Paper ET07

DevOps for Clinical Data Science: A Leap into the Future of Clinical Analysis

Frikkie Oppel, Bioforum, South Africa
Riaan van Straaten, BitCube, South Africa

ABSTRACT

Bioforum is modernizing its clinical programming environment by transitioning from traditional, folder-based workflows toward a dynamic Clinical Data Science Model built on Azure DevOps and Git. This transformation aims to strengthen governance, automation, traceability, and audit readiness across clinical analysis activities.

This paper outlines the objectives and operating model used in this transition, emphasizing computerized version control, structured branching strategies, automated CI/CD-based validation, comprehensive audit logging, and controlled management of blinded versus unblinded activities.

The approach is aligned with regulatory expectations, including 21 CFR Part 11 for electronic records and electronic signatures, ICH E6 (R2/R3) for Good Clinical Practice, and ALCOA+ data-integrity principles. The paper further discusses practical challenges encountered while adopting enterprise DevOps tooling to highly regulated statistical programming workflows and highlights solutions enabling scalable, compliant and reproducible execution.

Early implementation results indicate improvements in QC cycle speed and code hygiene through peer-review workflows. The paper concluded with planned enhancements, including broader automation, infrastructure-as-code, and advanced telemetry, supporting a continuously inspection-ready and future-proof clinical programming ecosystem.

DEFINITION - DevOps vs. Azure DevOps in the context of this paper:

DevOps: A software development methodology that aims to unify software development (Dev) and IT operations (IT) to improve collaboration, productivity, and efficiency. It emphasizes the integration of people, processes, and technology to deliver high-quality applications and services faster and more reliably.

Azure DevOps: Microsoft's cloud-based platform that provides tools to implement DevOps practices. The platform provides the option of integrated Git or Team Foundation (TFVC) Version Control.

INTRODUCTION

Modern clinical trials generate increasing volumes of data and require transparent, reproducible, and traceable analysis. Legacy, folder-based operating practices (for example, email-driven coordination, spreadsheet trackers, and shared network drives) create traceability gaps, increase status tracking and review effort, and complicate audit preparation. Across the industry, increasing data volume and regulatory expectations have exposed the limitation of folder-based, manual programming workflows. To address these issues, Bioforum initiated a structured transition to a DevOps-driven Clinical Data Science environment that embeds version control, automated validation, and continuous traceability into routine clinical programming work. The target state is a governed, automation-assisted operating environment that supports reproducible execution, controlled collaboration, and continuous inspection readiness.

THE CHALLENGE WE FACED

Prior to moving to a DevOps system, managing projects within clinical trial programming relied on manual processes and vehicles. Excel spreadsheets were used to track deliverables and serve as evidence of task completion. Communication happened mostly via Instant Messaging Apps and/or emails. Data and programs were stored on a file server network folder. These methods, though familiar, come with challenges: QC and documentation is a manual process; limited visibility into work status and dependencies; SAS code version confusion and risk of accidental overwrite; and restricted traceability between requests, code, and resulting outputs. These constraints become more severe as study scale and analytical complexity increase.

To address these issues, a DevOps system was considered. We set up a list of operational objectives with thorough consideration to regulatory requirements.

EVALUATING A DEVOPS SYSTEM

The objective was to achieve a security-and-quality-first transformation. The following operational objectives were set:

- Access Control and Security: repository- and branch-level controls; separation of blinded and unblinded activities; least-privilege access.
- Computerized Version Control (VCS): adoption and optimization of a VCS providing immutable history, branching, and peer-review workflows.
- Repository and Folder Architecture: standardized structures for code, configuration, metadata, and outputs.
- Azure DevOps Configuration: Boards for governed work tracking; Repos for source; Pipelines for automated validation; Artifacts for managed outputs.
- Continuous Integration/Continuous Development (CI/CD) Pipelines for QC: automated execution, log scanning, and policy-based gates prior to merge.
- Audit and Monitoring: organization-level audit logging, retention, and export for compliance monitoring.
- Documentation and Training: SOPs, WI, templates, and internal knowledge base.
- Business Outcomes Measurement: defined KPIs and ROI tracking (e.g., cycle time, rework rate, first-pass acceptance, onboarding time).

Thorough consideration was given to meet regulatory requirements:

- 21 CFR Part 11 criteria, under which electronic records and electronic signatures are considered trustworthy, reliable, and generally equivalent to paper records and handwritten signatures.
- ICH E6 Good Clinical Practice (current R2 with R3 principles), which establishes standards for trial conduct and data credibility; and
- Data-integrity expectations commonly summarized by ALCOA+, encompassing Attributable, Legible, Contemporaneous, Original, Accurate, Complete, Consistent, Enduring, and Available.

We started with a detailed project outline including milestones: Access Control framework, VCS optimization, CI/CD pipeline setup, configuration automation, performance monitoring, and knowledge base launch. Available systems and solutions in the industry were evaluated, and three possible solutions were identified for further assessment: Git, Team Foundation Version Control (TFVS) and Subversion (SVN).

To ensure infrastructure, regulatory, and operational requirements are met, stakeholder department leaders from Information Technology (IT), Quality Assurance (QA), and Biostatics and Statistical Programming were involved in the process from day one.

Access Control, Security, and the Computerized Version Control System

To assess what system will be most suitable for our requirements, all three systems were evaluated according to the following criteria: Collaboration model; DevOps integration; Access control granularity; and handling of large files.

Table 1 summarizes our findings:

Table 1: Evaluation of Computerized Version Control Systems:

Criterion	Computerized Version Control System		
	Git*	TFVC	SVN
Collaboration model	Distributed; lightweight branching; strong PR workflows	Centralized; good for large monoliths	Centralized; simpler mental model
DevOps integration	Deep CI/CD, Boards, Policies	Native to Azure DevOps (legacy)	Limited; external CI or custom
Access control granularity	Repo/branch-level (use separate repos for blinding)	Project-level; gated check-ins	Fine path-based permissions in one repo
Large files/binaries	Needs LFS/strategy; code-focused	Handles large repos well	Handles binaries better than Git
Adoption & ecosystem	Industry standard; rich tooling	Declining relevance vs Git	Stable but shrinking ecosystem

* Integrated in Azure DevOps.

Git in Azure DevOps allows distributed, resilient version control with robust branching and merging. It also includes deep integration with CI/CD pipelines, agile boards, and other automation tools, and has industry-standard features supporting peer review, audit trails, and regulatory compliance.

In pilot testing Git in Azure DevOps, an increase in speed and reliability was observed when running quality control cycles, compared to what was previously in place. We were able to create interactive dashboards to track status real-time, and peer reviews resulted in improved code hygiene whilst keeping an immutable audit trial.

TFVC was performing better when working with large, centralized codebases, but less suited for distributed, collaborative teams.

SVN offers simplicity and fine-grained access control but is inferior flexibility and DevOps integration capabilities needed for the modern, parallel workflows we had in mind.

Having evaluated the available systems, we selected Git in Azure DevOps due to its strong integration, auditability, and alignment with regulatory requirements.

OPERATIONALIZING AZURE DEVOPS AND GIT VERSION CONTROL

Once the system was selected, it was time to plan and implement the integration thereof.

It remains pivotal to tailor Azure DevOps configuration and internal Git workflows to our specific requirements. Strict separation between development and validated code is preferred as it strengthens independence in validating production work. Within the project repository (repo), the main branch holds only code that has passed QC and reviews—we call it the production codebase.

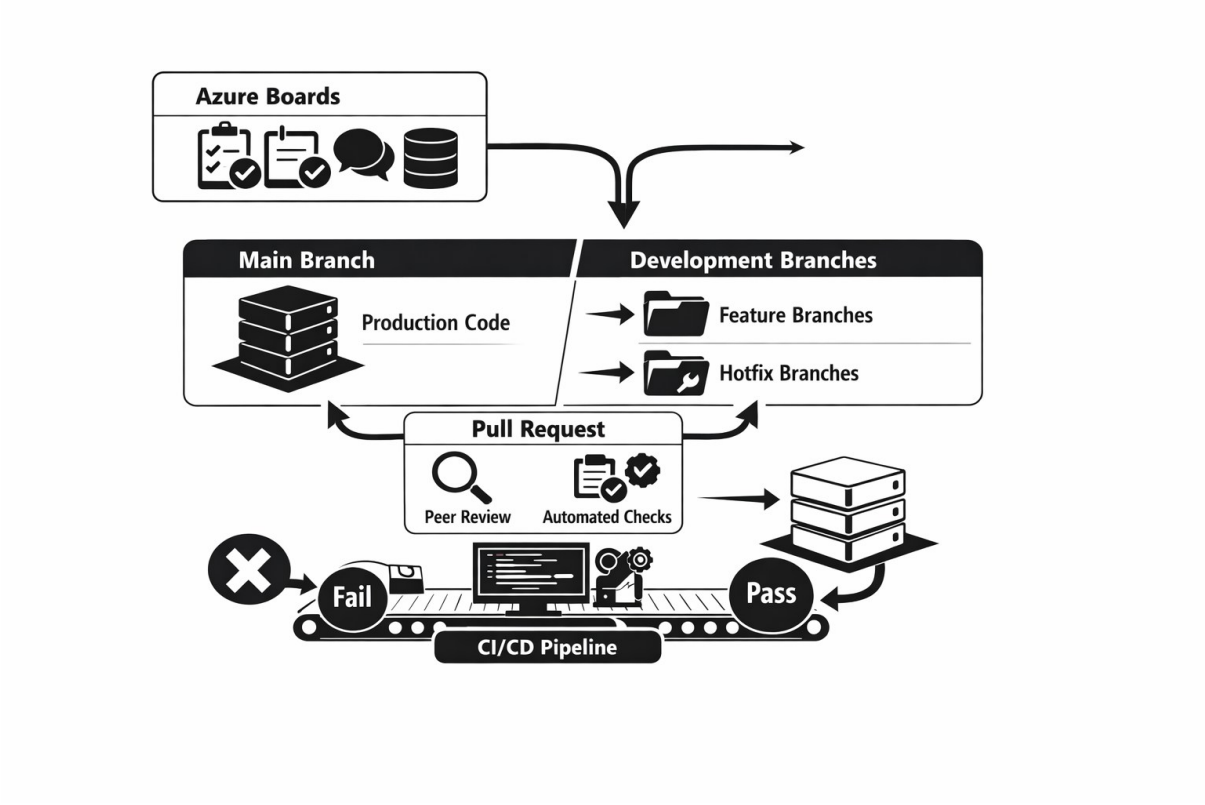
Developers work in separate feature branches, named for applicable tasks or tickets, allowing them freedom to experiment and develop without impacting others. For validation activities a dedicated QC branch is used for validation programming, from here the QC exchange is completed and communications are entered in Azure Boards, where it serves as an audit trail of updates, as well as a reliable way of tracking project progress at ground level.

Following validation, a Pull Request is issued for production code to be promoted (merged) to the Main branch. Merging requires the code to pass an additional set of automated checks, peer reviewer approval, and clean automated execution, embedding quality into the workflow.

The Git Workflow is configured to prevent destructive actions (for example, no Force-Push or History Rewriting), this ensures a tamper-proof record of changes. The result is a collaborative yet compliant branching model that keeps the validated codebase safe, auditable, and always deployment ready.

Figure 1 summarizes the implementation of the DevOps system.

Figure 1: Operationalizing Azure DevOps and Git



Azure Boards replace manual trackers. Every production and QC task is now represented by a visible work item, that enables real-time updates. Each commit and Pull Request references a Board ID, automatically linking code to requirements. The traceability is complete: Stakeholders can trace any output back to the exact code and documented request that prompted it. Reviews and signoffs occur directly on the work items. Audit readiness remains continuous as all activity, planning, coding, and reviewing is captured in Boards, Repos, and Pipelines.

Automated pipelines act as first-pass QC. Each Pull Request triggers a run that executes the code, checks logs, and validates compliance. The pipeline fails if any issue is detected, blocking merges until resolved. This automation ensures consistent quality and prevents errors from reaching production. It also archives logs and results automatically, creating a timestamped audit trail for every build.

Reproducibility is ensured through Git tags and pipeline artifacts. Each tag is a frozen snapshot of code used for a specific release. When code updates are required, a user creates a new branch created from the tagged commit, then applies the update under a new tag. This gives a clear chain of custody for every modification. Reruns are also tagged, allowing differentiation between runs using different data cuts. The environment information (for example, SAS or R package versions) is stored in pipelines and supports exact reproduction later, satisfying GxP and audit requirements.

In cases where separation of blinded and unblinded activities is required, the maintenance of separation benefits from the inherent secured isolation of splitting work into two repositories. The blinded repo is accessible to all programmers in the team but never contains unblinded data. When unblinding occurs, a new repo is cloned and restricted to authorized users. This controlled handoff maintains a clear audit trail of the transition. All work, post-unblinding, is tracked within that isolated repo. This ensures no contamination between blinded and unblinded environments and provides a transparent audit record.

Compliance and quality are enforced through system design. Branch policies prevent unreviewed code from entering main, and every action is logged under unique credentials. Azure DevOps audit logs capture who changed what and when. Git's immutable history acts as a signed, time-stamped record of all code changes. This automation means continuous audit readiness. By following our Git process, the team meets GxP and 21 CFR Part 11 requirements—without extra paperwork.

Azure DevOps accounts are set-up and set to applicable expectations as follows: (a) 21 CFR Part 11—identity-linked accounts, audit logs, electronic records governance, and validation of computerized systems; (b) ICH E6—documented procedures, adequate records, and audit trails; (c) ALCOA+—attributable activity through unique credentials; legible and enduring records via retained artifacts; contemporaneous, time-stamped event capture; original records preserved through immutable history and artifacts; accuracy supported by peer review and automated checks; completeness, consistency, and availability via linked work items, repos, and pipelines. Azure DevOps auditing at the organization level records administrative and policy events with definable retention and export.

Currently we don't have sufficient data to declare a roaring success story regarding business outcome measures, but early results are promising. The next phase of our implementation program will be focused on optimizing automation and workflow to increase efficiency.

FUTURE PLANS

Planned enhancements include broader use of infrastructure-as-code and containerized execution for consistent compute environments; policy-as-code to standardize repository protections; expanded analytics from audit logs and pipeline telemetry; and evaluated adoption of machine-assisted log review to prioritize QC findings.

CONCLUSION

Our objective was to modernize our approach to clinical programming by implementing a solution that will allow better control, better visibility of project status, and automation capabilities. The result is a DevOps-enabled operating model for clinical programming implemented in a manner consistent with regulatory expectations, by combining computerized version control, policy—based reviews, automated validation, and auditable traceability. The resulting framework supports reproducible analyses, controlled collaboration, and continuous inspection readiness. This transformation resolved our previous limitations of legacy practices and modernized our clinical programming approach for the benefit of our teams and clients.

REFERENCES (Selected)

- U.S. FDA. 21 CFR Part 11—Electronic Records; Electronic Signatures (official eCFR and FDA guidance).
- ICH E6 Good Clinical Practice: E6(R2) (FDA, 2018) and E6(R3) principles (EMA, 2025).
- MHRA. Guidance on GxP Data Integrity (2018, updated 2021).
- PIC/S PI 041-1. Good Practices for Data Management and Integrity (2021).
- Microsoft. Azure DevOps—Audit Logs (public preview) and Auditing Events list.
- Microsoft. Azure Repos—Branch Policies and Pull Request validation.
- Microsoft. Azure Boards—Linking work items to branches, commits, and pull requests.
- Microsoft. Azure Pipelines—Publish and download pipeline artifacts.
- Git SCM. Git documentation (branching, merging).
- GitHub Docs. About Git (distributed version control overview).
- ALCOA+ data-integrity summaries (industry references).

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Author Name: Frikkie Oppel
Company: Bioforum-The Data Masters
Address: 99 President Reitz Avenue, Bloemfontein, South Africa, 9301
Email: frikkie.oppel@bioforumgroup.com
Website: bioforumgroup.com

Author Name: Riaan van Straaten
Company: BitCube
Address: Novel Street, Bloemfontein, South Africa, 9301
Email: riaan@bitcube.tech
Website: bitcube.tech