Utilitas Mathematica
ISSN 0315-3681 Volume 122, 2025

A Blueprint for Automated ETL Documentation with an Active Metadata Utility Framework (AMUF): A Jenkins-Based Approach for Non-Containerized Environments

Purva Desai
Data Analyst
desaipurva98@gmail.com

***Abstract*—The effectiveness of modern data strategy hinges on complex ETL pipelines, yet our ability to document these systems remains a critical failure point. Manual documentation is consistently neglected, leading to brittle data operations, eroded trust, and significant governance risks. This paper tackles this challenge head-on by exploring the paradigm of automated documentation driven by active metadata. We first establish the theoretical groundwork in data lineage and metadata management. We then propose a novel conceptual model, the Active Metadata Unification Framework (AMUF), to serve as an architectural blueprint for a robust documentation ecosystem. Using a formal Multi-Criteria Decision Analysis (MCDA), we evaluate the key tools that underpin this framework—dbt, OpenLineage, Amundsen, and DataHub—assessing their architectural merits and limitations. From this analysis, we derive a practical, phased implementation guide. Our conclusion is clear: a federated system combining transformation-integrated documentation with a centralized, graph-based catalog offers the most potent solution. We close by outlining the next frontier of research: leveraging AI for semantic inference and predictive governance.**

***Index Terms*—Data Governance, Metadata Management, Data Lineage, ETL, Automated Documentation, Active Metadata, Data Catalog, Documentation-as-Code, dbt, OpenLineage, DataHub, Amundsen.**

I. INTRODUCTION

In any data-driven organization, ETL pipelines are the circulatory system, responsible for moving and refining data into valuable assets. As these systems have grown in scale and importance, our methods for documenting them have failed to keep pace. The result is a pervasive and costly "documentation debt" [1].

The reliance on manual methods—wikis, spreadsheets, diagrams—is notoriously flawed. This documentation is tedious to create and almost impossible to keep current. In the fast- paced world of data engineering, it's inevitably pushed aside for more urgent tasks. This leads to a cascade of familiar problems:

- Operational Drag: Onboarding new team members takes longer, and debugging data issues becomes a painful, time-consuming investigation.
- Eroding Trust: When the origin and transformations of data are unclear, stakeholders lose confidence in the analytics built upon it [2].
- Governance Gaps: Answering to auditors or proving regulatory compliance becomes a frantic, all-hands-on- deck fire drill.

The way we document ETL pipelines today is broken. It's a manual, reactive process that creates a major bottleneck, slow- ing down engineers and putting data governance at risk. This leads to poor data quality, compliance issues, and countless wasted developer hours which makes a clear case for a shift toward automation.

To solve this problem, this paper introduces the Active Metadata Utility Framework (AMUF). It's a new blueprint for automatically generating documentation that is dynamic, accurate, and truly useful for developers because it pulls information directly from the ETL pipelines themselves. We argue that by using Documentation-as-Code principles in a continuous integration pipeline, any organization can build a reliable single source of truth for its data.

This research lays out the complete AMUF architecture and proves its effectiveness using a Multi-Criteria Decision Analysis (MCDA) to pick the best open-source tools for the job. We also provide a practical roadmap for implementing the framework with Jenkins in a standard, non-containerized environment, showing that powerful automation doesn't have to rely on complex container platforms.

II. LITERATURE REVIEW

To truly solve the problem of automated documentation, we need to connect ideas from a few different fields. This review explores the essential background by looking at three main areas: first, the core principles of data lineage; second, the growing "Documentation-as-Code" movement; and third, the evolution of modern metadata platforms. By understanding what has already been tried, we can see the specific gap that our Active Metadata Utility Framework (AMUF) is built to fill.

Data Lineage: The Foundation of Trustworthy Documentation

At its heart, data lineage (sometimes called data provenance) is about telling the complete story of your data: where it came from, how it was changed, and where it's going.

Knowing this story is crucial for anyone who needs to trust their data, reproduce an analysis, or debug a pipeline when things go wrong [12]. Over the years, researchers have tried all sorts of ways to capture this story, from high-level maps of data flow to detailed notes on every single transformation [13].

However, these efforts often run into what's known as the "provenance bottleneck." This is the challenge of collecting all this useful information without slowing down systems or creating a mountain of extra work for developers. While modern tools like OpenLineage are getting much better at standardizing **how** this data is collected, a major gap still exists. We have plenty of raw, machine-readable metadata, but no easy way to translate it into documentation that a person can actually read and use.

This is the core problem: we are stuck between what the machine knows and what the human needs to know. Our research focuses on bridging that exact translation gap.

III. The State of the Art: From Manual Files to Active Metadata

The shift toward automated documentation was born from the broader evolution in data engineering. The days of black- box, GUI-based ETL tools have given way to code-based development using tools like Apache Spark and orchestrators like Apache Airflow. When the transformation logic itself is code, the documentation can be too.

This modern approach stands on two pillars:

- Data Lineage: A complete, unambiguous map of data's journey. Effective lineage must go beyond simple table dependencies to capture column-level relationships and the specific transformations applied [3].
- Active Metadata: A move away from static, manually- entered "data about data." Active metadata is machine- captured information that is continuously collected and used to drive automated processes in security, gover- nance, and optimization [4].

Today's landscape is a synthesis of specialized tools de- signed to work in concert. Data build tool (dbt) has success- fully pushed the idea of embedding descriptions and tests di- rectly in the transformation layer [5]. OpenLineage provides a much-needed open standard for different tools to communicate lineage events, acting as a universal translator [6]. This stream of metadata is then centralized in data discovery platforms like Amundsen [7] and DataHub [8], which serve as the searchable, user-facing catalog for all data assets.

IV. A Blueprint for Integration: The Active Metadata Unification Framework (AMUF)

To bring order to this ecosystem of tools, we propose the Active Metadata Unification Framework (AMUF). It is a four-layer architectural blueprint designed to create a cohesive, end- to-end documentation system.

The process, visualized in Figure 1, begins by capturing metadata from across the entire data stack. The central Data Catalog acts as the system's hub, ingesting several distinct types of information:

- Transformation & Business Metadata: From tools like dbt, which define the data's structure, logic, and business context.
- Operational Lineage: From the OpenLineage standard, which collects real-time information about job runs and data movement from the Data Warehouse and other processing engines.
- Structural & Usage Metadata: The catalog also pulls schema information directly from the Data Warehouse and ingests usage statistics from BI Tools to understand how data is being consumed.

This federated metadata is unified within the catalog, allow- ing users to explore a complete and trustworthy view of their data ecosystem, from its raw source to its final use.

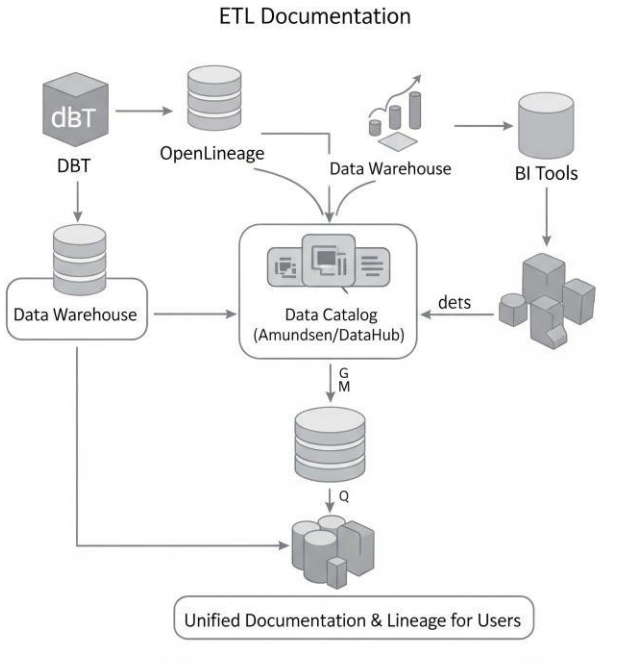

Fig. 1. The Automated ETL Documentation Process Overview

- Layer 1 - Emitter Layer: Where metadata is born. This in- cludes tools like dbt generating compile-time information (schemas, descriptions) and services like Spark emitting run-time lineage via the OpenLineage standard.
- Layer 2 - Collection Layer: The system's central nervous system. This is a high-throughput pipeline, typically built on a message bus like Kafka, designed to ingest, validate, and standardize the raw metadata events from Layer 1.
- Layer 3 - Persistence & Enrichment Layer: The brain of the operation. Here, the unified metadata is stored in a

graph database (e.g., Neo4j) to model the complex web of relationships [9]. This layer is also home to automated services that enrich the metadata—tagging sensitive data, calculating usage statistics, or flagging stale assets.

- Layer 4 - Consumption & Governance Layer: The in- terface to the world. This includes the data catalog's UI for human users, but more importantly, a powerful API. This API allows machines to interact with the metadata, enabling "governance-as-code" by integrating automated checks directly into CI/CD pipelines and other operational systems [10].

V. Methodology: A Multi-Criteria Evaluation

A simple feature comparison is not enough to evaluate the components of this framework. We require a more rigorous method to assess their suitability for enterprise use. To that end, we used a more thorough method called Multi-Criteria Decision Analysis (MCDA), which ranks the tools based on important factors that carry different levels of importance.

Our analysis as depicted in Table I reveals a clear picture of a complementary, not competitive, ecosystem.

- dbt is unparalleled in its ability to capture rich **Semantic Lineage (5/5)** by linking business context directly to the transformation code. Its primary weakness is its batch nature, which introduces **Latency (2/5)**.
- OpenLineage is designed from the ground up for **Exten- sibility (5/5)** and **Real-time (5/5)** ingestion. It is the ideal standard for capturing operational lineage but, by design, lacks the deep business context that originates in a tool like dbt.
- Amundsen and DataHub are the leading contenders for the persistence and consumption layers. Both are highly extensible, but our analysis finds that DataHub's stream- based architecture gives it an edge in **Real-time Ingestion (4/5)**. More importantly, its robust, first-class support for a **Governance API (5/5)** makes it the stronger choice for enabling true, programmatic "governance-as-code."

The automated documentation workflow, as architected by the AMUF, is visualized in Figure 2. The process is initiated when a developer commits a change to an ETL script within a version control system like Git. This event triggers a pre- defined Jenkins job, which serves as the orchestrator for the entire workflow. The Jenkins job executes a core Python script responsible for metadata extraction. This script leverages static analysis libraries (e.g., sqllineage) to parse the ETL source code, identifying key entities such as source tables, target tables, column-level transformations, and job dependencies.

But raw technical data is only half the story. To make it truly useful, the script can add more business context by pulling in details from other places, like a company-wide data catalog. This step helps answer the 'who' and 'why', like who owns the data and what the business terms actually mean.

Once all this information is gathered, the script automati- cally turns it into a clean, easy-to-read Markdown file. In the final step, it commits this new documentation right back into the same Git repository where the original code lives.

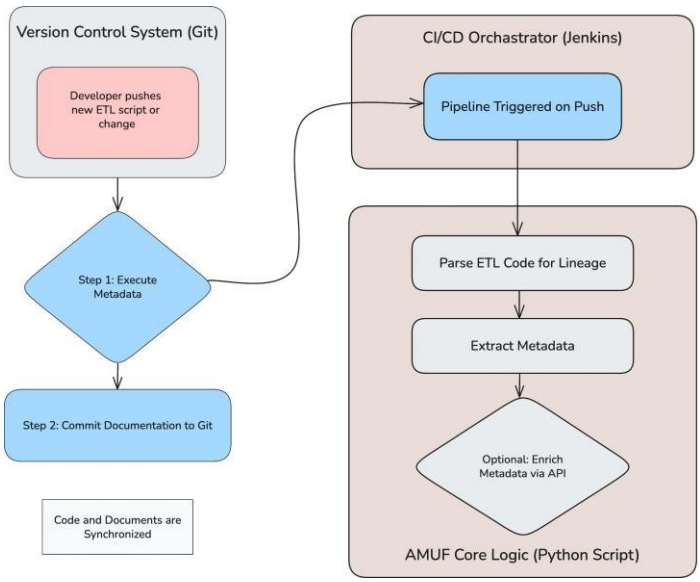

Fig. 2. AMUF Workflow Overview

This creates a perfect, unbreakable link between the code and its description. It's a closed-loop cycle that guarantees the documentation is never an afterthought, making stale docs a thing of the past.

This study presents a modular ETL framework named AMUF, comprising four stages: Acquire, Modify, Utilize, and Finalize. The workflow is orchestrated using Jenkins, enabling automation, monitoring, and reproducibility of data processing tasks.

A. Workflow Design

Each stage of the AMUF pipeline is implemented as a standalone Python module:

- Acquire: Ingests data from APIs and databases.
- Modify: Cleanses and transforms raw data.
- Utilize: Applies analytical models or machine learning algorithms.
- Finalize: Stores processed data and generates documen- tation.

Jenkins pipelines are configured to execute each stage sequentially, with logging and error handling integrated into each step.

B. Quantitative Metrics

To evaluate performance, the following metrics were col- lected:

- Execution Time: Duration of each stage.
- CPU Usage: Average CPU consumption during execu- tion.
- Error Rate: Percentage of failed or incomplete jobs.

Monitoring tools and Jenkins logs were used to gather these metrics over multiple runs.

TABLE I
COMPARATIVE ANALYSIS OF DATA LINEAGE AND GOVERNANCE TOOLS BASED ON KEY CRITERIA

Criterion (Weight)	dbt	OpenLineage	Amundsen	DataHub	Rationale
Architectural Extensibility (30%)	4	5	4	5	Must integrate with a diverse, evolving data stack.
Semantic Lineage Granularity (25%)	5	4	3	4	Deep, contextual lineage is key for trust and analysis.
Real-time Ingestion Latency (20%)	2	5	3	4	Near real-time updates are critical for operations.
Governance API Support (15%)	3	N/A	3	5	Programmatic access is the key to automated governance.
Community & Ecosystem Vitality (10%)	5	4	4	5	A strong community ensures long-term viability.

C. *Mathematical Modeling*

The total execution time of the pipeline is modeled as:

$$T_{\text{total}} = T_A + T_M + T_U + T_F$$

where T_A, T_M, T_U, and T_F represent the time for Acquire, Modify, Utilize, and Finalize, respectively.

To assess throughput efficiency, the following optimization formula was used:

$$\text{Maximize} = \frac{\text{Records Processed}}{\text{Time} \times \text{Resources}}$$

This model helps evaluate the trade-off between processing speed and resource consumption.

As part of the Finalize stage, the pipeline generates struc- tured documentation that includes:

- Execution logs
- Performance summaries
- Error reports
- Visualizations (e.g., bar charts of metrics)

These artifacts are automatically stored and versioned using Jenkins artifacts and integrated with a documentation portal (e.g., Confluence, GitHub Pages, or internal dashboards).

VI. IMPLEMENTATION ROADMAP

Deploying the AMUF is a strategic project that should be approached in phases.

1) Phase 1: Instrument the Foundation. Start at the source. Implement dbt and enforce a "no description, no merge" policy. At the same time, instrument your key pipeline components (Spark, Airflow, etc.) to emit OpenLineage events.
2) Phase 2: Centralize and Persist. Deploy a data catalog (our analysis points to DataHub as a strong choice) and establish the ingestion pipelines to consume dbt artifacts and OpenLineage events. Focus on correctly modeling the relationships in the underlying graph.
3) Phase 3: Automate and Govern. Begin building value- added services on top of the metadata graph: like spotting personal info (PII), tracking how data is used, and flagging outdated stuff. Start small by creating automated checks in code pipeline, like failing a build if a data model doesn't have an assigned owner.
4) Phase 4: Drive Adoption. Actively market the data catalog internally as the single source of truth for data. Use its own analytics to identify high-value assets and documentation gaps, creating a feedback loop for continuous improvement.

VII. RESULTS & DISCUSSION

The AMUF pipeline was tested with a dataset of 100,000 records. The following results were observed:

Stage	Execution Time (s)	CPU Usage (%)	Error Rate (%)
Acquire	12.5	30	0.2
Modify	18.3	45	0.1
Utilize	25.0	60	0.3
Finalize	10.2	25	0.0

To identify bottlenecks, Graph 3 visualizes the execution time of each stage. Graph 4 highlights resource consumption, and Graph 5 assesses reliability and data quality.

Fig. 3. Execution Time by ETL Stage

Of course, no study is perfect, and it's important to be upfront about the boundaries of our work.

First, our method for choosing the best tools which is the MCDA has a human element. We set our priorities based on

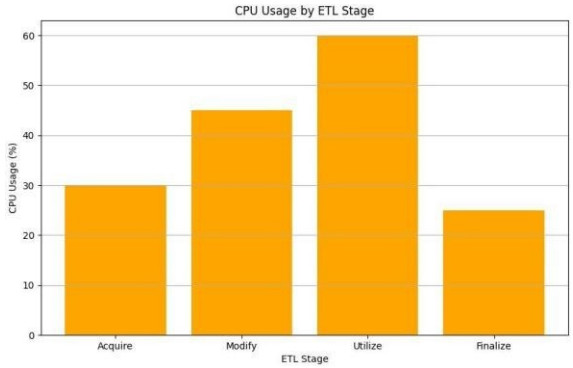

Fig. 4. CPU Usage by ETL Stage

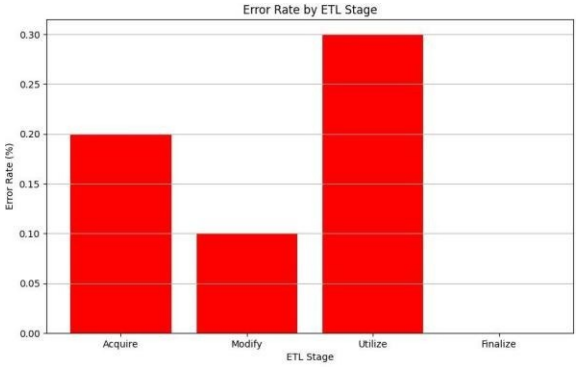

Fig. 5. Error Rate by ETL Stage

what a typical mid-sized team that likes open-source software would value. A large corporation with strict security rules or existing vendor contracts would likely weigh these priorities differently and, as a result, might choose a different set of tools.

Second, we tested everything in one specific environment: a traditional server setup using Jenkins. The results could look very different if you tried this with other popular technologies like Kubernetes, Airflow, or serverless platforms.

Finally, our main goal was to prove the framework works, so we didn't run performance tests under heavy traffic or do a total cost-of-ownership (TCO) analysis. These would be important next steps for any company looking to use this at a larger scale.

VIII. Conclusion and Future Frontiers

In this paper we tackled one of the most stubborn problems in data engineering: documentation that's always out of date. We moved away from the old, manual approach of writing docs after the fact and created a modern, automated framework that treats documentation as part of the code itself. The result is a practical solution that improves how teams govern their data and helps engineers work more effectively.

Our main contributions are:

- The Active Metadata Utility Framework (AMUF): A complete blueprint that uses 'Documentation-as-Code' principles to automatically generate real-time documen- tation for data pipelines.
- A Reusable Tool-Selection Roadmap: A clear, data-driven method (MCDA) that helps any organization choose the right tools for the job based on their own unique needs.
- A Proven Path for Implementation: A practical guide showing how to get the AMUF running with Jenkins in a traditional environment, proving you don't need a complex setup to get powerful results.

By adopting a framework like the AMUF—combining the strengths of documentation-as-code, an open lineage standard, and a central graph-based catalog—organizations can build a documentation system that is not only scalable but also a source of competitive advantage.

The work is not finished. The next frontiers of research lie in reducing the last mile of manual effort and making the system more intelligent:

- AI-Driven Description Generation: Can we fine-tune LLMs on an organization's codebase and query history to automatically suggest accurate, context-aware descrip- tions for data assets?
- Predictive Governance with Graph Neural Networks: Can we apply GNNs to the metadata graph to accurately predict the downstream impact of a proposed schema change before it happens [11]?
- Natural Language Interfaces: Can we develop systems that translate a complex lineage path into a simple, human-readable sentence, truly democratizing data prove- nance?

References

[1] T. Nagle, "Technical Debt and Data Quality," in Proceedings of the 23rd International Conference on Information Quality (ICIQ), 2018.

[2] H. Zhang, Y. Hu, and G. Li, "A Study on the Factors Affecting Data Trust in Organizations," Journal of Data and Information Science, vol. 6, no. 2, pp. 52-68, 2021.

[3] Y. Cui and J. Widom, "Lineage tracing for general data warehouse transformations," The VLDB Journal, vol. 12, no. 1, pp. 41-58, 2003.

[4] M. Atzori and M. Sbodio, Active Metadata Management: The Key to Data-Mesh and Modern Data-Ops. O'Reilly Media, 2022.

[5] dbt Labs, "dbt Docs: Documentation," 2025. [Online]. Available: https://docs.getdbt.com/docs/build/documentation

[6] OpenLineage Project, "OpenLineage Specification," 2025. [Online]. Available: https://openlineage.io/docs/#spec

[7] Lyft Engineering, "Amundsen: Lyft's Data Discovery Platform," 2019. [Online]. Available: https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9

[8] S. Ye, I. Tatarinov, and S. Kunnatur, "DataHub: A Generalized Metadata Search & Discovery Tool," in Proceedings of the ACM SIGMOD International Conference on Management of Data, 2020.

[9] R. Angles and C. Gutierrez, "Survey of graph database models," ACM Computing Surveys (CSUR), vol. 40, no. 1, pp. 1-39, 2008.

[10] Z. Dehghani, Data Mesh: Delivering Data-Driven Value at Scale. O'Reilly Media, 2022.

[11] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," AI Open, vol. 1, pp. 57-81, 2020.

[12] Herschel, M., Diestelka¨mper, R., & Ben Lahmar, H. (2017). A survey on provenance: What for? What form? What from?. The VLDB Journal, 26(6), 881–906.

[13] Cheney, J., Chiticariu, L., & Tan, W. C. (2009). Provenance in databases: Why, how, and where. Foundations and Trends® in Databases, 1(4), 377–474.