BioForum THE DATA MASTERS

Turning Data Into Insights

DevOps for Clinical Data Science

A Leap into the Future of Clinical Analysis

Frikkie Oppel
Associate Director, Statistical Programming

Why change now?

We didn’t become programmers because we love admin!

“Ye ole” way of working was holding us back!

Enter DevOps for Clinical Data Science!

The Legacy State

- Manual QC and documentation
- Tricky Collaboration.
- Accidental overwrites or Version Confusion
- Limited “Audit Readiness”

The Plan & Priorities

It needed to be a security-and-quality-first transformation, not a tooling experiment. Priorities, in order:

1. Access Control & Security (mitigate unintentional/malicious access and unblinding).
2. Version Control System.
3. Folder Structure Review & Optimization.
4. Configuration & Batch Automation.
5. DevOps & CI/CD Pipelines for QC.
6. Performance Monitoring & VCS-based analytics.
7. Documentation, Training & Knowledge Base.
8. ROI tracking with milestones.

BioForum THE DATA MASTERS

Choosing a VCS: Git vs TFVC vs SVN

Criterion	Git (with Azure DevOps)	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)	Fine at project level; not file-path in Git sense	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

BioForum THE DATA MASTERS

Operationalizing Git + Azure DevOps
(how we actually make it work)

Branching Strategy and Pull Requests:

- **Tailored GitFlow model:** main (validated production code), develop (integration/staging), short-lived feature/bugfix branches; release branches for deliverables; hotfix branches for urgent fixes.

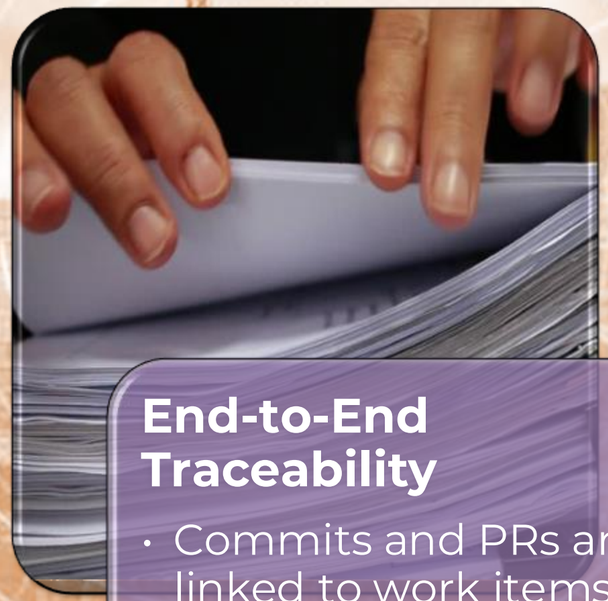
- **Branch Policies:** direct commits to protected branches are disallowed; all changes flow through Pull Requests; no force-push or history rewrite.

- **Peer review & status checks:** PRs require reviewer approvals and passing CI (automated build/QC) before merge; supports independent QC branch for double-programming.

Azure Boards for Traceability

Boards replace spreadsheets

- Tasks, requirements, bugs, and reviews live as work items with status, assignee, discussion, and attachments.



End-to-End Traceability

- Commits and PRs are linked to work items
- Every change references a specific task
- Work items show related builds/results

Integrated Approvals

- QC/validation steps are logged on work items and tied to PRs
- Dashboards surface progress and blockers in real time

BioForum THE DATA MASTERS

CI Pipelines for QC Automation

Continuous Integration (CI) — Pipelines as First-pass QC

- Automated Azure Pipelines run upon code changes
- Every PR/merge triggers pipelines to execute programs (batch SAS/R), run unit tests where applicable, lint code, and scan logs for errors/warnings.

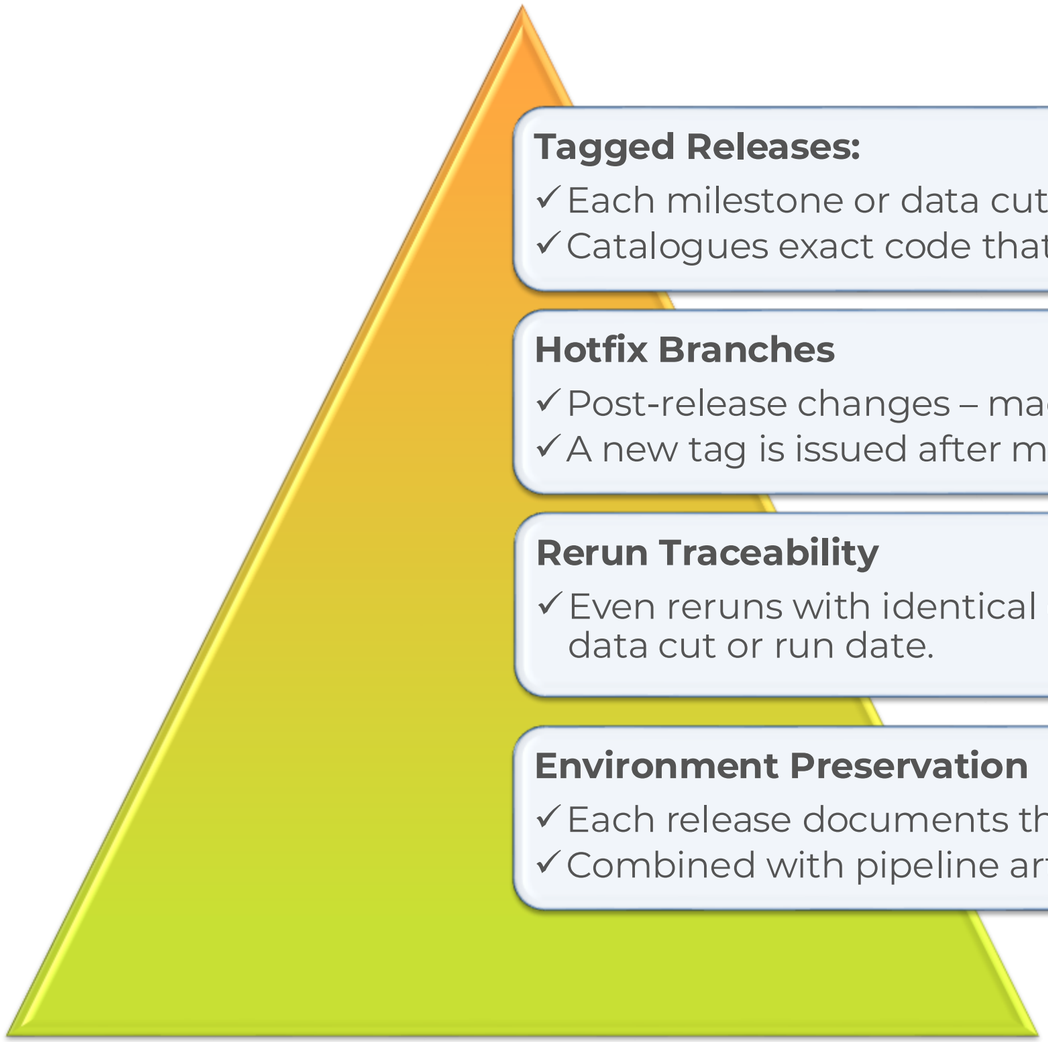
Quality Gates — Pipelines as QC gatekeepers

- When error occurs – pipeline failures block merges;
- compliance checks (e.g., Pinnacle 21, Define-XML validators) can run automatically.

Artifacts and Logs — Pipeline publishes run artifacts

- All compare outputs, validation reports are published and retained with the build as auditable evidence.

Reproducibility with Tags and Artifacts



Tagged Releases:

- ✓ Each milestone or data cut is tagged in Git (e.g. “2025-07-DBLock”).
- ✓ Catalogues exact code that generated results and enables 100% future reproducibility.

Hotfix Branches

- ✓ Post-release changes – made in hotfix branches created from tagged commit.
- ✓ A new tag is issued after merging (for example, v1.1), tracing differences between releases.

Rerun Traceability

- ✓ Even reruns with identical code are tagged separately and linked to artifacts – distinguishing results by data cut or run date.

Environment Preservation

- ✓ Each release documents the environment (for example, SAS or R package versions).
- ✓ Combined with pipeline artifacts, these form a complete electronic notebook of analyses.

BioForum THE DATA MASTERS

Blinded and Unblinded Workflows

Separate Repositories

- Separate Repos – maintain the treatment blind.
- The blinded repo never contains unblinded data.

Controlled Unblinding Handoff

- Blinded repo is cloned into a new unblinded repo – only unblinded team have access.
- Clone event records the unblinding point.

Post-Unblinding Changes

- Occur in the unblinded repo with full tracking of commits and reviews.
- The blinded team remains blind to these changes.

Ongoing Blinded Work

- Blinded repo remains in place for further blinded activities.
- Access controls ensure separation – all permission changes logged and approved.

BioForum THE DATA MASTERS

Compliance and Quality Controls

Branch Protections – All merges require Pull Requests:

- ✓ Reviewer approval + Passing CI checks.
- ✓ No direct pushes or history rewrites are allowed.

Audit Trails

- ✓ Azure DevOps logs all key actions at user level.
- ✓ PRs record reviewers and timestamps – required 21 CFR Part 11 audit evidence.

Git provides Immutable Records

- ✓ Tagged releases serve as read-only snapshots of validated code → audit-ready record of programming activity

Built-in Quality Culture

- ✓ Git and DevOps workflows achieves compliance naturally – accountability and traceability are automatic.
- ✓ Manual forms and trackers are no longer needed.

SOP modernization:

Embedding DevOps into Quality

Quality through DevOps

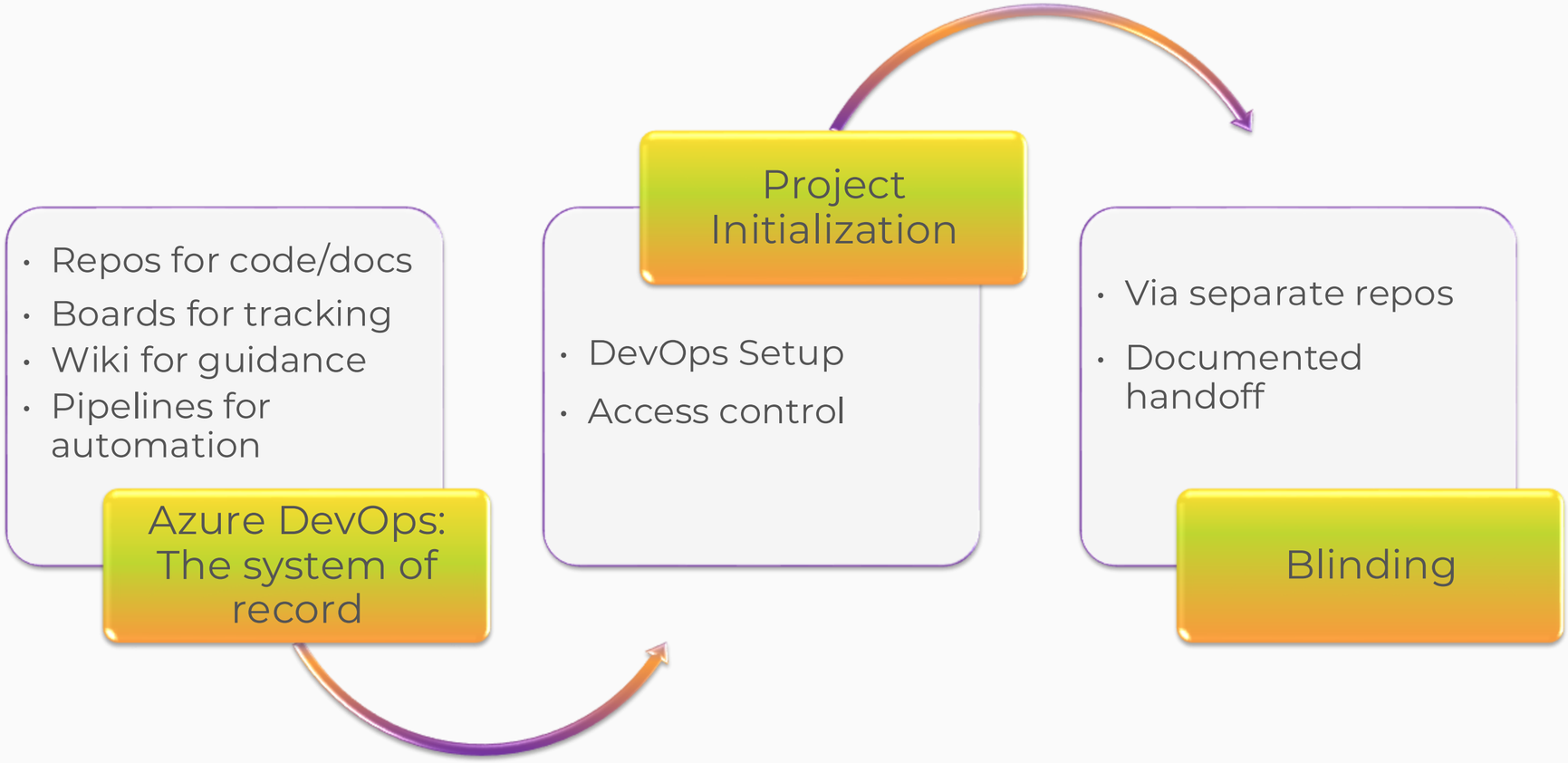
Independence enforced via branches (feature/ & qc/)

Pull Request approvals and pipeline checks before merge to main

QC evidence stored as artifacts.

Risk-based statistical review flagged in Boards

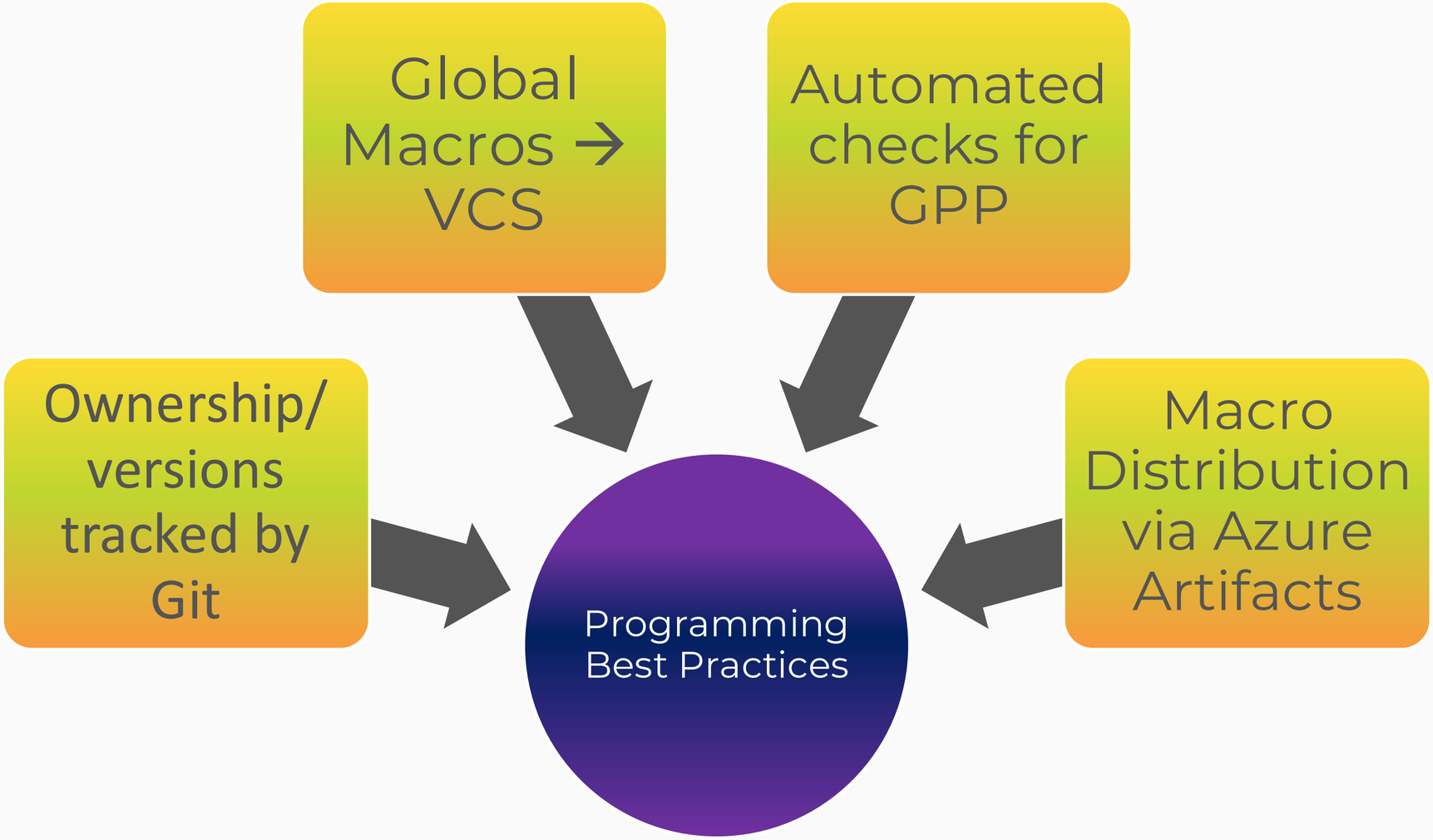
SOP modernization:
Embedding DevOps into Quality



Project Management

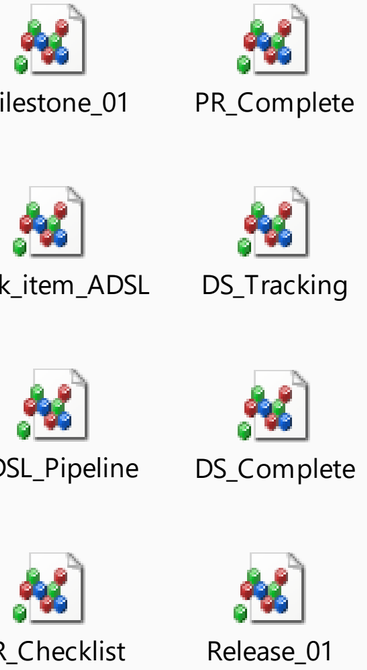
SOP modernization:

Embedding DevOps into Quality



Real-world:

1. ADaM area for next dry-run.
2. Work Item for ADSL
3. Pipeline for ADSL execute and syntax check
4. Pull Request review checklist.
5. Completed PR merge feedback
6. Dataset-level progress tracking
7. ADSL and ADLB fully completed
8. Formal Release



All Done:

Results & Lessons Learned

Early outcomes:

- Real-time oversight via Boards & dashboards
- Automated log scanning and validation artifacts
- Audit readiness becomes continuous
- Access control rework + repo segregation

Lessons learned:

- Start with security and branching policies
- Make training visual and contextual
- Keep repo structures consistent
- Define the unblinding handoff once and stick to it

What’s Next?

- Expand automation
- Performance monitoring from logs
- Infrastructure-as-code for analysis environments
- Deeper analytics on repos/Boards for training and workload optimization
- And much more

DevOps isn’t about moving faster at the expense of quality
It’s about building quality into the way we work. With Git and Azure DevOps, our clinical programming practice is more transparent, more collaborative, and inherently audit-ready. The destination is valuable, but the journey — the SOPs, the training, the culture — is what makes it sustainable.

BioForum THE DATA MASTERS

Q&A

What would be your first pilot?

Thank you.