Paper PD05

Leadership Strategies and Practical Frameworks for Statistical Programming in a Multilingual Era

Shaveta Bansal, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

In today's clinical and research environments, statistical programming spans multiple languages—e.g. SAS, R, Python, and SQL—within increasingly complex data ecosystems. Leading effectively in this multilingual era requires not only technical depth but also strategic governance and adaptive leadership. This paper introduces practical frameworks and skill-building strategies to help statistical programming leaders manage cross-language workflows while ensuring reproducibility, compliance, and innovation. Emphasizing continuous learning, collaboration, and responsible technology adoption, the paper outlines approaches such as establishing language-aware coding standards, creating language-neutral documentation, enforcing data integrity, and leveraging automation and AI under human oversight. Additional strategies include peer learning communities, structured development programs, and reusable assets that align with organizational and regulatory objectives. By applying these frameworks, leaders can build resilient, technically fluent teams that accelerate digital transformation and deliver consistent, auditable results for impactful clinical research.

INTRODUCTION

Statistical programming in clinical trials has grown beyond being just a technical task—it now plays a strategic role in shaping trial design, ensuring data integrity, meeting regulatory requirements, and ultimately improving patient outcomes. Today's programming environment is multilingual, with SAS, R, and Python often used together. This creates both opportunities and challenges for teams and leaders who need to maintain quality, consistency, and collaboration across different tools and workflows.

This paper provides practical guidance for managing these complexities while keeping processes traceable and results auditable. For brevity, let's focus on four key areas that are essential for successful implementation of multi lingual programming within an organization :

a) Code Standardization and Governance
b) Documentation and Communication (programming-language interoperability)
c) Data Integrity and Regulatory Compliance
d) Collaboration and Knowledge Sharing

Let's explore them one by one.

1) Code Standardization and Governance

Leaders can implement code standardization and governance by setting language-aware coding standards across SAS, R, and Python, enforcing reproducibility through deterministic settings and metadata, and using governance structures with automated checks to ensure compliance and quality at scale.

a. **Define Global, Language-Aware Standards**

Leaders can start by establishing organization-wide style & structure guides for SAS, R, Python - covering guidelines for - naming conventions, comments, modularization, error handling, logging. **Centralized metadata repositories** are a critical tool in establishing and maintaining global, language-aware standards. For effective and seamless programming integration across multiple languages it becomes absolutely critical to store information like dataset names and descriptions, variable attributes (type, length, derivation logic), and controlled terminology like CDISC codelists etc in a common place. To ensure interoperability, metadata should be stored in language-neutral formats such as YAML, JSON, or Excel, enabling seamless integration across SAS, R, and Python environments. Some of the global standards are explained briefly below -

Naming Conventions – Set standard rules for naming datasets, variables, functions, macros, and files. E.g. snake_case for naming variables; lower_case_with_underscores for function

Comments – Set clear guidelines for writing clear, meaningful comments. e.g. Header comments defining purpose, author, date, version; Inline comments to explain complex logic or assumptions.

Modularization - Breaking code into reusable modules/functions instead of long scripts. E.g Use SAS macros for repeated derivations; R functions for analysis steps; Python classes for workflow automation.

Logging – Establish define process for consistent logging of process steps, warnings, and errors for traceability. E.g. SAS: Use PROC PRINTTO or custom log macros; R/Python: Use logging libraries with standardized formats.

Define numeric tolerances – Programmers encounter small differences in numbers because different programming languages handle, for example, rounding differently. Similarly, set rules for **locale and encoding** (like UTF-8) so dates, decimals, and special characters look the same everywhere. These steps help avoid errors and make output consistent and easy to check for regulatory reviews.

Error Handling – Establish use of standard approach for detecting and managing errors.

Let's follow one scenario to cover standard approach for **error handling.**
Scenario: Calculate BMI for valid records and generate Log warnings for invalid data.

Standard approach for Error handling:

- **Guardrails first** (schema/dataset checks): Confirm the input exists and has the required fields before doing any calculations. Check whether the ADSL dataset exists and abort if missing.

- **Defensive validation**: Detect missing or invalid values (e.g., height_cm <= 0, missing weight_kg) and avoid computing BMI on bad data.

- **Safe computation**: Apply a reusable BMI routine only to valid records.

- **Observable behavior**: Emit warnings or errors to the log so reviewers can trace what happened and why.

Rule: bmi = weight_kg / (height_cm/100)^2
Key: USUBJID

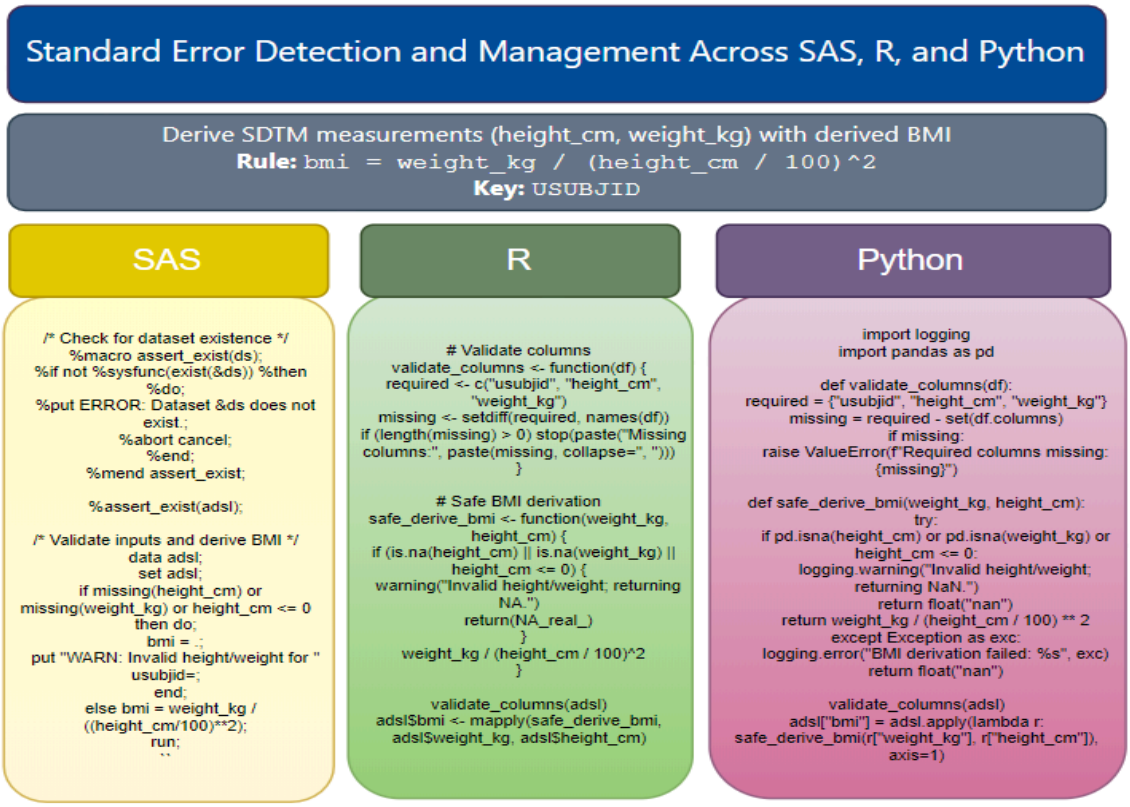


Figure: Above figure showcases side by side comparison on how standard approach can be used across three different programming languages.

b. Governance & Review at Scale

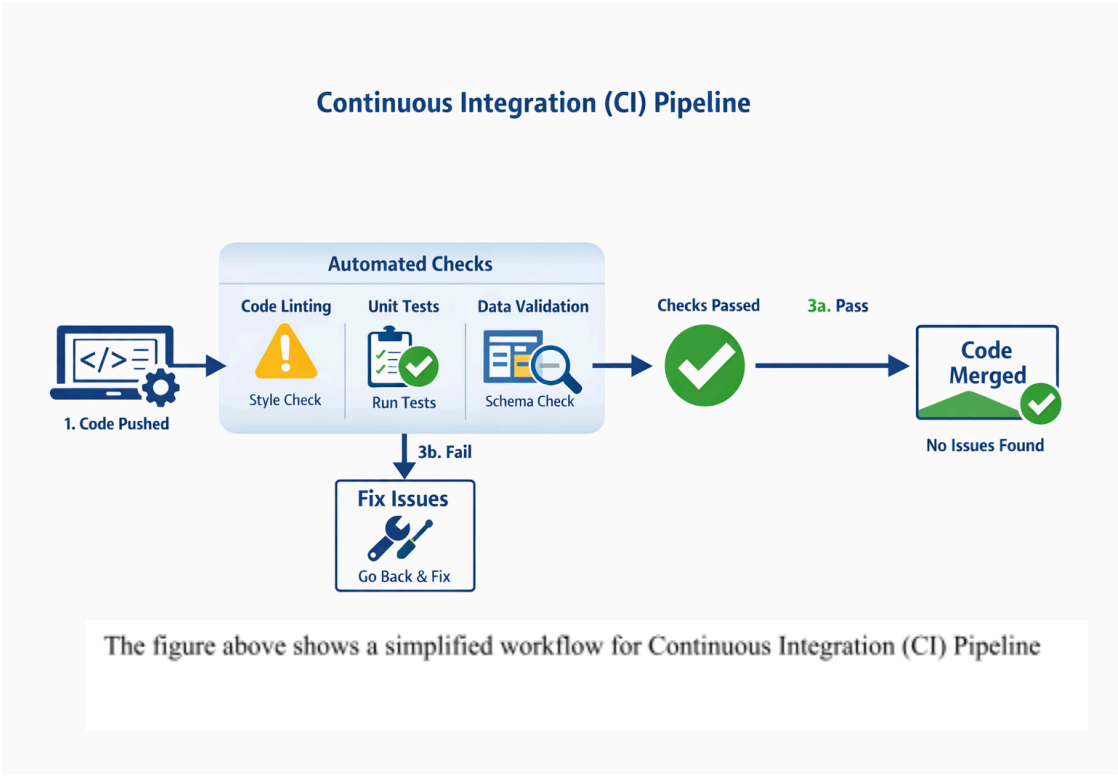
- Leaders can help stand up a **Multilingual Code Review Board** enrolling and engaging senior reviewers for each language to **set standards, review critical code, and ensure consistency and compliance** across all languages. This prevents language-specific silos and promotes best practices organization-wide.

- Leaders can champion use of **Architecture Decision Records (ADRs)** as standard practice. ADRs are concise documents that capture the reasoning behind key technical choices—such as deciding to use **SAS for ADaM build**, **R for model fitting**, or **Python for QC automation**. They help teams understand *why* a decision was made, what alternatives were considered, and under what context it remains valid.

  To encourage adoption, leaders can provide a short, standardized ADR template. ADRs should live in a central, version-controlled location (e.g., GitHub, SharePoint)
  This lowers the barrier to entry and ensures consistency across teams. Emphasize that ADRs should be quick to write and easy to understand, not long technical papers. Whether in design reviews, sprint planning, or cross-functional meetings, leaders can prompt teams to create or update ADRs whenever a major decision is made. This normalizes the process and ties ADRs into existing governance.

c. Automated Enforcement

Leaders create a culture where quality, consistency, and reproducibility are built into the workflow—not inspected after the fact. Leaders can ensure that automated enforcement is not optional by requiring CI (Continuous Integration) pipelines in platforms like GitHub or Azure DevOps.



The figure above shows a simplified workflow for Continuous Integration (CI) Pipeline

This practice helps to catch common mistakes early (e.g unused variables, inconsistent naming, bad indentation etc.) and prevents broken code, mismatched data structures, or incomplete documentation from entering the main branch — improving reliability and audit readiness.

Leaders can help the teams by providing starter templates, resourcing the teams to add tests and fix issues. Key is to start with critical paths and schema checks and expand over time.

d. Gen AI Governance

Multi-lingual programming implementations often rely on Gen AI to generate code across SAS, R, Python, and more—and even to speed up tasks like writing unit tests or populating CI pipeline steps. This makes strong Gen AI governance essential to ensure all AI-generated content is accurate, validated, and safe. Leaders should specify which AI tools and workflows the organization approves, such as a coding assistant that runs inside a secure environment.

For any AI-generated code, teams must run validation tests to make sure the output is correct and safe to use. Any code that affects submissions, regulatory deliverables, or scientific decisions must still be checked and approved by a human expert.

To ease the burden of implementing these standards, organizations can begin by focusing only

on the most critical workflows - before expanding to the full programming ecosystem. Providing teams with reusable templates, shared metadata structures, and prebuilt CI/automation examples can greatly reduce setup time and make adoption feel more manageable. Leaders can also help by freeing up expert reviewers and programmers to act as champions, offering targeted training across SAS, R, and Python, and ensuring teams have the time and capacity to adopt new practices without compromising ongoing delivery commitments.

2) Documentation and Communication (Programming-language interoperability)

Language-agnostic documentation means capturing programming logic in a way that is not tied to a particular language, while still working from the same shared instructions. Instead of defining derivations, dataset attributes, and TLF rules separately in each language, organizations write them once in neutral formats such as YAML or JSON. These files become a single source of truth that SAS, R, and Python can all interpret consistently, improving interoperability and reducing the risk of mismatched logic.

Let's take a simple example like calculating BMI—can be expressed in a standardized, language-agnostic format that any team can use, whether they write code in SAS, R, or Python. YAML file below demonstrates how core logic can be written once and then interpreted consistently across different programming languages.

```yaml
derivation:
  name: BMI
  description: Body Mass Index from weight_kg and height_cm
  formula: "weight_kg / ((height_cm / 100) ^ 2)"
  decimals: 1

inputs:
  required:
    - USUBJID
    - weight_kg
    - height_cm

rules:
  valid_row: "USUBJID not missing AND height_cm > 0 AND weight_kg > 0"

error_handling:
  invalid_height: "height_cm <= 0 OR missing(height_cm)"
  invalid_weight: "weight_kg <= 0 OR missing(weight_kg)"

outputs:
  bmi: BMI
  valid_flag: BMI_VALID
  reason: BMI_REASON

postconditions:
  bmi_nonnegative: "BMI >= 0"

audit:
  version: "1.0.0"
```

Figure above shows **YAML version** of the BMI specification

Polyglot authoring tools like Jupyter further help by allowing SAS, R, and Python code to appear together in one reproducible document, keeping logic, outputs, and explanations aligned. A consistent repository structure, combined with automated pipelines that rebuild documentation

whenever code changes, ensures specifications stay synchronized with implementation.

Nevertheless, implementing new, language-agnostic standards is often difficult because teams are still operating within long-established, and sometimes siloed processes—each language (SAS, R, Python) has its own habits, templates, and approval steps that were never designed to work together. Legacy specs are often stored in static Word or Excel documents, making them hard to update, version, or translate into modern workflows like YAML-driven automation. Existing review cycles and workflows can also be rigid, relying on manual checks that don't translate well into CI/CD or cross-language validation.

Leaders can help overcome these barriers by simplifying first steps — starting with a few high-value derivations and modernizing specs gradually rather than all at once. They can invest in training and designate "integration champions" who help teams transition away from legacy methods. Most importantly, leaders can ensure teams have protected time and resources to adopt new workflows.

3) Data Integrity and Regulatory Compliance

Multilingual programming can make compliance and auditability more challenging because different languages process dates, characters, precision, and metadata differently—which can lead to outputs that diverge subtly across SAS, R, and Python. Regulators expect complete traceability, reproducible results, and a clear record of how data was transformed, so inconsistent behavior across languages can create real submission risk.

Leaders must ensure strong compliance foundations by standardizing data rules (as discussed in first section of this paper) and requiring teams to store metadata and schemas in shared, version-controlled locations. IT engagement is crucial to automate audit trails that capture software versions, parameters, Git commit hashes, and run logs every time a code is executed, regardless of language or platform.

For compliance, leaders can partner with IT to embed CDISC validation and statistical method checks and quality into CI pipelines. Automated tools e.g Pinnacle 21, PROC COMPARE, pytest, or testthat, should run automatically on every pull request, with pipelines designed to block merges when compliance checks fail.

Leaders also rely on IT to implement access controls, electronic signatures, and audit logs. This includes restricting who can approve code, who can access production data, and how system changes are recorded.

Finally, leaders should promote 'gold-standard' comparative testing and dual programming for critical endpoints; dual programming for primary analyses. This practice builds confidence in multilingual results and reduces submission risk across the organization.

4) Collaboration and Knowledge Sharing

A powerful way for leaders to enhance collaboration is by designing structured learning pathways that work within their organization and expose programmers to languages outside their primary expertise. This could include job rotations creating "micro-apprenticeships," where staff spend a few weeks collaborating with a different language team to understand coding styles, validation approaches, and pipeline configuration.
Leaders can further reinforce knowledge sharing by creating incentives—such as recognition programs, career advancement pathways, or contribution credits—for individuals who document

reusable patterns, contribute to common libraries, or mentor peers across languages.

Leaders can also invest in building internal “innovation sandboxes”—safe, isolated environments where teams experiment with new tools, frameworks, or methods without the pressure of delivery timelines.

Establishing a culture of visibility is equally important. Leaders can promote open project workspaces where teams publish interim results, design choices, test outcomes, and retrospectives, allowing others to learn continuously. Hosting quarterly internal meetups, multilingual hackathons, and learning symposia encourages cross-team idea exchange and sparks new collaborations.

Additionally, leaders can strengthen connections beyond their own company by engaging with cross-pharma communities and inviting external experts to share emerging best practices.

By setting expectations, investing in shared resources, and modeling collaborative behaviors themselves, leaders create an ecosystem where multilingual teams not only work efficiently but also grow collectively—ultimately increasing quality, reducing duplication, and accelerating scientific delivery.

Challenges in Implementing These Approaches—and How Leaders Can Help

Although the strategies in this paper create a strong foundation for multilingual programming excellence, implementing them in a real organization is rarely straightforward. The complexity stems not from lack of intent, but from the reality that no single person, team, or function owns the entire ecosystem. As a result, even well-designed frameworks can struggle to gain traction without coordinated leadership support.

One of the biggest challenges is **competing priorities.** Programmers and study teams often operate under tight delivery timelines, leaving little capacity to adopt new coding standards, convert specifications to neutral formats, or contribute reusable assets. Transformation work can feel like “extra” work when milestones are pressing.

Leaders can help by protecting dedicated time for modernization efforts, adjusting workload expectations, and recognizing teams for long-term improvements, not just short-term output.

A second challenge is **fragmented ownership** across functions such as Statistical Programming, Data Management, Biostatistics, and IT. Metadata repositories, CI pipelines, ADR governance, and AI governance each sit with different owners, and coordinating changes across these groups can be slow.

Leaders can mitigate this by establishing cross-functional steering groups with clear decision-making rights, jointly funded roles, and shared accountability for multilingual standards.

Legacy systems and **processes** pose another major barrier. Many teams rely on long-established templates, Word specifications, or SAS-only workflows. Even when people see value in new methods, shifting ingrained behaviors is difficult—especially across large global teams.

Leaders can address this by starting small: modernizing a few high-impact workflows, introducing language-neutral specs gradually, and providing starter kits that reduce the cognitive load of switching to new practices.

Another challenge that can be seen in teams is **skill gap**. Not every SAS programmer is familiar with Git, YAML, R validation tools, or automated testing frameworks. Meanwhile, R and Python programmers may not fully understand regulatory expectations encoded in SAS conventions. Leaders can accelerate capability building by investing in structured multilingual training programs, pairing experts across languages, and creating micro-apprenticeships that give people hands-on experience in new environments.

Finally, sustaining momentum can be difficult because improvements are distributed across many contributors - none of whom can drive transformation alone. Leaders can help by creating incentive structures that reward contributions to shared assets, providing recognition pathways for “language champions,” and ensuring that maintainers receive organized support rather than relying on volunteer energy.

Across all these challenges, the role of leadership is not to do the work themselves, but to shape the environment so that consistent, collaborative, multilingual programming becomes the easiest and most natural way of working. By protecting capacity, clarifying ownership, investing in capability, and reinforcing the value of shared standards, leaders ensure the frameworks in this paper can be implemented sustainably - creating resilient, interoperable programming ecosystems that deliver high-quality, auditable clinical results.

REFERENCES

- Nygard, M. (2011). Documenting Architecture Decisions.
  https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions
- ADR GitHub Organization. (n.d.). Architectural Decision Records (ADRs): Motivation, templates, tools.
  https://adr.github.io/
- SAS. (n.d.). PROC COMPARE Documentation (SAS Help Center).
  https://documentation.sas.com/doc/en/pgmsascdc/v_067/proc/n0c1y14wyd3u7yn1dmfcpaejllsn.htm
- pytest. (n.d.). *How* to use fixtures (official docs).
  https://docs.pytest.org/en/7.1.x/how-to/fixtures.html
- Wickham, H. (n.d.). *testthat—Unit Testing for R (official site).*
  https://testthat.r-lib.org/
- Greenlight Guru. (2025). CSV vs. CSA—What the FDA shift means.
  https://www.greenlight.guru/blog/csa-vs-csv

ACKNOWLEDGMENTS

Author would like to acknowledge AI for simplifying the research for this paper and Merck leadership team for enabling adoption of multi-lingual strategies and frameworks.

CONTACT INFORMATION

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

Shaveta Bansal
Merck & Co., Inc.
126 E. Lincoln Ave
Rahway, NJ 07065 USA (732)594-3766
Email:shaveta.bansal1@merck.com