it has become: **a software system undergoing continuous change**.

Sommerville emphasizes the point that **complex systems require defined processes, controlled change, and lifecycle discipline** (Sommerville, 2016). Analytics is a complex system. It requires the same response.

This approach deliberately avoids traditional analytics maturity models, which fail to reflect how capabilities evolve in practice. A deeper treatment of this argument is outside the scope of this paper.

Five Operational Disciplines

Modern software does not become reliable through any single practice. It becomes reliable when multiple disciplines reinforce one another across the lifecycle. Analytics is no different.



Five disciplines are consistently employed wherever complex systems are built and maintained:

1. **DevOps / DataOps** – Automation that reduces manual risk and shortens feedback loops
2. **QA & Testing Discipline** – Systematic validation that mistakes are caught early and on purpose
3. **Governance** – Defined ownership, lineage, standards, and accountability
4. **Project Management (Agile Delivery Discipline)** – Process and structured change
5. **Software Development Life Cycle (SDLC)** – The integrating framework that ties all stages together

Each discipline addresses a different failure mode introduced by craft. DevOps accelerates safe change. Testing enforces intent. Governance preserves consistency. Agile manages uncertainty. None of these, however, operates in isolation. They form a system. They require an organizing spine. That spine is the SDLC. Governance does not function as a competing discipline in this model; rather, it serves as a boundary condition that determines whether these disciplines—SDLC included—can operate coherently at the enterprise scale.

SDLC - The Anchor Discipline

Among the five disciplines, SDLC provides the structural backbone that connects intent to execution. It is sometimes misunderstood as a rigid, waterfall-era artifact. In practice, it is neither rigid nor outdated. SDLC is the organizing discipline that ensures complex systems can grow, change, and remain trustworthy over time. **It is the structure into which DevOps, testing, governance, and delivery practices naturally fit.**

At its core, the SDLC answers a simple question: *How does work move from idea to operation without collapsing under change?*

In analytics, the lifecycle is already present, informally:

- Questions are asked (requirements)
- Logic is designed (models and transformations)
- Dashboards and pipelines are built
- Results are checked – casually
- Content is deployed to users
- Breakage is discovered after the fact

When the SDLC is applied explicitly, it replaces improvisation with intention. Each stage exists to prevent a known class of failure:

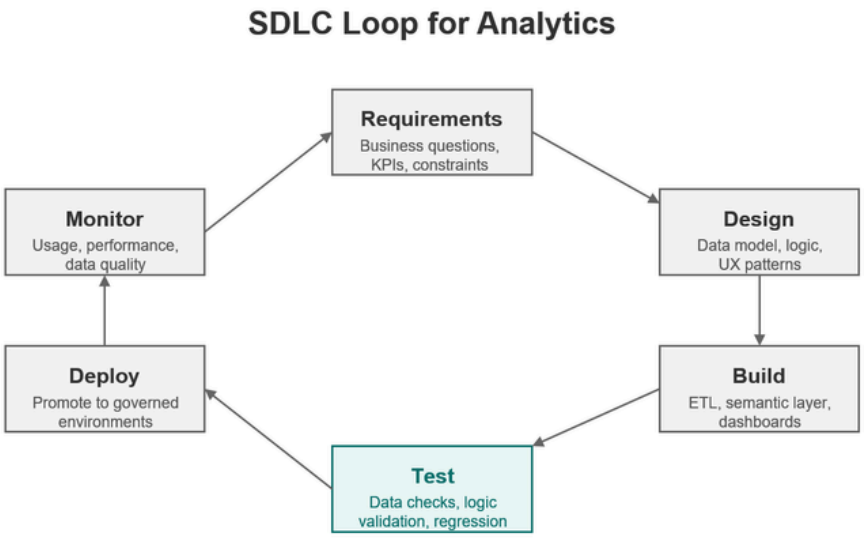
- **Requirements** reduce ambiguity and rework.
- **Design** prevents duplication and drift.
- **Build** enforces consistency and reuse.
- **Test** catches errors before users do.
- **Deploy** controls, change, and preserve lineage.

Monitor ensures systems remain trustworthy over time.

Without an SDLC, analytics teams may still perform these activities – but out of order, unevenly, and without feedback loops. The result is speed early, instability later.

With an SDLC, analytics becomes governable, testable, and scalable.

From SDLC to ADLC



The practical consequence of adopting the engineering discipline is an organizational-level operating-model change.

The success of craft-based analytics depends on exceptional individuals compensating for weak systems. Engineering-based analytics depends on systems: ordinary people producing reliable outcomes through a shared process.

This shift changes how success is achieved:

- From individual brilliance to institutional capability.
- From firefighting to controlled change.
- From fragile outputs to durable assets.

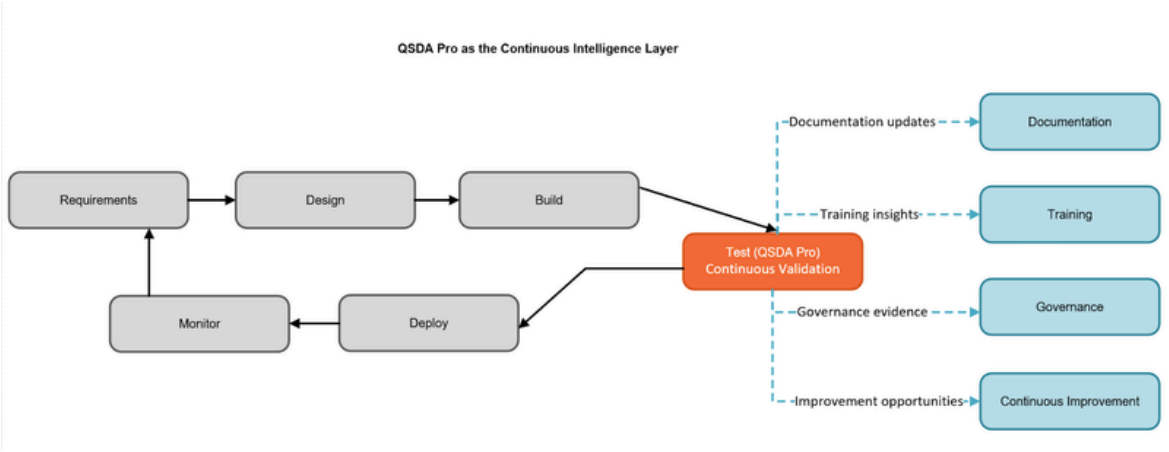
The applied structure does not slow analytics down. It makes speed sustainable.

The remainder of this paper examines how this shift works in practice – and where tools like **QSDA Pro** fit within it – not as replacements for discipline, but as enablers of it.

Where QSDA Pro Fits

Analytics engineering succeeds or fails at the same pressure points that challenge software systems: changes introduced during build, assumptions broken during testing, and drift that accumulates quietly in production. These are not conceptual failures; they are structural ones. QSDA Pro exists precisely at those points of highest risk.

Rather than redefining analytics practice, QSDA Pro strengthens it by supplying the automated structural validation that modern analytics systems require but rarely possess. It does not introduce a new discipline. It operationalizes an existing one.



QSDA Pro's Role in the Lifecycle

In an engineering-driven analytics lifecycle, quality cannot be added retroactively. It must be validated continuously, as changes occur. QSDA Pro enables this by embedding automated analysis directly into the stages where analytics artifacts are most vulnerable.

During Build, QSDA Pro provides immediate visibility into the internal structure of Qlik applications. Expressions, variables, data dependencies, and script components are analyzed systematically rather than reviewed informally. This enables developers and analytics engineers to identify broken logic, invalid references, and structural inconsistencies early, before they propagate downstream.

During Test, During the testing phase, QSDA Pro functions as an automated regression engine for Qlik applications. It executes checks against applications and reports errors, failures, or violations. It evaluates applications against known quality checks and reports newly detected errors or failures as analytics evolve. QSDA Pro is intended to provide early, objective signals when a change introduces risk. This replaces manual, error-prone comparison with objective evidence. Testing becomes a repeatable engineering control.

During Monitor, QSDA Pro extends quality beyond release. As applications evolve, upstream systems change, and business logic is refined, QSDA Pro detects drift – new inconsistencies, unused objects, and emerging structural risk. This transforms monitoring from reactive troubleshooting into continuous integrity assurance.

Across all three stages, QSDA Pro supplies something analytics teams have historically lacked: **machine-scale visibility into their own systems**.

Beyond Testing

Although QSDA Pro formally operates within the testing domain, its outputs extend well beyond quality control. By exposing the internal structure of analytics artifacts, it generates durable by-products that strengthen other engineering disciplines.

- **Governance support** through concrete evidence of definitions, dependencies, and change history
- **Living documentation** derived directly from real applications rather than manually maintained artifacts
- **Audit and risk visibility** through objective records of validation and structural analysis
- **Early technical debt detection**, allowing teams to address fragility before it becomes operational pain

These are not additional features layered on top of testing; they are natural consequences of making analytics systems observable.

Part of a Disciplined System

QSDA Pro provides structural validation and regression intelligence that enables these processes to operate reliably at scale.

This distinction is intentional. In mature engineering environments, no single tool carries the burden of correctness. Stability emerges from systems where people define intent and automation enforces integrity. QSDA Pro occupies that enforcement role for analytics – quietly, continuously, and without expanding its scope beyond what engineering discipline demands.

Final Thoughts

Analytics has already crossed the boundary from reporting to infrastructure. Once it becomes central to decision-making, it inherits the same requirements as any critical software system: accuracy, repeatability, controlled change, and trust.

The organizations that succeed are not those that build the most dashboards, but those that treat analytics as an engineered capability. They replace superheroes with systems, intuition with governance, and inspection with discipline. They recognize that analytics does not fail because people lack talent, but because systems lack structure.

QSDA Pro exists within this reality. It does not redefine analytics engineering, nor does it substitute for sound process. It strengthens the disciplines that make analytics reliable by providing automated structural insight where manual methods inevitably fail.

Analytics has already become software. The only remaining question is whether businesses will recognize it as such and adopt analytics engineering as a discipline.

Let’s talk DevOps for Qlik.

www.motio.com

+1 (972) 447 - 9595

Request a Demo

Motio®