PharmaSUG 2026 - Paper DS-440

From Specification to SDTM at Speed: Deploying the SDTM Engine in Production

Lynn Xiuling Zhang, Merck & Co., Inc., Rahway, NJ, USA;
Jacques Lanoue, Merck & Co., Inc., North Wales, PA, USA;
Ulf Nielsen, MSD, Zürich, Switzerland

ABSTRACT

Clinical research faces a persistent clinical data bottleneck: manual, fragmented workflows delay compliant SDTM delivery. This paper introduces the SDTM Engine, a metadata-driven, multilanguage platform (Python, SQL) that unifies ingestion, specification authoring, and generation of audit-ready SDTM outputs. Centralizing mappings and derivations in a machine-understandable specification (MUS), the Engine collapses mapping and programming into a single, natural-language-like authoring step, reducing Excel-based instruction overhead and accelerating delivery with traceable, executable specifications. Key capabilities include pre-defined and extendable derivation functions, the Function Standard Library with governed workflows, comprehensive lineage and traceability reporting, and embedded automation for validation and observability. Cloud-native architecture and a centralized metadata store enable scaling across heterogeneous sources (EDC, labs, operational systems) while ensuring completeness, consistency, and compliance. To maintain speed without compromising GxP policy, the program adopts a gated System Development Life Cycle (SDLC) with automated testing and parallel Function Standard Library development under a separate SOP, yielding predictable, audit-ready releases. Beyond technical advances, the Engine catalyzes business transformation by redefining roles (Study Lead, Developer/System Architect, Metadata Analyst), formalizing standard library lifecycle artifacts, and investing in training for MUS authoring and multilanguage proficiency. The results demonstrate faster, safer SDTM delivery; reduced handoffs; improved transparency; and enhanced reuse, positioning teams to meet evolving regulatory expectations with higher data quality and scalability. The paper provides practical guidance on architecture, workflow patterns, features, and validation strategies to modernize SDTM automation across complex trials.

INTRODUCTION

Clinical research faces a persistent “clinical data bottleneck,” where manual, fragmented, and inefficient workflows slow the path from scientific discovery to regulatory submission. Traditional processes rely on human-controlled tasks and siloed systems that don’t communicate well, creating errors, delays, and inconsistent data interpretations across teams. While community initiatives such as CDISC OAK demonstrate the potential of metadata-driven automation, they often depend on bespoke development on the algorithms within formatted R packages; as study designs and data grow more complex, this customization burden impedes broad adoption. As regulatory expectations intensify, especially around standardized SDTM datasets demanded by agencies like the FDA, the industry needs automation that delivers speed, accuracy, and full compliance, simultaneously and on a scale.

The SDTM Engine addresses these gaps with a multilanguage integration (Python, SQL), metadata-driven platform that automates a three-step flow: ingesting heterogeneous raw sources, specification authoring to standardize SDTM by unifying traditional mapping-to-programming, and generate compliant, audit-ready outputs by simply clicking one button. It outputs a complete submission package with machine-readable metadata, annotated CRFs, blueprints, and reviewer guides, while maintaining full lineage tracking so every data point is transparently traceable from source to submission. Built on a validated development process, the Engine enables users to use functions from standard library, enabling rapid adaptation to evolving regulations and study needs. Beyond technical innovation, the SDTM Engine catalyzes operating model change by evolving processes, roles, and skills within clinical programming to deliver consistent, auditable, and scalable SDTM automation.

HIGH-LEVEL OVERVIEW OF SDTM ENGINE SYSTEM

PURPOSE AND SCOPE

- Deliver a single, unified system to ingest heterogeneous sources, author machine-understandable specifications (MUS), and generate compliant SDTM outputs.
- Support multilanguage integration (Python, SQL) and governed extensibility to adapt to evolving study needs and regulations.

ARCHITECTURE AND CLOUD-SCALE DESIGN

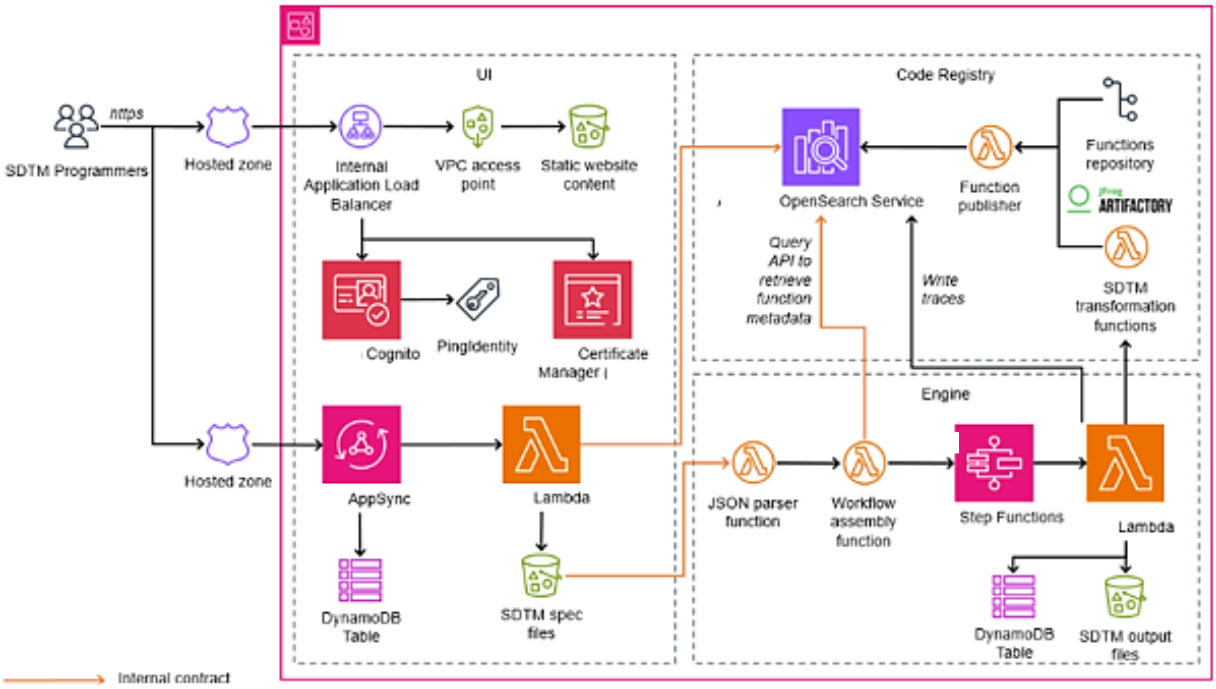

Figure 1. System Architecture

The system architecture as shown in Figure 1 includes UI and Code Registry modulars.

- User Interface (UI) provides workplace for SDTM programmers to author specifications without code programming.
- Cloud-native of Code Registry services support generating JSON Specifications, assembling and processing back-end Python-based functions and derivations, and creating deliveries.
- Centralized metadata store for mappings, derivations, and controlled vocabulary.
- Service modules for ingestion, transformation, validation, documentation, and observability.

SYSTEM DATAFLOW

System contains two work environments for consuming different data from one centralized data source for different business purposes. It enables flexible orchestration of data transformation and validation of workflows across diverse environments. Figure 2 shows how the data flows across the two environments along with the business activities.

- Pre-preview environment for Specification authoring and standardization and consuming mock data.
- Production Environment for study data execution per the Specification and generating the Auditable

deliveries by consuming study data

By ingesting heterogeneous raw sources, system transforms data into standardized SDTM, and generates compliant, audit-ready outputs. It embeds intelligence across critical data science tasks (e.g., complex derivations, unit conversions, event timing), reducing manual effort while elevating consistency.

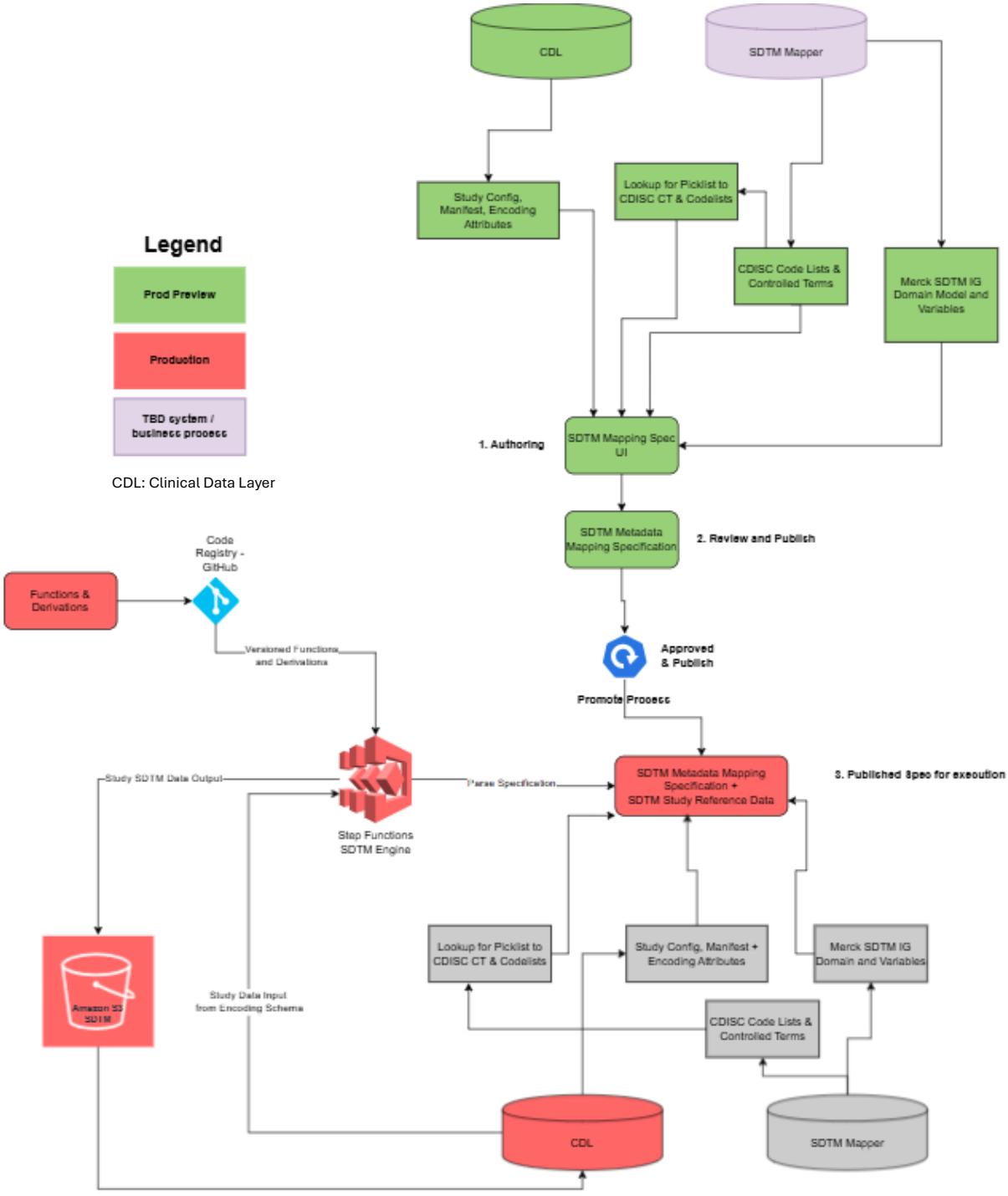

Figure 2. Data Flow in Prod Preview Environments and Production Environment

END-TO-END BUSINESS WORKFLOW

The SDTM Engine is designed to replace traditional, fragmented programming processes with a single, end-to-end workflow managed entirely through the user interface (Figure 3). An SDTM programmer begins by selecting the assigned study and configuring key parameters (e.g., SDTM IG and MedDRA versions) either directly in the UI or by uploading the relevant reference files. Within the same interface, the programmer then authors the sourcetotarget domain mappings by selecting from standardized, production-ready, preprocessing derivations and functions. Once the mappings are complete, a single action ("Publish") triggers automated generation of SDTM domains, logs, and related outputs in the prodpreview environment.

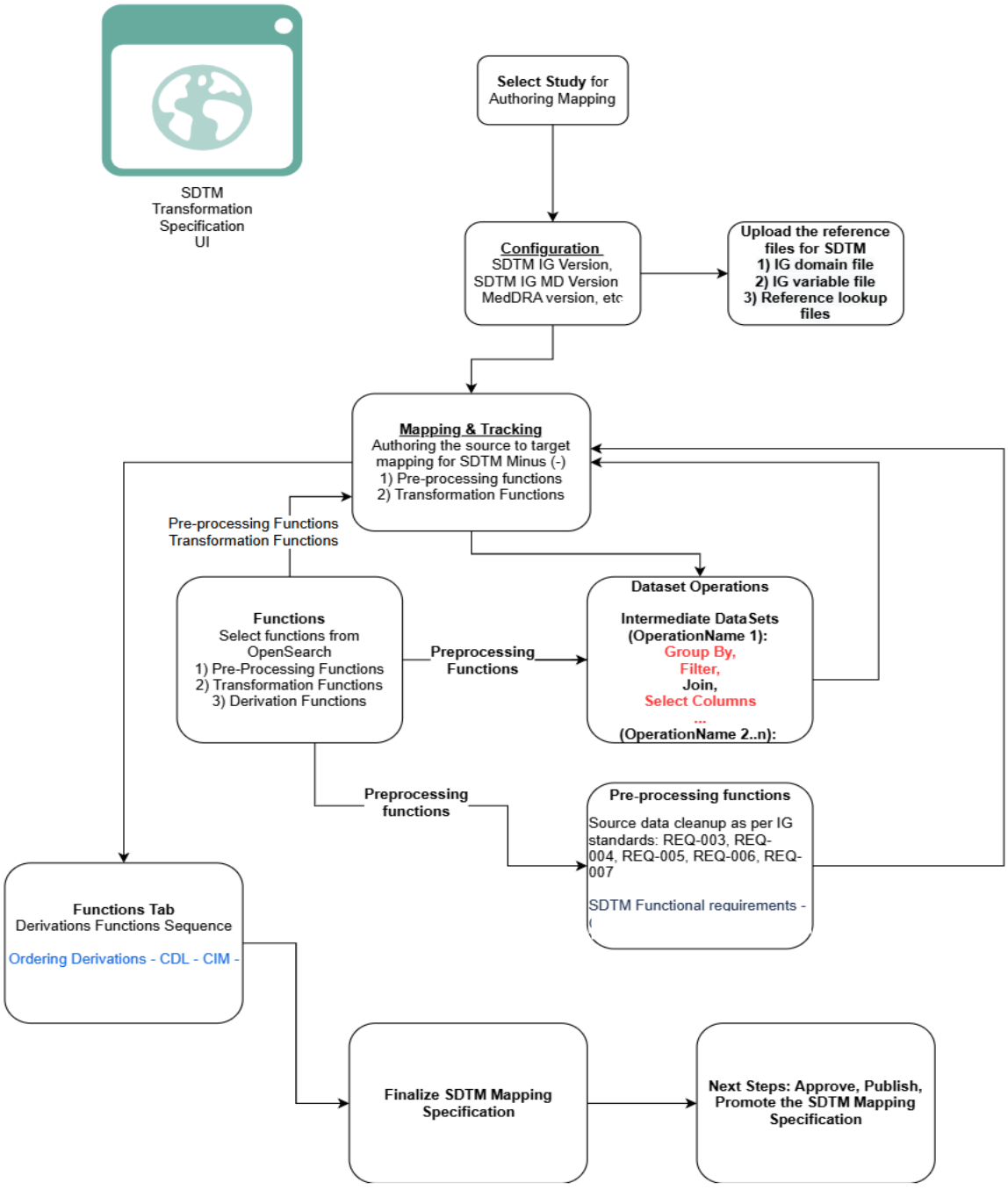

Figure 3. End-to-End business workflow

After review and approval, the study lead promotes the specification, at which point the Engine creates a JSON representation of the finalized mapping specification, executes it against live study data, and automatically produces SDTM domains and other audit-ready deliverables. This automation minimizes manual programming and data handling, thereby reducing the risk of human error, improving consistency across studies, and accelerating delivery timelines. Because the specifications are executable and centrally managed, the Engine provides comprehensive lineage and traceability reporting, from source data through intermediate transformations to final SDTM outputs, along with embedded validation and observability capabilities.

Together, these features support higher accuracy, regulatory-ready documentation, and scalable, repeatable SDTM production.

SYSTEM KEY FEATURES AND INNOVATIONS

UNIFIED MAPPING AND PROGRAMMING

The SDTM Engine collapses what has traditionally been a two-step, handoff-driven process into a single, natural-language-like authoring step. It clearly shows how mapping and programming are unified across the end-to-end business workflow.

In the conventional model, the study lead first drafts a mapping specification manually, often in Excel, and then programmers translate that specification by programming and validating it, a process often referred to as “double programming.” In contrast, within a single user interface, the SDTM Engine allows the SDTM programmer to author the Machine Understandable Specification (MUS) directly by defining standard functions that are ready for production use that has been developed and validated outside of the Engine.

By eliminating Excel-based specification instructions and programming duties, this model significantly improves efficiency and accuracy. It reduces handoffs and rework, reduces manual effort and the potential for human errors, minimizes the risk of misinterpretation, and creates a single, authoritative source of truth. Intent, executable specification, and resulting datasets are all linked in one traceable chain, enabling streamlined SDTM delivery with clear lineage, stronger auditability, and more reliable outcomes.

SDTM ENGINE FUNCTION STANDARD LIBRARY

In the user interface (UI), SDTM programmers can invoke functions and derivations in standard library directly during mapping authoring, without writing any code. The custom functions are developed and validated in Python through a separate, regulated process, and are stored in the Github which could be uploaded and pushed into SDTM Engine to ensure system compatibility and performance.

The Function Standard Library are designed to be flexible and intuitive, so they can be reused across multiple studies and contexts. For example, scenarios such as DIRECT_MOVE (described in Figure 4) can be implemented once and then applied consistently wherever needed. This reusability is a key feature that enables the SDTM Engine to scale across many studies and systems and to be adopted broadly.

By decoupling function development from day-to-day mapping authoring and allowing programmers to consume these validated functions directly in the UI, the SDTM Engine supports rapid adaptation to evolving regulations and ongoing study requirements, while maintaining control, consistency, and high quality.

```
Feature: DIRECT MOVE function
    The DIRECT_MOVE function is designed to move the value of a variable in the
    source data to an existing variable in the target dataset and record the traceability information.
    It ensures data consistency and provides traceability of the mapping.

 Scenario Outline: Direct move value in a source variable to specified target variable
    Given the <source_dataframe> exists
    And the <source_column_name> exists in the <source_dataframe>
    And the <target_dataframe> exists
    And the <target_column_name> exists in the <target_dataframe>
    When the direct move function is executed
    Then the <target_dataframe> has a column named <target_column_name>
    And the value in the column <source_column_name> for each record in the <source_dataframe> exists in
        the column <target_column_name> for each record in the <target_dataframe>


 Examples: Direct move value in a source column to specified target column
| source_dataframe | source_column_name | target_column_name | codelist        | target_dataframe  |
  AE_input.csv     | AE_AETERM          | AETERM             |                 | ae_target.csv     |
  AE_input.csv     | AE_AEOUT           | AEOUT              |                 | ae_target.csv     |
  AE_input.csv     | AE_AESEV           | AESEV              |                 | ae_target.csv     |
  AE_input.csv     | AE_AESER           | AESER              | YN_ct.json      | ae_target.csv     |
  AE_input.csv     | AE_AESCAN          | AESCAN             | YN_ct.json      | ae_target.csv     |
  AE_input.csv     | AE_AESOD           | AESOD              | YN_ct.json      | ae_target.csv     |
  CM_input.csv     | CMTRT              | CMTRT              |                 | cm_target.csv     |
  CM_input.csv     | CMINDC             | CMINDC             |                 | cm_target.csv     |
  CM_input.csv     | CMDOSU             | CMDOSU             |unit_ct.json     | cm_target.csv     |
```

Figure 4. The feature of DIRECT_MOVE and one example of its scenarios in Notepad++

MACHINE UNDERSTANDABLE SPECIFICATIONS (MUS)

Machine Understandable Specifications (MUS) generated by the SDTM Engine are stored in a JSON format that is both human-readable and machine-executable. The Engine consolidates all mapping information authored by the SDTM programmer in the UI, including back-end functions, derivations, standard functions, and reference files, and compiles them into a single downloadable JSON file. This MUS serves as a centralized, executable specification for the Engine, enabling automation, validation, and efficient reuse across studies.

JSON Structure Overview

It is the Specification for SDTM transformations, mapping source variables from clinical trial datasets to SDTM domains and variables.

It includes:

- Study Metadata: Study ID, therapeutic area, study name, SDTM version, dictionary versions, publish info.
- Mappings: Each mapping links a source table/variable to a target SDTM domain/variable, with transformation functions, code lists, types, and groupings.
- Functions: Lists of transformation, derivation, and preprocessing functions with IDs and versions.
- Operations: Dataset operations and derivations.

Documented Schema

The MUS file follows a documented schema that captures the structure of the specification, including top-level fields (Figure 5), mapping entries (Figure 6), functions (Figure 7), and operations (Figure 8).

Field	Type	Description
Study ID	string	Unique identifier for the study
ParsedStudyId	string	Parsed version of Study ID
TA	string	Therapeutic area
Study Name	string	Name of the study
SDTM Version	string	SDTM Implementation Guide version
Dictionary Versions	string	List of dictionaries (e.g., MedDRA, WHODD, LOINC, CT)
Publish User ID	string	User who published the specification
Publish Date Time	string	Timestamp of publication
Specification Name	string	Name of the specification

Figure 5. Top-Level Fields

Field	Type	Description
SourceTable	string	Name of the source table
SourceVariable	string	Name of the source variable
SourceVariableAlias	string	Alias for the source variable
InterimDataset	string	Indicates interim dataset usage
TargetList	string	List of target mappings
SDTMDomain	string	SDTM domain (e.g., AE, CM, DM, EG, EX, MH, VS, etc.)
SDTMVariable	string	SDTM variable name
SDTMTransformationFunction	string	ID of the transformation function
FunctionParameterValue	string	Parameter value for the function
Group	integer	Grouping indicator
SDTMCodelist	string	Codelist used for the variable
SDTMType	string	Data type (e.g., text, integer, float)

Figure 6. Entries in each mapping

Field	Type	Description
Function	string	Function ID
Version	integer	Function version
Sequence	integer	Sequence/order of function

Figure 7. Functions

Field	Type	Description
studyDatasetOperations	string	Dataset operations
datasetOperations	string	Operations on datasets
sdtmDerivations	string	SDTM derivations

Figure 8. Operations

Key Patterns and Relationships

Check Program 1 for the patterns of the JSON structure:

- Domain Mapping: Each source variable is mapped to a target SDTM domain/variable, often with a transformation function and codelist.
- Function References: Transformation logic is modular, referenced by unique function IDs and versions.
- Grouping: Some mappings are grouped (Group 0, Group 1, etc.), for logical separation (Transpose).
- Type Enforcement: Each SDTM variable is assigned a type (text, integer, float), ensuring data consistency with the SDTM IG.

```
{
 "StudyID": "string",
 "ParsedStudyId": "string",
 "TA": "string",
 "StudyName": "string",
 "SDTMVersion": "string",
 "DictionaryVersions": ["string"],
 "PublishUserID": "string",
 "PublishDateTime": "string",
 "SpecificationName": "string",
 "Mappings": [
 {
 "SourceTable": "string",
 "SourceVariable": "string",
 "SourceVariableAlias": "string",
 "InterimDataset": "string",
 "TargetList": [
 {
 "SDTMDomain": "string",
 "SDTMVariable": "string",
 "SDTMTransformationFunction": "string",
 "FunctionParameterValue": "string",
 "Group": "integer",
 "SDTMCodelist": "string",
 "SDTMType": "string"
 }
 ]
 }
 ],
 "Functions": [
 {
 "Function": "string",
 "Version": "integer",
 "Sequence": "integer"
 }
 ],
 "Operations": {
 "studyDatasetOperations": "string",
 "datasetOperations": "string",
```

```
  "sdtmDerivations": [
  {
  "Function": "string",
  "Version": "integer",
  "Sequence": "integer"
  }
  ]
  }
}
```

Program 1. Example Schema Fragment in JSON Specification

CHALLENGES AND ADAPTIVE SOLUTIONS

As the SDTM Engine evolved from concept to an operational platform, we faced a dual pressure: fast-changing study demands and the need to maintain a high-confidence, compliant system. Rather than letting these pressures degrade quality or slow delivery, we reframed them as design constraints and used them to shape our operating model, architecture, and governance.

SYSTEM QUALITY STRATEGY AND COMPLIANCE

Challenge: Fast timelines from ongoing studies and Agile delivery risked regression, inconsistent behavior, and documentation gaps, potentially eroding regulatory confidence. SDTM Engine Function Standard Library needed to evolve quickly without compromising GxP controls.

Opportunity: Treat speed as a driver for automation and governance. Separate technical platform development from business function standard library development to enable parallel progress while preserving compliance.

Adaptive Solutions:

- System Development Life Cycle (SDLC) discipline for Agile: Defined gated SDLC with documented requirements, design reviews, and controlled releases. Automated unit, integration, and regression tests. Enforced structured SIT/UAT with clear entry/exit criteria and traceable outcomes.
- Parallel development under a separate SOP: Business team is responsible for developing, updating, validating and maintaining the standard functions through the Analysis and Reporting process which is separated from the System Development Life Cycle (SDLC) procedures. It enables system development and code development to proceed parallelly while preserving compliance.

Desired result: Predictable, audit-ready releases with rapid study support; continuous compliance through embedded testing, validation, and governed promotion.

MANAGEMENT TRANSFORMATION

Challenge: Workflow innovations, including unified mapping and programming into a single authoring step, natural-language-like specifications, and standard functions contributions, reshaped responsibilities and skill requirements for study leads and programmers.

Opportunity: Elevate roles toward standards stewardship, reuse, and quality; formalize standard function lifecycle; invest in multilanguage and metadata-driven competencies.

Adaptive Solutions:

New processes: Instituted a standard function lifecycle with required artifacts (design notes, test cases, validation reports, lineage, change logs, etc.).

Role redefinition:

- SDTM Study Lead: Owns trial-level mapping and unified authoring; orchestrates cross-functional input; ensures compliance and delivery.
- SDTM Developer/System Architect: Designs and maintains global/study functions and derivations;

stewards the standard function catalog; enforces standards and performance.

- SDTM Metadata Analyst: Reviews mappings and derivation specs; ensures terminology alignment, SDTM conformance, and consistency; leads traceability checks.

Training: Multilanguage proficiency (Python, SQL), MUS authoring, validation/observability tooling, and structured onboarding.

Desired result: Faster, safer delivery; consistent, reusable function standard library; strengthened audit readiness and team adaptability.

CONCLUSION

The SDTM Engine demonstrates that a modular, multilanguage architecture combined with a single-step, natural-language-like authoring paradigm and extending it through Function Standard Library for governed, rapid function customization, can materially improve SDTM automation across complex trials. By unifying mapping and programming into traceable, executable specifications and enabling parallel, SDLC-aligned development of custom logic, the approach reduces handoffs and instruction overhead, enhances reusability and transparency, and shortens delivery timelines. Just as importantly, it enables measurable organizational gains by standardizing workflows, elevating skillsets, and fostering a culture of continuous automation, positioning teams to meet evolving regulatory expectations with higher data quality and scalability.

By presenting practical insights on system design, workflow patterns, key features, and validation strategies, the paper offers guidance for clinical programmers and data managers seeking to modernize SDTM automation and improve data quality across their organizations.

REFERENCES

https://www.cdisc.org/

Faster Data Transformation - SDTM Engine - Foundational Principles of the SDTM Engine. Available at https://www.linkedin.com/pulse/faster-data-transformation-sdtm-engine-jacques-lanoue-29x7e/

Faster Data Transformation - SDTM Engine - Machine Understandable Specifications (MUS). Available at https://www.linkedin.com/pulse/faster-data-transformation-sdtm-engine-jacques-lanoue-hb7ne/

ACKNOWLEDGMENTS

This work is dedicated to the exceptional professionals at Merck & Co., Inc. who made this effort possible. We are especially grateful to Srinivas Malipeddi for his guidance on the Function Standard Library development and implementation section and to Bhaskar Ponugoti for his contributions and for providing innovative ideas in the industry solutions such as CDISC-OAK.

CONTACT INFORMATION

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

Lynn Xiuling Zhang
Merck & Co., Inc., Rahway, NJ, USA
Xiuling.zhang@merck.com

Jacques Lanoue
Merck & Co., Inc., North Wales, PA, USA
jacques.lanoue@merck.com

Ulf Nielsen
MSD, Zürich, Switzerland
ulf.nielsen@msd.com