Paper ET06

The AI Reckoning: Why Statistical Programmers Must Evolve Now

Bhavin Busa, Clymb Clinical, Burlington, MA, United States

ABSTRACT

SAS and R programming are not going away; they will remain foundational to Statistical Programming. Yet the function is rapidly evolving as Artificial Intelligence (AI) and commercially off-the-shelf (COTS) tools reshape daily workflows. The future demands programmers who can integrate Large Language Models (LLMs) into standard-driven solutions, ensuring compliance, consistency, and scalability.

CDISC standards play a pivotal role in this transformation. Models such as the Unified Study Design Model (USDM) and the Analysis Results Standard (ARS) are enabling AI-driven applications that automate mapping specifications for SDTM/ADaM and accelerate TFL development. LLMs will further advance this shift by supporting metaprogramming and assisting in the generation of SAS and R code seamlessly integrated into existing workflows.

This presentation will explore how SAS, R, and AI can co-exist in a model-based ecosystem, the role of CDISC in enabling innovation, and the challenges Statistical Programming teams must overcome to adopt this shift.

INTRODUCTION

Statistical Programming has evolved through multiple technology waves, from manual SAS and R programming to macro-based reuse, and to widespread adoption of CDISC standards. Despite these advances, the discipline has remained largely code-centric: specifications are interpreted manually, logic is repeatedly implemented, and validation occurs late in the lifecycle.

Artificial Intelligence (AI), particularly Large Language Models (LLMs), represents a fundamentally different inflection point. Unlike prior productivity tools, LLMs introduce the ability to reason over structured inputs, generate executable code, and assist across workflows. This shift challenges long-standing assumptions about how statistical programming effort scales with study complexity.

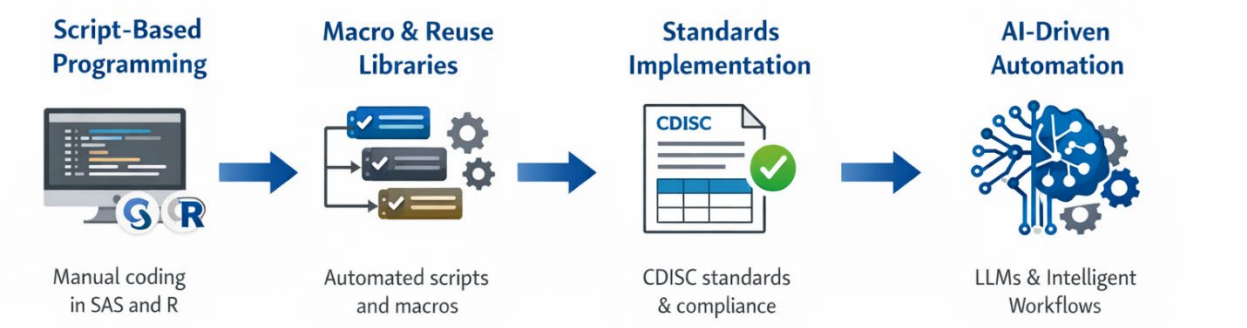

Figure 1: Evolution of Statistical Programming illustrating progression, from script-based programming to AI-augmented, model-centric workflows.

Statistical Programming is at an AI reckoning

Teams that apply AI in fragmented, non-unified environments, where specifications are unstructured, controls are inconsistent, and organization-level prompt strategies and governance are absent, risk amplifying inconsistency, reducing traceability, and eroding confidence in results.

In contrast, organizations that take the opposite approach are positioned to win. These organizations implement AI within unified, well-controlled environments built on structured specifications (as inputs), CDISC standards, for example, the Unified Study Design Model (USDM) and the Analysis Results Standard (ARS), and established organizational assets such as validated reference code, macro libraries, and controlled terminology. Consistency is enforced through system-level, organization-wide prompt

frameworks rather than ad hoc user prompts, with structured input metadata enabling traceability across specifications, code, and results. Crucially, statistical programmers remain actively involved through human-in-the-loop review, ensuring scientific judgment, regulatory confidence, and accountability are preserved as automation scales.

Figure 2: Key Pillars of the AI-Enabled Statistical Programming Framework. The six blocks represent the major themes addressed in this paper, spanning technical, architectural, organizational, and role-based transformation.

1. FROM CODE-CENTRIC TO MODEL-CENTRIC PROGRAMMING

Traditional code-centric workflows rely on manual translation of specifications into programs. Changes propagate downstream, creating rework across code, outputs, and reviewing artifacts. Scaling this approach becomes increasingly difficult as studies grow in complexity.

Model-centric programming inverts this paradigm by emphasizing machine-readable, execution-ready specifications as first-class inputs [1,8]. Analysis intent is captured once using structured, standard-aligned models, providing the precision required for reliable downstream automation and effective use of LLMs. Code becomes a generated artifact rather than a handcrafted deliverable, driven directly from these models. Validation shifts upstream toward specification quality and downstream toward consistency across metadata, code, and results. While legacy CDISC dataset models such as SDTM and ADaM remain valuable foundations, newly available standards such as USDM and ARS significantly amplify the value of structured inputs by explicitly modeling study design and analysis intent, enabling LLMs to operate on well-defined, machine-interpretable context rather than ambiguous free text.

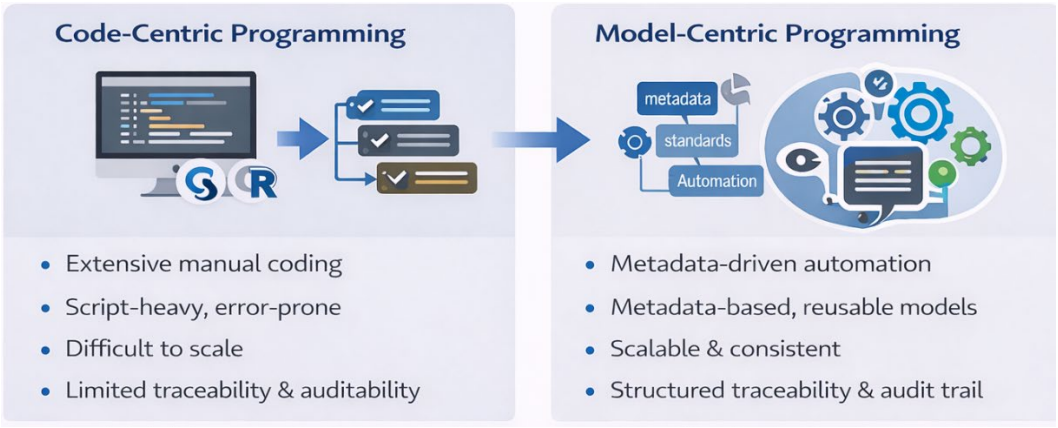

Figure 3: Code-Centric vs. Model-Centric Statistical Programming

2. CDISC AS THE ENABLER, NOT THE CONSTRAINT

CDISC standards are often viewed as compliance requirements rather than innovation enablers. Recent initiatives such as CDISC 360 and 360i reposition standards as the backbone of end-to-end automation across the clinical trial lifecycle [4,5].

USDM provides a machine-readable representation of study design, while ARS formalizes how analyses are defined, executed, and reported [6,7,11]. ARS links datasets, methods, and results within a traceable logical model.

This structured foundation is essential for AI because LLM performance is directly dependent on the quality, structure, and machine-readability of its inputs. LLMs operating on free text or loosely defined specifications introduce variability and ambiguity. In contrast, machine-readable, execution-ready specifications expressed through structured models provide the precise context required for consistent and traceable outcomes. While legacy CDISC dataset models such as SDTM and ADaM remain important building blocks, newer standards such as USDM and ARS explicitly model study design and analysis intent, significantly amplifying the value of structured inputs. When LLMs are constrained by these models and standardized JSON representations, AI outputs become reproducible, auditable, and suitable for regulated environments [1,3,9,10].

3. WHERE LLMS FIT, AND WHERE THEY DO NOT

Large Language Models are most effective when treated as components within a governed system rather than as standalone problem solvers. Their strengths lie in synthesizing patterns from structured context, accelerating repetitive tasks, and translating well-defined intent into executable artifacts. When deployed within a controlled, standards-driven environment, LLMs can significantly reduce manual effort while improving consistency across studies and deliverables.

In Statistical Programming, this means LLMs should operate on machine-readable specifications, structured metadata, and predefined organizational knowledge bases. When analysis intent is clearly defined through models such as ARS, and when inputs are constrained through standardized JSON structures, LLMs can reliably generate SAS and R code, assist with metaprogramming, and support consistency checks across outputs. In this role, LLMs function as force multipliers for established workflows rather than sources of independent analytical judgment.

However, the limitations of LLMs become evident when these conditions are not met. Deploying LLMs in loosely governed environments, where inputs vary by user, prompts are improvised, and specifications are primarily free text, introduces variability that undermines traceability and reproducibility. In such settings, LLMs may generate superficially plausible outputs that are difficult to validate, reconcile, or defend in regulated contexts. This risk is compounded when LLMs are expected to infer analysis intent or make statistical decisions without explicit modeling and human oversight.

LLMs excel at pattern recognition, code synthesis, and contextual assistance. Their value in Statistical Programming lies in augmentation rather than autonomy.

Appropriate use cases include:

- Generating SAS and R code from ARS-aligned metadata
- Supporting metaprogramming and boilerplate logic
- Assisting with mapping specifications and consistency checks

Poor use cases include:

- Autonomous definition of analysis intent
- High-stakes statistical decision-making
- Unconstrained interpretation of free-text specifications

4. PRACTICAL ARCHITECTURE FOR SAS, R, AND AI CO-EXISTENCE

A sustainable AI-enabled architecture is layered rather than disruptive, and it must feel familiar to the people who run studies. In practice, this means providing a user interface that mirrors today's workflow (specification → programming → QC/review → outputs) while introducing an enterprise platform underneath that standardizes inputs, controls, and traceability across the organization.

A practical reference architecture includes:

- **Experience Layer (Workflow UI)**: A guided interface for building and reviewing specifications, selecting standard options, attaching supporting context, and tracking comments/approvals. The User Interface (UI) should mimic existing ways of working so adoption is natural, while capturing decisions in a structured, machine-readable form.
- **Specification & Model Layer (Structured Inputs)**: A machine-readable, execution-ready specification model (not limited to any single standard) that represents analysis intent, populations, endpoints, derivations, and output requirements. This layer produces governed inputs, often expressed as structured metadata (e.g., JSON), that are stable enough to drive automation.
- **Knowledge & Standards Layer (Enterprise Assets)**: A curated library of organizational artifacts such as reference implementations, validated macro/function libraries, code templates, and controlled terminology. These assets reduce variability and provide guardrails for both humans and AI.
- **AI Orchestration Layer (System-Level Prompts & Policies)**: Centralized prompt templates, policies, and routing that enforce consistency across users (rather than ad hoc prompts). This layer determines what context is provided to the LLM (specifications, relevant reference code patterns, standards rules) and applies controls such as restricted actions and logging.
- **Execution Layer (SAS/R Runtime)**: SAS and R engines execute generated or templated programs within validated environments. Execution remains deterministic; AI accelerates authoring and assembly, not the underlying validated computation.
- **Quality, Traceability & Audit Layer (Human-in-the-Loop)**: Integrated review, QC, and approval workflows where programmers remain accountable. Every generated artifact is traceable back to the structured specification and the enterprise assets used, with versioning, logs, and QC checks.

Our team is presenting a complementary paper that describes an AI-assisted code generation capability implemented within such an enterprise platform [2]. While the specific implementation details are out of scope for this paper, the example demonstrates how LLMs can be used to generate SAS and R programs when driven by structured, machine-readable specifications, governed enterprise knowledge assets, and system-level prompt orchestration. Importantly, the solution reinforces that AI operates as an assistive layer, augmenting existing programming ecosystems, while keeping statistical programmers in the loop and preserving traceability, reviewability, and regulatory confidence.

5. ORGANIZATIONAL AND SKILL CHALLENGES

AI-enabled, model-centric workflows introduce challenges that are as much organizational as they are technical [1,3]. Successfully operationalizing AI in Statistical Programming requires deliberate changes across skills, processes, and governance models.

Key organizational challenges include:

- **Shift in Skill Profiles**: Statistical Programmers must expand beyond language syntax and tool proficiency to include metadata modeling, interpretation of structured specifications, and understanding how AI systems consume inputs. Familiarity with machine-readable models, JSON-based specifications, and automation frameworks becomes essential.
- **Governance and Control Models**: Organizations must define clear governance around how AI is used, including who can trigger code generation, how prompts are standardized at the system level, and how outputs are reviewed and approved. Ad hoc or user-specific AI usage undermines consistency and increases risk.
- **Validation and Quality Frameworks**: Existing validation approaches, which often focus on hand-

written programs, must evolve to accommodate generated code. Emphasis shifts toward validating specification models, reference code libraries, and the deterministic execution environment rather than each individual program instance.

- **Change Management and Adoption**: Introducing AI-enabled platforms without aligning them to existing workflows can create resistance. User interfaces and processes must closely mirror current ways of working while quietly enforcing structure, standards, and traceability behind the scenes.
- **Accountability and Human Oversight**: Despite increased automation, accountability cannot be delegated to AI. Statistical programmers and reviewers remain responsible for ensuring scientific correctness, regulatory compliance, and interpretability of results.

When these challenges are addressed holistically, AI-enabled automation strengthens rather than weakens control. Structured specifications, standardized enterprise assets, and human-in-the-loop review create more transparent, auditable, and scalable workflows compared to traditional, code-centric approaches.

6. THE EVOLVING ROLE OF THE STATISTICAL PROGRAMMER

As AI becomes embedded in Statistical Programming workflows, the role of the Statistical Programmer evolves from primarily writing code to designing, governing, and validating end-to-end analytical workflows. This shift does not diminish the importance of programming expertise; instead, it broadens the scope of responsibility and influence.

Key aspects of this evolving role include:

- **Specification Ownership**: Actively shaping clear, machine-readable, execution-ready specifications that accurately capture analysis intent and serve as reliable inputs to automation and AI systems.
- **Model and Metadata Stewardship**: Understanding and working with structured models and metadata frameworks, ensuring consistency between specifications, generated code, and reported results.
- **Enterprise Standards Alignment**: Applying and extending organizational standards, reference code, macros, and analysis patterns so that generated outputs remain consistent across studies and teams.
- **AI-Guided Programming Oversight**: Reviewing, validating, and refining AI-generated SAS and R code, focusing on correctness, interpretability, and adherence to statistical and regulatory expectations.
- **Quality and Traceability Accountability**: Ensuring that every output remains traceable to its originating specification, inputs, and reference assets.
- **Cross-Functional Collaboration**: Working more closely with statisticians, data standards teams, platform engineers, and QA groups to design workflows that are both scientifically sound and operationally scalable.

In this model, statistical programmers remain central to clinical reporting. Their expertise shifts from repetitive implementation to higher-value activities, providing judgment, context, and governance, ensuring that AI-augmented workflows remain accurate, compliant, and trustworthy.

CONCLUSION: ADAPT NOW OR BE DESIGNED AROUND LATER

The AI reckoning in Statistical Programming is already underway. Organizations that delay adaptation risk being constrained by tools and workflows designed without their input. Those that move forward deliberately, by combining structured, machine-readable specifications, enterprise controls, and human oversight, are positioned to lead rather than react.

Crucially, success will not come from relying on a single tool or technology. SAS and R remain essential execution engines, but they must be complemented by purpose-built platforms and solutions, whether built internally or licensed, that address different types of deliverables across the clinical lifecycle. No single solution can optimally support all specifications, analyses, outputs, and review workflows. Instead, a multi-solution ecosystem, unified through common models, interfaces, and governance, enables flexibility without fragmentation.

In this model, AI serves as an accelerant rather than a replacement, using structured specifications, reference code, and system-level controls to generate consistent, traceable outputs. Organizations that invest in the right combination of languages, platforms, and governance frameworks will be best positioned to scale automation responsibly while preserving scientific rigor and regulatory confidence.

REFERENCES

1. Bosman Malan, Busa Bhavin. TFL Automation Through Design, Specify, Execute, and Report: A Seamless End-to-End Framework. PHUSE APAC Connect, February 2026.
2. Dedhia Navin, Busa Bhavin. AI Code Generator: Automating TFL Programming with ARS Metadata. PHUSE US Connect, March 2026.
3. Bosman, Malan. Automated Code Generation - Evaluating Deterministic and Probabilistic Approaches. PHUSE US Connect, March 2026.
4. CDISC 360 Project White Paper, June 2021. https://www.cdisc.org/sites/default/files/2021-06/CDISC_360_Project_White_Paper.pdf
5. CDISC 360i Program Kickoff Presentation. Enabling Standards-Driven Automation from Study Design Through Reporting, February 2025. https://www.cdisc.org/sites/default/files/pdf/360i-Program-Kickoff20250218.pdf
6. CDISC Analysis Results Standard (ARS), Version 1.0, April 2024. https://cdisc-org.github.io/analysis-results-standard/
7. CDISC Analysis Results Standard User Guide, Version 1.0. https://wiki.cdisc.org/display/ARSP/Analysis+Results+Standard+User+Guide+v1.0
8. Busa, Bhavin.; Marshall, Richard.; LeRoy, Bess. '*All You Need to Know about the New CDISC Analysis Results Standards!*' PharmaSUG, May 2023.
9. Bosman, Malan. '*Advancing TFL Automation: R-Based Meta-Programming Powered by ARS*' PHUSE US Connect, March 2025.
10. Wachara, Julie. '*Let the Machine Do the Coding: A SAS-Based Framework for Automating TFL Generation with ARS Metadata*' PHUSE US Connect, March 2025.
11. CDISC. Unified Study Definitions Model (USDM) v2.0. 2023.

CONTACT INFORMATION

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

Bhavin Busa
Principal & Co-founder,
Clymb Clinical
bhavin@clymbclinical.com