connect·share·advance

From Programmer to AI-Augmented Programmer

The Evolution of Clinical Programming in the Age of Generative AI

Sivakumar G | Advisor – Clinical Standards

Global Statistical Sciences, Eli Lilly and Company

phuse.global

What We'll Cover Today

01

The Shift Underway

From a linear, code-heavy model to AI-augmented programming

02

Copilot vs. Autopilot

A framework for reasoning about AI autonomy in a regulated setting

03

Prompt Engineering

A new technical discipline for clinical programmers

04

Opportunities

SDTM, ADaM, TLF, and documentation workflows

05

Governance & Accountability

Data privacy, unblinding risk, GxP, and human ownership

06

Skills & the Road to 2030

What the future clinical programmer looks like

Clinical Programming Is Under Pressure to Change

Trials are more complex than ever — multiple data sources, global sites, adaptive designs, decentralised components, wearables, and real-world data. Yet a large share of programming effort remains highly repetitive.

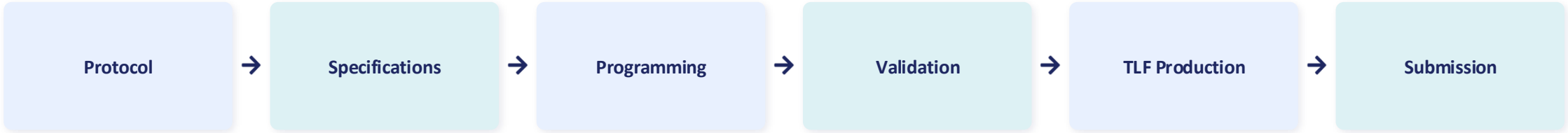
"Will AI arrive in clinical programming?"

"How should we use AI responsibly and effectively?"

Repetitive activities dominating today's workload:

- SDTM mapping
- ADaM derivations
- TLF programming
- Validation programming
- Documentation generation
- Define-XML preparation
- Review guide creation

The Traditional Clinical Programming Model



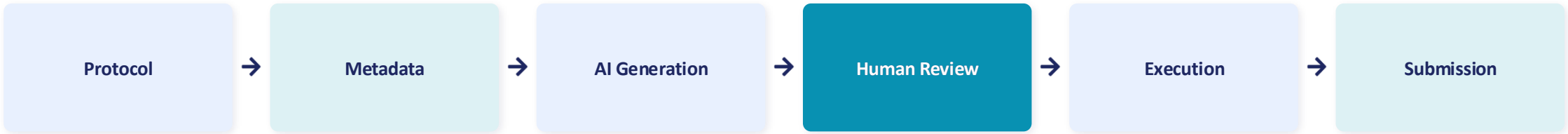
A linear hand-off model — the programmer owns nearly every step end to end.

Within this model, the programmer is responsible for:

- Interpreting specifications
- Writing every line of code
- Developing validation programs
- Fixing discrepancies manually
- Preparing documentation by hand

Value depends on individual coding expertise — but as CDISC standards and macro libraries mature, the marginal value of hand-writing code from scratch declines.

The Emergence of AI-Augmented Programming



Human review is a mandatory gate before execution and submission — not optional, not skippable.

From Coder to Orchestrator

Rather than manually developing every artefact from scratch, the programmer increasingly interacts with AI systems that generate code, draft documentation, interpret specifications, suggest derivation logic, produce validation checks, and generate traceability information. The programmer's job shifts to framing the problem correctly, supplying the right context, and rigorously evaluating what the AI system produces.

How AI Changes the Programmer Role

Yesterday's Programmer

- Writing every line of code
- Manual specification review
- Manual derivation implementation
- Independent QC programming
- Static documentation creation

Tomorrow's Programmer

- Solution Design — shaping automation approaches
- Metadata Engineering — reusable specifications
- AI Output Review — verifying generated logic
- Traceability Verification — regulatory transparency
- Automation Framework Development

Copilot Versus Autopilot

A framework for reasoning about the right degree of AI autonomy in a regulated environment.

Copilot Model

- AI assists
- Human decides
- Human validates
- Human remains accountable

Examples: ADaE derivation generation · TLF skeleton creation · QC checklist generation · Define-XML drafting

Autopilot Model

- AI executes workflow steps
- Human reviews outputs
- Human approves deliverables

Examples: metadata-driven TLF generation · automated compliance checks · automated review reports · reusable submission packages

One Task, Three Levels of AI Assistance

Three operating models for ADAE derivation (or any clinical programming task)

Level	AI Role	Human Role
Manual	No AI involvement	Interpret specs, code, test, document
Copilot	Draft logic, code, QC checks	Provide context, review every derivation, approve
Autopilot	Execute governed metadata-driven workflow	Review exceptions, evidence, final output

In all three models, humans remain in charge. The question is: how much routine work can AI take off the programmer's plate?

The Quality of AI Output Depends on the Prompt

Poor Prompt

```
Create ADAE.
```

No source datasets. No derivation logic. No standards referenced.

Better Prompt

```
Create R code for ADAE dataset using
SDTM AE and ADSL domains.
Derive:
  - TRTEMFL
  - SAFFL
  - TRT01A
Follow ADaM IG principles.
Include validation checks.
```

Source datasets, required variables, regulatory context, and expected outputs are all explicit.

The richer the context, the more reliable — and reviewable — the output.

Context-Rich Prompting: Four Ingredients

Study Context

- Protocol information
- Indication
- Design characteristics

Metadata Context

- Specifications
- Controlled terminology
- Standards

Programming Context

- R (data.table, tidyverse) or SAS
- Performance: arrow, duckdb, furrr
- Naming conventions

Validation Context

- QC checks
- Expected traceability

From Macro Libraries to Governed Prompt Assets

Then: Macro Libraries

Reusable, validated SAS logic that codified organisational best practice and reduced duplicated effort across studies.

Now: Prompt Libraries

- SDTM domain generation
- ADaM derivations
- TLF creation
- Define-XML support
- Review guide generation

From Prompt to Production

End-to-end AI-assisted ADAE development with human control gates.

Metadata & Standards	Rich Prompt Construction	AI Code Generation	Human Review & Validation	Traceability & Testing	Approved Output

Typical Outcomes:

Activity	Expected AI Contribution
Initial code generation	Produces a review-ready first draft
Documentation generation	Produces a structured draft with substantial reuse potential
Human review	Mandatory
Regulatory accountability	Human-owned

Note: Deterministic automation executes predefined rules consistently. Generative AI proposes context-dependent outputs that require evaluation. The strongest operating model combines both: deterministic controls around generative capabilities.

Case Study: AI-Assisted ADAE Derivation

What AI-assisted programming looks like in practice, from specification to review-ready output.

Input

- ADaM specification
- AE and ADSL metadata
- Organisational derivation rules
- Controlled terminology
- Programming conventions

AI-Generated Output

- Draft ADAE derivation logic
- R or SAS code
- Variable-level traceability
- QC checklist
- Assumption and exception log

Human Review

- Confirm treatment-emergent logic
- Review partial-date handling
- Validate population flags
- Verify source-to-target traceability
- Execute tests against golden datasets

Measured Result

- Time to first review-ready draft
- Percentage accepted without material rework
- Number of defects detected
- Review effort compared with the manual process

Opportunities: SDTM & ADaM Development

SDTM Development

- CRF interpretation
- Specification drafting
- Mapping suggestions
- Domain recommendation

Metadata-driven SDTM automation frameworks are already reducing repetitive programming effort in early adopter organisations.

ADaM Development

- Derivation code generation
- Traceability support
- Validation recommendations
- Population flag derivations

Emerging approaches combine metadata-driven automation with AI-generated, study-specific code.

Opportunities: TLF Production & Documentation

TLF Production

- Table code creation
- Shell interpretation
- Statistical programming templates
- QC verification

TFL automation initiatives continue to gain momentum across the industry.

Documentation Generation

- Reviewer guides
- Validation reports
- Programming documentation
- Traceability summaries

Highly structured documentation is well suited to automation — reducing effort while improving consistency.

Risk-Based Use-Case Prioritization

Where should we start? A decision framework for responsible AI adoption.

Low Risk: Automate First

- Formatting and templating
- Documentation boilerplate
- Code commenting
- Define-XML skeletons
- Non-production test-data generation

Medium Risk: Controlled Pilot

- Standard SDTM mapping suggestions
- Standard ADaM derivation drafts
- TLF skeleton generation
- QC checklist generation
- Traceability summary drafting

High Risk: AI-Assisted Cautiously

- Eligibility and protocol deviation logic
- Complex estimand-related derivations
- Novel endpoint programming
- Regulatory submission narratives
- High-impact study flags

Restricted: Specially Controlled

- Unblinded interim analysis
- Randomization logic
- DMC programming
- Sensitive patient-level data
- Safety signal interpretation

Four Critical Guardrails for Safe LLM Use

The opportunities are real — but so are the risks. Four considerations are non-negotiable.

Approved Environment

Only information necessary for the task should be shared with AI systems.

Unblinding Risk

Special care with interim analyses and DMC-related activities to protect study integrity.

GxP Expectations

AI-generated outputs may require validation and oversight before regulated use.

Auditability

Organisations must show what was generated, how it was reviewed, and who approved it.

If AI generates the code, who owns the submission?

Humans.

Regulatory authorities review organisational accountability, not algorithmic accountability. Programmers and study teams remain responsible for correctness, compliance, traceability, and submission quality.

AI can assist. AI cannot assume regulatory responsibility.

The Future Clinical Programmer: Essential Skills

Technical Skills

- R: data.table, arrow, duckdb
- dtplyr & furrr (parallel)
- Python (pandas, polars) & SQL
- Metadata standards

AI Skills

- Prompt engineering & libraries
- RAG & agentic frameworks (MCP)
- LLM evaluation & benchmarking
- AI governance & audit trails

Clinical Skills

- Protocol interpretation
- Endpoint understanding
- Submission readiness

Automation Skills

- Workflow design & orchestration
- CI/CD validation pipelines
- Git / version control
- Reusable, metadata-driven frameworks

Will Clinical Programmers Be Replaced?

The more useful question: which activities will be automated — and which will remain human-centric?

Increasingly AI-Assisted

- Boilerplate coding
- Repetitive derivations
- Standard documentation
- Routine QC checks

Requires Human Judgement & Accountability

- Study interpretation
- Scientific judgement
- Regulatory accountability
- Risk assessment
- Strategy development

One Possible AI-Augmented Workflow for 2030

Programmers spend less time writing repetitive code, and more time on:

- **Reviewing logic**
- **Governing automation**
- **Ensuring quality**
- **Designing reusable solutions**
- **Managing AI-enabled workflows**

Note: Deterministic automation executes predefined rules consistently. Generative AI proposes context-dependent outputs that require evaluation. The strongest operating model combines both: deterministic controls around generative capabilities.

Start Small, Measure, Then Scale

A practical six-week pilot roadmap for responsible AI adoption.

1

Select Bounded Use Case

Low risk, repetitive, well-defined (e.g., Define-XML)

2

Define Review & Approval Gate

Who decides? How do they verify correctness?

3

Build Test Cases

Golden examples: What does correct look like?

4

Measure Quality & Effort

Defects, time savings, percentage usable without rework

5

Document Prompt, Inputs, Outputs

Build your evidence package for regulators

6

Scale Cautiously

Only after acceptance criteria are met

Key Takeaways

A major shift, not a replacement

Comparable in scale to CDISC adoption — AI augments programmers rather than replacing them.

New skills required

Prompt engineering, metadata design, automation governance, and AI oversight.

Accountability stays human

AI can assist; it cannot assume regulatory responsibility for correctness or compliance.

Programmers move up the value chain

From typing code to designing, reviewing, and governing intelligent systems.

The future belongs not to AI alone, and not to programmers alone, but to programmers who know how to work with AI.

connect·share·advance

THANK YOU!

Questions & Discussion

Sivakumar G

Advisor – Clinical Standards | Global Statistical Sciences, Eli Lilly and Company