PharmaSUG China 2025 - Paper AI-177

Cross-module streamlining: database and UI designs to foster ADaM spec-code synchronization

Nana Yang, Tingting Zeng, Yaohui Zhu, Yanan Sui, Mingliang Fan and You Li, BeOne

ABSTRACT

The traditional way of interacting between ADaM specification and code is to make modifications in Excel file and then notify corresponding programmers to modify ADaM code. This inevitable frequent modification of specifications and codes leads to heavy communication and risk of misalignment. Additionally, it is challenging to track historical versions of both specifications and codes. To effectively address these challenges, we have developed ACIRA (ADaM Code Intelligence for Rapid Analytics) - an AI-driven system designed to visually present real-time modifications to ADaM specifications and facilitate code generation and management. Additionally, ACIRA collaborates with the company's upstream product called SpecMaster (ADaM Specification Management) , which is a tool equipped with functionalities such as precise version control implementation and efficient specification management to enhance workflow integration. By integrating with SpecMaster, ACIRA can access the latest and most accurate specification versions. This integration ensures that the code generation process in ACIRA is always based on the most up-to-date information, enhancing the overall workflow integration and coherence between specification management and code development.

This paper aims to offer a comprehensive technical overview of version control system and user interface (UI) design for specification modifications and ADaM spec-code synchronization. Leveraging Python's SQLAlchemy, a powerful Object Relational Mapping (ORM) framework and ACIRA exclusive tables that achieve interconnection with upstream SpecMaster through foreign keys, database enables efficient querying, restoration, and storage of specification and code modifications made at any given time. In addition to its advanced backend functionality, ACIRA features a highly streamlined and visually intuitive UI, which allows users to effortlessly monitor and track any specification modification in real-time. Every modification is presented in a clear and accessible manner, enabling quick identification of any difference. The dashboard enables visual tracking of horizontal comparisons of various metrics across different projects. What's more, the system is designed with user-efficiency in mind. Based on the updated specification content, users can trigger AI code generation with a single click. This seamless integration creates a continuous, real-time workflow that spans from specification modifications to ADaM code generation and management , eliminating manual bottlenecks and enhancing the overall efficiency of data analysis process.

INTRODUCTION

In the dynamic landscape of data analysis and management, especially within the pharmaceutical and clinical research sectors, the ADaM stands as a cornerstone for structuring and analyzing clinical trial data. However, the traditional methodology for interacting between ADaM specifications and code has long been a source of inefficiency and even error, hindering the seamless progress of data-driven projects. The conventional approach to managing ADaM involves making modifications to specifications within an Excel file, followed by notifying programmers to update the corresponding ADaM code. This seemingly straightforward process is fraught with significant drawbacks.

Firstly, the iterative nature of specification and code modifications generates a substantial communication burden. Each alteration necessitates meticulous coordination between specification managers and code developers. For instance, in a large-scale clinical trial project involving multiple stakeholders, a minor change in the ADaM specification might require numerous email exchanges, meetings, and clarifications to ensure that developers or stakeholders more fully understand the update. This back-and-forth communication not only consumes valuable time but also increases the likelihood of misunderstandings, as information can be lost or misinterpreted during the transfer.

Secondly, the risk of misalignment between the specification and code looms large. Since modifications are implemented sequentially, there is a constant danger that the code fails to mirror the updated specification accurately. In a regulatory environment where precision is paramount, such misalignments can lead to incorrect data analysis results. This, in turn, may cause delays in drug development timelines and pose potential compliance risks.

Another critical challenge lies in the arduous task of tracking historical versions of both specifications and code. In an ever-evolving project, with numerous modifications made over time, it's essential to have a clear record of modifications, timestamps, and responsible parties. Without an effective version control mechanism, tracing back the development history becomes a herculean task. It becomes difficult to pinpoint when a particular issue arose, who makes modification, how to understand the rationale behind past decisions, or track a previous, stable state of the ADaM implementation in case of errors.

To surmount these formidable challenges, our team has engineered ACIRA, an AI-driven system that ushers in a new era of the dynamic collaboration between ADaM specification and code.

THE ADVANCED AUDIT TRAIL VERSION CONTROL SYSTEM OF ACIRA

LEVERAGING SQLALCHEMY

A pivotal component of database design is its advanced version control system, which is built upon a database versioning tool for Python - SQLAlchemy, a powerful ORM framework. SQLAlchemy has several features:

1. Not store updates which don't change anything.
2. Support alembic migrations.
3. Revert objects data as well as all object relations at given transaction even if the object has been deleted, transactions can be queried afterwards using SQLAlchemy query syntax.
4. Query for changed records at given transaction.
5. Query for versions of entity that modified given property.
6. Query for transactions, at which entities of a given class changed, history models give access to parent objects relations at any given point in time.

So SQLAlchemy provides a high-level, abstract interface for interacting with databases, enabling efficient querying, restoration, and storage of specification and code modifications. With SQLAlchemy, ACIRA can perform complex database operations with relative ease. For example, it can query the database to retrieve specific versions of the ADaM specifications and code, allowing users to review past states of the project. It also enables the restoration of previous versions if needed, providing a safety net in case of accidental deletions or incorrect modifications. The storage of specification and code modifications are optimized, ensuring that data is stored in a structured and accessible manner, even as the volume of modifications accumulates over time.

EXCLUSIVE TABLES AND FOREIGN KEY INTERCONNECTION

ACIRA utilizes exclusive tables designed to establish a connection with the upstream SpecMaster through foreign keys. These tables serve as a central repository for all information related to ADaM specifications and code. This design allows separation of dataset definition and implementation (similar to template-instance relationship) and cascade deletion and dynamic loading through foreign key associations.

The use of foreign keys creates a relationship between the tables, ensuring that the data is organized in a way that facilitates easy retrieval and analysis. For instance, when a user wants to view the code modifications associated with a particular specification version, the foreign key relationships allow the system to quickly retrieve the relevant data from different tables. This structured data storage and retrieval mechanism not only improves the efficiency of the version control system but also provides a solid foundation for auditing and traceability, which are crucial in regulated industries.

DATABASE DESIGN

The database tables have the following hierarchical classification, first divided into four layers: dataset, variable, paramcd, and codelist. For the upstream SpecMaster, it stores tables of the four layers and their corresponding version tables. For the downstream ACIRA, in addition to the tables of the four layers connected to the upstream SpecMaster via foreign keys (without columns from SpecMaster), there are other auxiliary tables, such as:

1. ADaM code examples tables for the four layers.
2. Tables related to recording specification issues.
3. Tables related to recording then time use of development ADaM code by dataset.
4. Tables related to trial design attribute settings and structured Standard Definitions and Derivation Rules of Key Variables.
5. AI-related tables, including code generated by prompts table, prompt libraries table, AI conversation table, and AI model-related table.

For a specific study, the adam_setting_id (denoting the study ID) remains consistent across all relevant tables, ensuring unified identification.

Figure 1 is the database tables and connection of upstream SpecMaster and downstream ACIRA

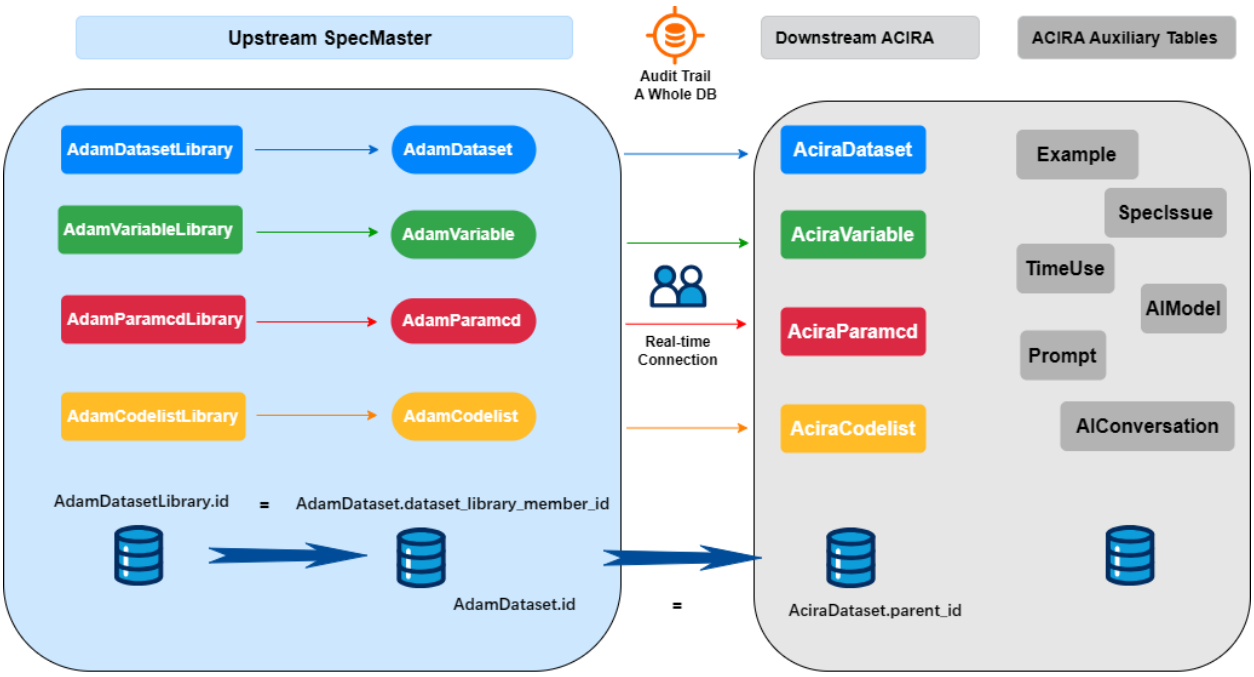


Figure 1. Database tables and connection of upstream SpecMaster and downstream ACIRA

Here is an example for implementing version control and audit trail functionality for ADaM specification and code using SQLAlchemy-Continuum, while leveraging foreign keys to establish relationships between different entities. This mechanism combines version control and foreign key associations to build a full-chain audit trail system from specification definition to code implementation.

One is for audit trail implementation mechanism,

Firstly, make_versioned(user_cls=User) is the core component initialization. It enables version control via SQLAlchemy-Continuum and specifies the User model to track change operators. Automatically generates the following audit components:

1. Transaction Table: Records timestamps, user IDs, and descriptions of all modifications (e.g., adam_dataset).
2. History Tables: Automatically creates history tables (e.g., adam_dataset_version) for each version-controlled model.
3. Version Records: Automatically saves the previous state to the history table each time data is changed.

Secondly, __versioned__ = {} is for model-Level version control configuration. When no fields are excluded, SQLAlchemy-Continuum enables version tracking for all fields of the model by default. Each time a record is changed, a history table is automatically created to store the previous state.

When modifying entities and committing the modifications to the database, Continuum records the operations performed (INSERT, UPDATE, or DELETE) into version entities. These operation types are stored by default in a small integer field named operation_type. The "Operation" class provides convenient constants for these values, as demonstrated in the following example:

The first version's operation type is INSERT

article.versions[0].operation_type == Operation.INSERT

The second version's operation type is UPDATE

article.versions[1].operation_type == Operation.UPDATE

The third version's operation type is DELETE

article.versions[2].operation_type == Operation.DELETE

Thirdly, there are mainly three kinds of key features of audit trail.

1. Automatic history recording: Every create, update, or delete operation on table automatically stores a change record in the corresponding history table. History tables include all fields from the original table, plus transaction_id (transaction ID), end_transaction_id (transaction ID when the record became invalid), and operation_type (operation type: INSERT/UPDATE/DELETE).

2. Transaction-Level Tracking: version_class(Model) returns the history class for the model, used to query records from specific versions.

3. Transaction Table: SQLAlchemy-Continuum automatically creates a transaction table to record timestamps, operators (via the User model), and descriptive information for each change.

The other one is for foreign key relationships and data association design,

1. Cross-hierarchy foreign key association (SpecMaster to ACIRA) is the bidirectional association. AciraDataset associates with the adam_dataset_version history table via composite foreign key (parent_id, parent_transaction_id). Enables precise mapping of ACIRA metadata to historical versions of SpecMaster, ensuring traceability to specific specification versions.

2. Inter-model relationship definition (within SpecMaster). Cascade operations: cascade='all, delete-orphan' ensures that orphaned child records are automatically deleted when the parent is deleted and modifications to child records are automatically synchronized to parent relationships. Lazy Loading: lazy='dynamic' avoids loading all associated records at once, optimizing queries for large datasets.

3. Multi-level foreign key chains (Dataset → Variable → Paramcd). Passive delete support: "passive_deletes=True" combined with foreign key "ondelete" settings enables: When AdamDataset is deleted, associated AdamVariable and AdamParamcd records are automatically deleted via foreign key constraints. Improves database operation efficiency without additional cascade configurations.

4. ACIRA-specific foreign key associations. AI Code Association associates ADaM datasets with AI-generated code (PromptCode) via foreign keys. Audit value records the mapping between dataset definitions and AI code for traceability of code generation context.

Here is a code example.

```
pip install SQLAlchemy-Continuum
import sqlalchemy as sa
from sqlalchemy_continuum import ma

make_versioned(user_cls=User)

# table in SpecMaster library
class AdamDatasetLibrary(PrimaryKey, PaginatedAPIMixin, TimestampMixin, db.Model):
    __versioned__ = {}
__tablename__ = 'adam_dataset_libraries'
    name = db.Column(db.String(16), nullable=False)
    description = db.Column(db.String(500), nullable=False)
    type = db.Column(db.String(200), nullable=False)
    structure = db.Column(db.String(500), nullable=False)
    purpose = db.Column(db.String(500), nullable=False)
    key_variables = db.Column(db.String(500), nullable=False)
    comment = db.Column(db.String(2000), nullable=True)
    document = db.Column(db.String(500), nullable=True)
    programming_comment = db.Column(db.String(5000), nullable=True)
    pages = db.Column(db.String(500), nullable=True)

# table in SpecMaster
class AdamDataset(PrimaryKey, PaginatedAPIMixin, TimestampMixin, db.Model):
    __versioned__ = {}
    __tablename__ = 'adam_datasets'
    name = db.Column(db.String(16), nullable=False)
    order = db.Column(db.Integer, nullable=True)
    dataset_library_member_id = db.Column(UUIDType(binary=False),
db.ForeignKey('adam_dataset_libraries.id'), nullable=False, index=True)
    definition = db.relationship('AdamDatasetLibrary',
backref=db.backref('implemented_instances', cascade='all, delete-orphan',  lazy='dynamic'))
    adam_setting_id = db.Column(UUIDType(binary=False), db.ForeignKey('adam_settings.id',
ondelete="CASCADE"), nullable=False, index=True)
    variables = db.relationship('AdamVariable', backref='dataset', lazy='dynamic',
cascade='all, delete-orphan', passive_deletes=True)
    parameters = db.relationship('AdamParameter', backref='dataset', lazy='dynamic',
cascade='all, delete-orphan', passive_deletes=True)
    codelists = db.relationship('AdamCodelist', backref='dataset', lazy='dynamic',
cascade='all, delete-orphan', passive_deletes=True)
    tags = db.relationship('AdamTag', backref='dataset', lazy='dynamic')
    ready = db.Column(db.Boolean, default=True)
    example_code = db.relationship("PromptCode", backref='acira_meta_dataset',
    cascade='all, delete-orphan', passive_deletes=True, lazy='dynamic')
    prompt_step_code = db.relationship("PromptStepCode", backref='acira_meta_dataset',
```

```
                                        cascade='all, delete-orphan', passive_deletes=True,
lazy='dynamic')
    select_examples = db.relationship('DatasetSelectExample', backref='acira_meta_dataset',
                                      cascade='all, delete-orphan', passive_deletes=True,
lazy='dynamic')

# table in ACIRA
class AciraDataset(PrimaryKey, PaginatedAPIMixin, TimestampMixin, db.Model):
    __versioned__ = {
        'exclude': ['example_code', 'prompt_step_code', 'select_examples']
    }
    __tablename__ = 'acira_meta_dataset'
    __table_args__ = (
        db.ForeignKeyConstraint(
            ['parent_id', 'parent_transaction_id'],
            ['adam_datasets_version.id', 'adam_datasets_version.transaction_id'],
            ondelete="NO ACTION"
        ),
    )
    parent_id = db.Column(UUIDType(binary=False), nullable=True, index=True)
    parent_transaction_id = db.Column(db.BigInteger, nullable=True, index=True)
    code_adjust = db.Column(db.String(5000))
    ready_for_code = db.Column(db.String(200))
    spec_issue = db.Column(db.Boolean)
    spec_issue_details = db.Column(db.String(2000))

# after define all models, call configure_mappers:
sa.orm.configure_mappers()
# Validates version class relationships using version_class and parent_class
from sqlalchemy_continuum import version_class, parent_class
version_class(AdamDataset)  # AdamDataset History class
parent_class(version_class(AdamDataset))  # AdamDataset classquit;
```

Table 1. Example for data storage in database

AdamDatasetLibrary (upstream SpecMaster Library)

id	created_at	updated_at	created_by	updated_by
47819ca7920d4f03b3b1da4a364beee4	XX	XX	XX	XX

name	description	type	structure	key_variables	comment
ADSL	Subject-Level Analysis Dataset	SUBJECT LEVEL ANALYSIS DATASET	One record per subject	STUDYID, USUBJID	XXXX

AdamDataset (upstream SpecMaster)

id	name	dataset_library_member_id	adam_setting_id
79a70bf2d73f4ed5832e10d919a27754	ADSL	47819ca7920d4f03b3b1da4a364beee4	723e3a78765e4bf9b376c81b4c02c847

AciraDataset (downstream ACIRA)

id	code_adjust	ready_for_code	parent_id	parent_transaction_id
078e243a82b046bb8da1629fc8efaa3a	where not missing(ARMCD);	Y	79a70bf2d73f4ed5832e10d919a27754	7475

AciraDataset Version (downstream ACIRA)

id	code_adjust	parent_id	parent_transaction	transaction	End_transa	Operation_t

			_id	_id	ction_id	ype
078e243a82b046bb8da1629fc8efaa3a	where not missing(ARMCD);	79a70bf2d73f4ed5832e10d919a27754	7475	7594	7698	0
078e243a82b046bb8da1629fc8efaa3a	where not missing(ARMCD);test111	79a70bf2d73f4ed5832e10d919a27754	7475	7698		1

THE INTUITIVE UI FOR SPECIFICATION CHANGES AND CODE SEAMLESS SYNCHRONIZATION AND CONSISTENCY CHECK

The UI of ACIRA is highly interactive, designed to facilitate a seamless workflow from specification modifications to code generation. Based on the updated specification content, users can trigger code generation with a single click. This simplicity in operation eliminates the need for manual intervention at multiple stages, reducing the time and effort required to move from specification updates to code implementation.

The seamless integration between specification management and code generation creates a continuous, real-time workflow. This means that as soon as a specification change is made and approved, the corresponding code can be generated and deployed quickly. By eliminating manual bottlenecks, ACIRA significantly enhances the overall efficiency of the data analysis process, enabling ADaM programmer to complete projects faster and with greater accuracy.

REAL-TIME VISUALIZATION OF SPECIFICATION MODIFICATIONS WITHIN ONE PROJECT

One of ACIRA's core advantages is its intuitive user interface centered on user experience. ACIRA features a highly streamlined and visually intuitive UI, enabling seamless real-time monitoring of specification modifications. The UI presents every ADaM specification modification clearly through color-coding, highlighting, and detailed change logs, allowing users to quickly identify version differences and understand implications without sifting through complex documents. At its core, ACIRA leverages data visualization techniques to provide an intuitive and comprehensive view of specification evolutions over time via interactive timelines and side-by-side comparisons, enhancing development team transparency and empowering ADaM programmer to make prompt, informed decisions by visualizing how modifications impact the overall project.

Figure 2 shows a screenshot of the overview of specification change. When the "Review Changes" button is clicked, the specific content of specification modifications is presented, divided into four subpages: Dataset, Variable, ValueLevel, and Codelist. Each page uses different colors to distinguish Add, Update, and Delete operations. Records can be easily filtered by "Diff Type", and a search box allows filtering records containing any specified characters. When the "Show Details" slider is toggled on, it highlights the specific content of specification modifications compared to the latest previous version.

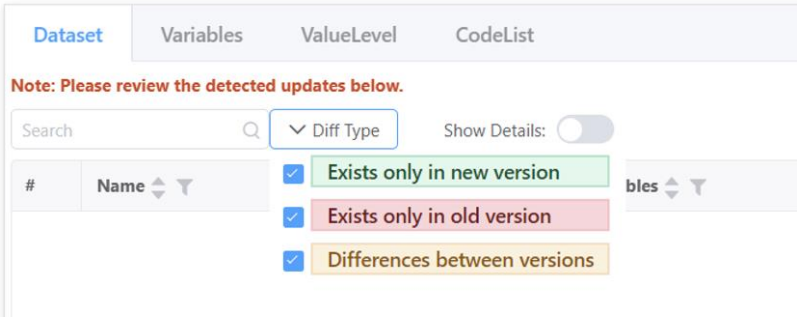


Figure 2. Overview of specification change

Figure 3 exemplifies this:

1. The first row shows an example of "Show Details", review can look at highlighted "Test update yy" text.
2. The second to fourth row demonstrates an example of adding PARAMCD at the ValueLevel,
3. The fifth row shows an example of deleting PARAMCD at the ValueLevel.

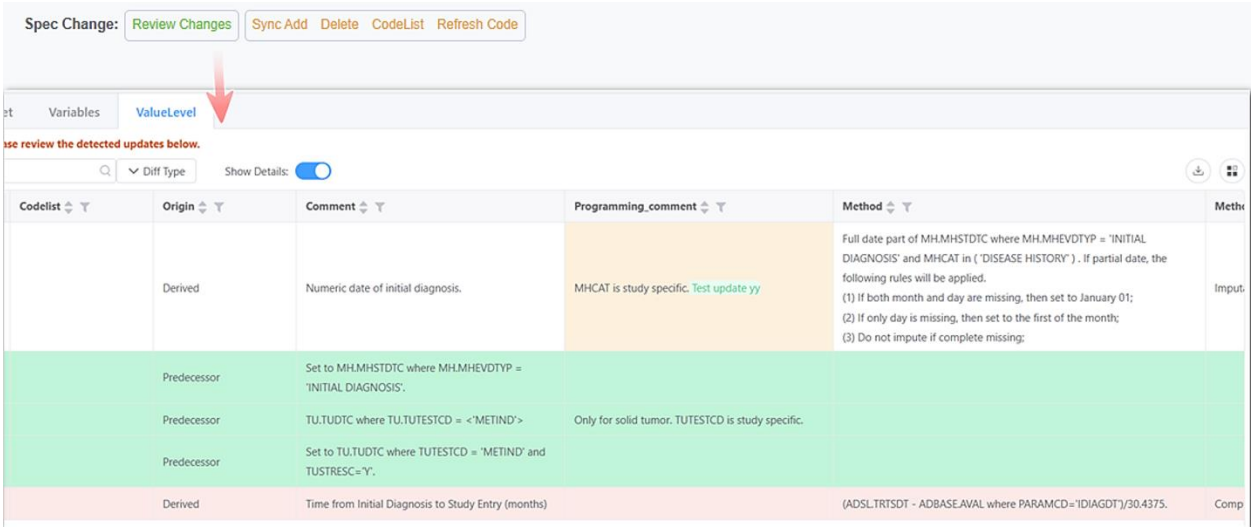


Figure 3. Example of cases for update, add and delete.

Figure 4 shows an example of ACIRA's actions when additions or deletions occur at the upstream Dataset, Variable, or ValueLevel layers. By clicking "Sync Add", "Delete", or "CodeList" buttons, users can synchronize records of additions or deletions from upstream specifications into the ACIRA part in the same database.

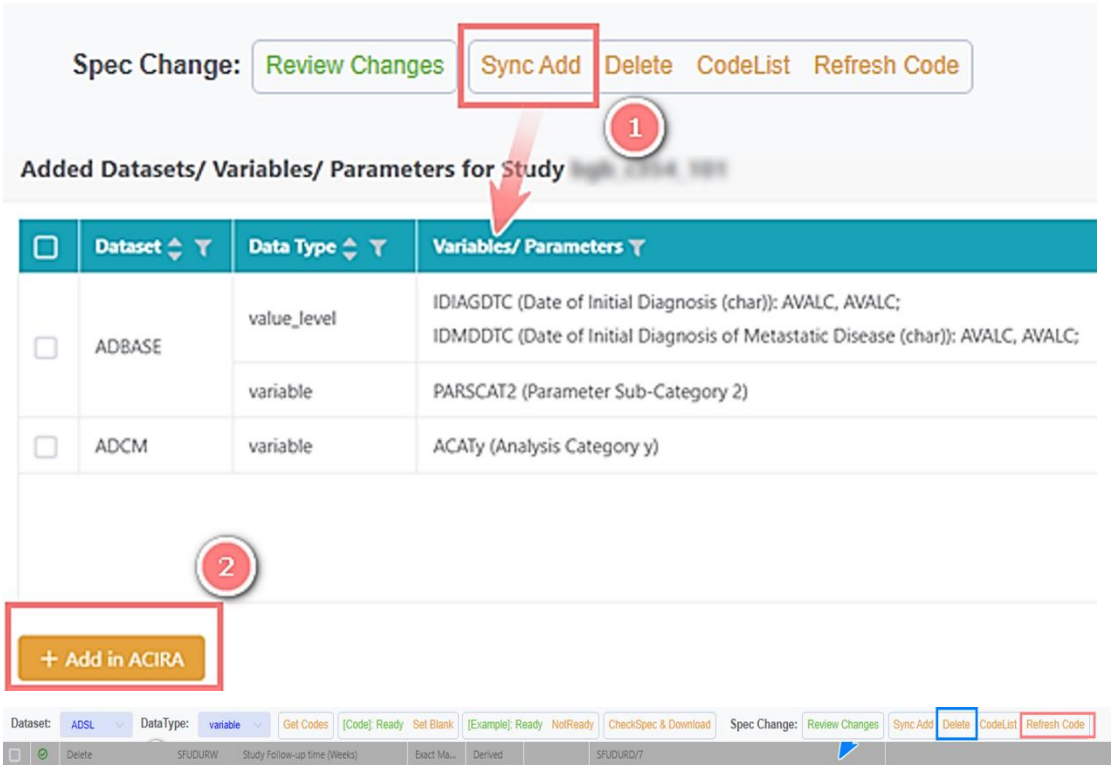


Figure 4. Example of synchronizing add, delete and codelist.

EMAIL ALERT (待探索出来后再加这部分)

CODE SEAMLESS SYNCHRONIZATION, AUTOMATED GENERATION AND MANAGEMENT

Another core advantage of ACIRA lies in its one-click code synchronization and AI-driven code automation. In contrast to traditional methods where significant delays often exist between specification modifications and code updates, ACIRA ensures

codes are always synchronized with the latest specifications through real-time AI processing. Once ADaM specifications are updated, the system automatically analyzes modifications and user can confirm modifications and generate corresponding AI code, eliminating manual coding efforts, reducing repetitive task time, and minimizing human error—AI algorithms strictly adhere to updated specs to enhance accuracy. This synchronization brings far-reaching value: it avoids misusing incorrect codes due to delays, while shortening coding time and improving efficiency by reducing iterative communication and manual update processes. Particularly in complex projects with numerous code modules, this feature replaces time-consuming, error-prone manual coding with automated precision, ensuring seamless alignment between specifications and generated code.

For scenarios where code refresh is required due to specification modifications, there are two operation modes:

One is batch processing, as shown in Figure 5. First, filter records where the "Spec Change" column equals "Update", select all filtered records, and click the "Refresh Code" button. After the AI generates code based on the new specification content, the "Update" status in the "Spec Change" column will be displayed as "Update (AI code exists)". This means the AI uses method, comments and confirmed codes from previous specifications as example to generate new code according to the method and comment in the new version specification.

Clicking on each variable will navigate to the workspace page shown in Figure 6. The upper part uses different colors and strikethroughs to visually present the comparative modifications in format, label, method, comment, programming comment and code compared to the previous version. In the code area below, the new code generated by AI on the left and the previous code on the right clearly highlight the modifications between the two versions. If the AI-generated code is not suitable, users can freely adjust it. When users confirm that the specification modifications are synchronized with the code, clicking "Sync SpecUpdate" will remove the synchronized cases from the specification change overview page, which also applies to add and delete scenarios.

The other operation mode is non-batch processing with confirming one-by-one: select a specific variable to enter the workspace page, then click "Sync SpecUpdate", it will generate a new AI code and display comparisons following the same logic as above.



Figure 5. Example of all batch processing.

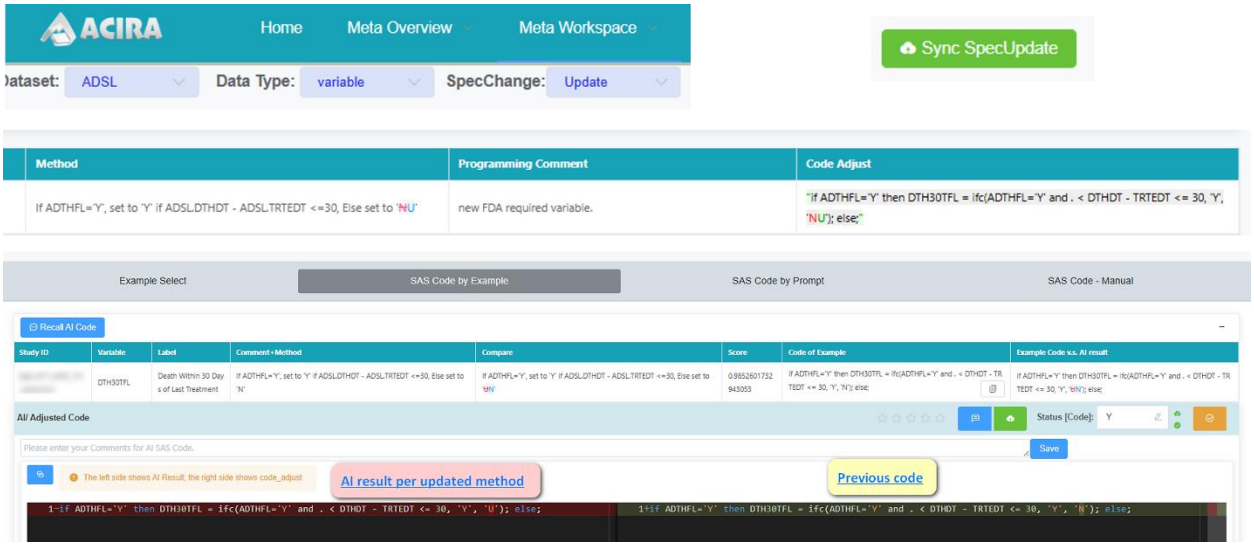


Figure 6. Example of one-by-one batch processing.

COMPARE SPECIFICATION CHANGE AND CODE CONSISTENCY CHECK

The UI for comparing the consistency verification within specification modifications and code is similar to the UI of "Spec Change", but its purpose is to overall review of modifications in both specification and code during selected time period, such as before delivery. It allows reviewers to select different dates freely and conduct a phased, retrospective overview of whether specification modifications and corresponding code updates within a specific period are as expected and double confirm whether the synchronization is correct. This provides a "double safeguard" against operational errors by cross-validating textual specifications with generated code.

Figure 7 shows the UI, where "Old Version" serves as the base date for comparison, and "New Version" serves as the compare date.

The timeline selector supports granular date range adjustments, enabling reviewers to focus on critical milestones. After selecting the dates and clicking the "Compare" button, all specification modifications and corresponding codes from the base date to the compare date are displayed at four levels: Dataset, Variable, ValueLevel, and Codelist.

Additionally, a new column "spec_change" is added to record whether users have synchronized the specification modifications in ACIRA system. The UI also includes a batch export function, allowing users to generate detailed excel reports with new version specification content and freely select columns for exporting, which can be attached to quality assurance documentation for audit trails. This comprehensive verification workflow ensures that no specification-code discrepancies go unnoticed throughout the project lifecycle.

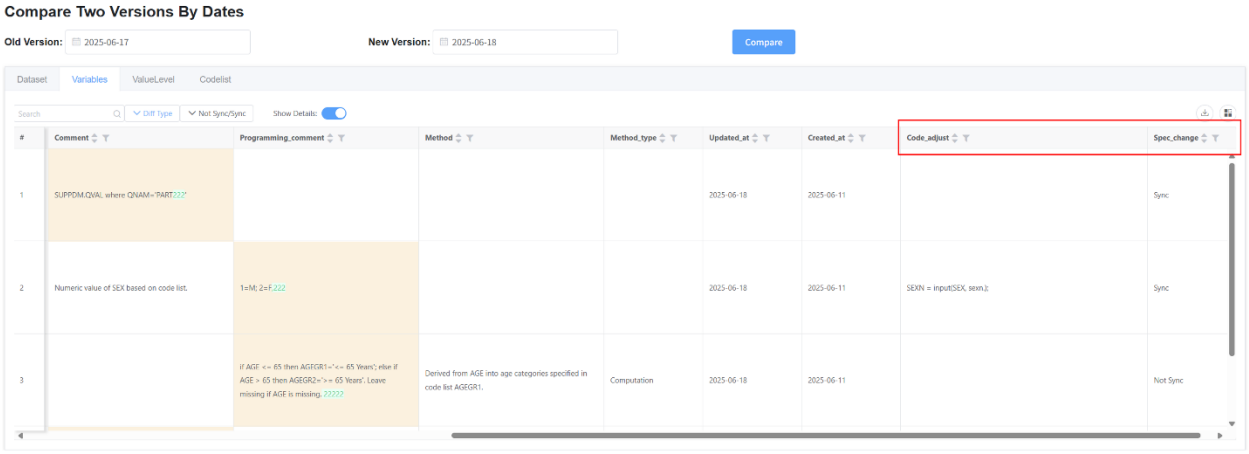


Figure 7. Example of consistency check

THE DASHBOARD: A WINDOW TO CROSS-PROJECT INSIGHTS

Dashboard is different from specification compare/ synchronization in develop console, it empowers users to visually track horizontal comparisons of diverse metrics across projects, offering a comprehensive overview of progress, productivity, and quality. Customizable to prioritize metrics aligned with specific needs, it facilitates real-time monitoring of project timelines, operational efficiency, and quality control workflows. Reviews can oversee the number of modifications by study-version-dataset, and can filter modifications by category, such as total/ open/ closed or row add/ update/ delete.

Each dashboard component delivers granular insights as in figure 8, including:

1. Avg. Resource Save: In-depth evaluation of resource consumption and performance per dataset.

2. Tot. # Datasets/Vars/Param-Vars: Summarized counts across three hierarchical layers.

3. Avg. Completion Rate: Project-level and dataset-level averages for progression.

4. Avg. Semi-Auto Degree: Project/dataset metrics for exact-match rates, prompt code efficiency, and example implementation.

5. Check Points/Tot. # Found Issues: Global specification checkpoints with summarized issue counts per criterion.

6. Spec Issues Open/Resolved/Tot. #: System-recorded specifications issues tracked per project.

7. Spec Change Not Sync/Sync/Tot. #: Project-wise statistics on not sync/sync specification modifications (with percentage counts), top-frequency dataset distributions at the "Dataset" level per project, aggregated summaries categorized by project, dataset, and variable. For selected projects, developers can analyze specification change synchronization between a base date and comparison date.

The system continuously enhances metadata validation checks, enabling developers to identify specification discrepancies during code generation—streamlining communication and quality assurance. leveraging multi-dimensional visibility that allows reviewers to gain holistic cross-project perspectives to aid decision-making. All data is downloadable for advanced analytical workflows.

Figure 9 is an example of dashboard spec change overview. You can customize various visualization perspectives according to requirements.

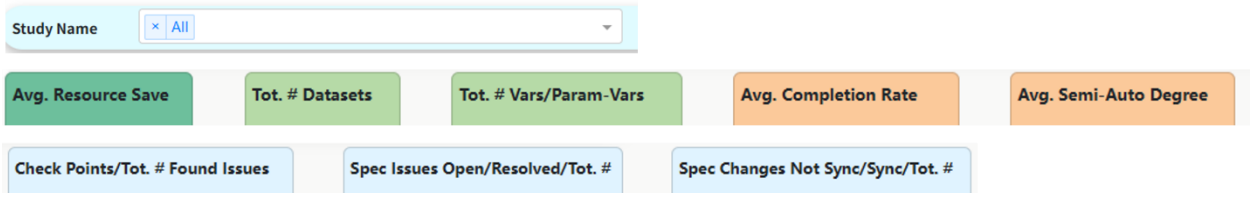


Figure 8. Dashboard items category

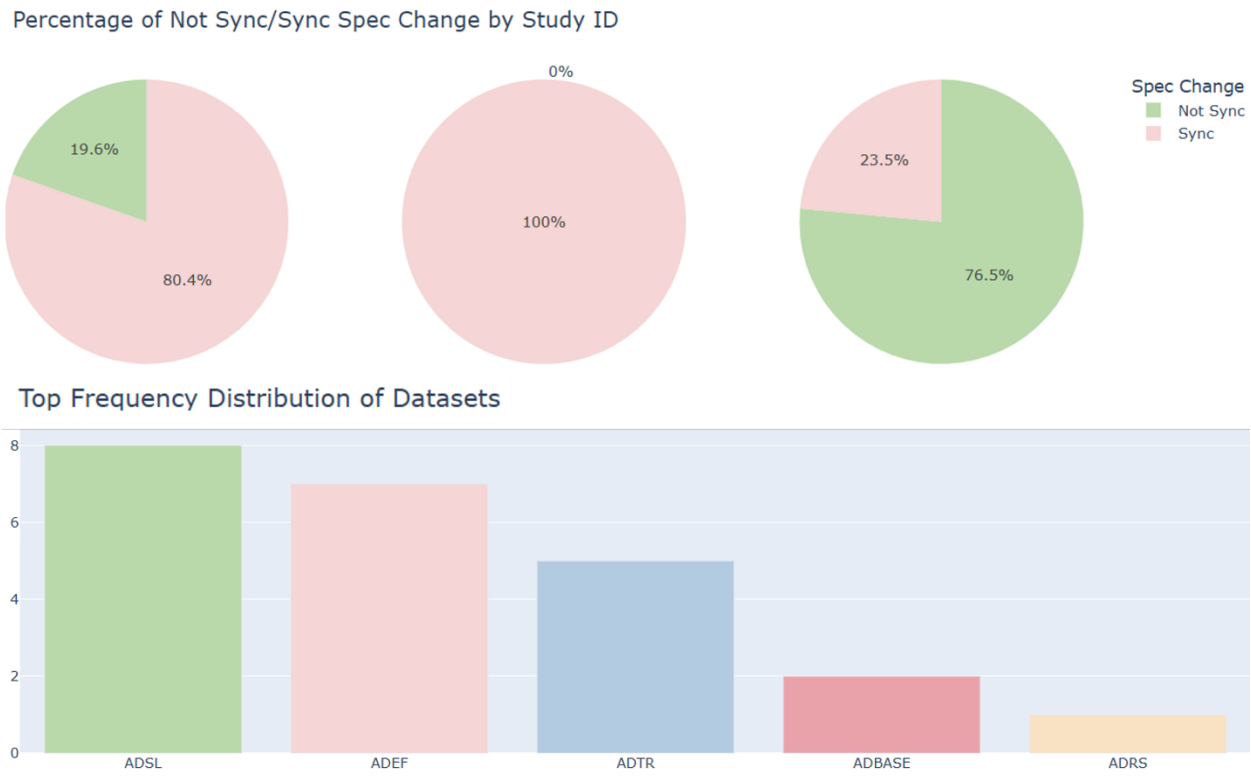


Figure 9. example of dashboard spec change overview

HOW TO QUERY AND COMPARE DATA FOR SPECIFICATION CHANGES

Data querying needs to consider three hierarchical levels simultaneously: dataset, variable, and paramcd. This includes data from the upstream SpecMaster and its corresponding version tables, as well as data from the downstream ACIRA and its respective version tables.

The detailed steps are as follows:

Step1: Query data from current

Select current specification information from SpecMaster parent table (adam_dataset, adam_variable, adam_paramcd), including dataset, variable, paramcd, type, label, length, format, codelist, origin, comment, method, time information (updated_at and created_at), etc.

Select current ACIRA metadata from ACIRA parent table (acira_dataset, acira _variable, acira _paramcd), including code, operation type, time information (updated_at and created_at), link information (parent_id, parent_transaction_id, transaction_id). ACIRA metadata can link to SpecMaster metadata with parent_id/ parant_transation_id).

Step2: Query data from version

Select history metadata from SpecMaster version table (adam_dataset_version, adam_variable_version, adam_paramcd_version) and ACIRA version table (acira_dataset_version, acira _variable_version, acira _paramcd_version) during base and compare snapshot date. ACIRA codes in ACIRA version table can also link to history SpecMaster meta. Get the last records which are linked to SpecMaster meta if there're multiple ones. For existing records, without operation_type=2 (belong to delete), get the last record with created_at or updated_at before the snapshot date. For deleted records, keep them only if the deletion datetime is after

snapshot (get deletion datetime from transaction table)

Step3: Compare data between base and compare snapshot date

For “Update”: keep all the updated records

For “Delete”: only for deleting variables/ parameters in existing datasets, without showing deleting whole data.

For “Add”: only for adding variables/ parameters which were not in the initial version of dataset, without showing adding new datasets for the first time.

If sync hasn’t been done (no match), highlight as “Not Sync”, otherwise as “Sync”.

CONCLUSION

ACIRA revolutionizes ADaM specification and code management by addressing traditional inefficiencies through an AI-driven, SQLAlchemy-powered framework, such as communication overload, alignment risks, and version tracking gaps. By integrating with upstream SpecMaster via hierarchical foreign key relationships, ACIRA enables real-time version control, seamless code generation, and cross-validated consistency checks. Its intuitive UI facilitates one-click code synchronization, color-coded change visualization, and phased retrospective reviews, while the dashboard offers cross-project metrics for resource optimization and quality assurance. Leveraging SQLAlchemy-Continuum for transaction-level auditing, ACIRA ensures traceability from specification modifications to AI-generated code, creating a unified workflow that minimizes manual errors and enhances regulatory compliance and audit trail. This integration of advanced database architecture, AI automation, and user-centric design establishes ACIRA as a transformative solution for clinical data management, driving efficiency and accuracy in pharmaceutical research.

REFERENCES

SQLAlchemy-Continuum. SQLAlchemy-Continuum — SQLAlchemy-Continuum 1.3.6 documentation

CONTACT INFORMATION

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

Nana Yang

BeOne

nana.yang@beigene.com

Tingting Zeng

BeOne

tingting.zeng@beigene.com

Yaohui Zhu

BeOne

yaohui.zhu@beigene.com

Yanan Sui

BeOne

yanan.sui@beigene.com

Mingliang Fan

BeOne

mingliang.fan@beigene.com

You Li

BeOne

you.li@beigene.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of

SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies