Paper CT05

Beyond Sign-Off: Practical Tips for Meaningful Reviews by Leads

Jisna Joseph, Catalyst Flex, Kochi, India
Bimal Thomas, Catalyst Flex, Trivandrum, India

ABSTRACT

Final package reviews are critical for ensuring the quality, accuracy, and consistency of clinical deliverables. Experienced leads add value by identifying potential issues, anticipating downstream impacts, and aligning outputs with sponsor expectations. Leveraging AI tools, review dashboards, and custom macros can streamline the process, reduce review time, and enhance quality. This paper highlights key aspects for improving lead-level reviews, including review strategies, the importance of checklists, and automation tools. A centralized dashboard helps run quality checks efficiently and ensures consistency across outputs, whether for internal review, interim delivery, or final CSR submission. This paper explores practical challenges such as mismatched MedDRA versions, outdated programming trackers, missing page breaks in outputs, and unapproved hardcoding. It emphasizes how experienced leads, supported by automation tools and structured review processes, can deliver high-quality outputs efficiently, even under tight timelines.

INTRODUCTION

Ensuring quality in the deadline-driven world of clinical data programming is a significant challenge for lead programmers. With tight timelines, complex standards, and increasing regulatory expectations, reviews often risk becoming a mere formality rather than a meaningful quality checkpoint. A systematic and structured final review process can help leads maintain consistency, prevent critical errors, and ensure that nothing is overlooked before sign-off.

This paper presents practical techniques and tips that enable lead programmers to move beyond routine validation and transform reviews into a value-added process. By incorporating strategies such as maintaining a tailored checklist for each submission type, detecting hard-coding, batching QC activities, resolving log issues, validating MEDRA versions, and ensuring alignment with SDTM and ADaM specifications, leads can significantly enhance review quality. These practices aim to strengthen data integrity and regulatory compliance while reducing the risk of overlooked errors. Implementing a structured approach not only improves deliverable quality but also promotes efficiency, accountability, and consistency across programming teams.

CHECKLIST FOR FINAL REVIEW

A checklist is a structured tool that contains multiple checkpoints or items to verify during a process or delivery. It serves as a guide to ensure that all critical steps are addressed systematically. By following a checklist, team leads can minimize the risk of overlooking important tasks and confirm that every requirement is completed efficiently and accurately.

WHY IS IT IMPORTANT?

Creating a checklist is extremely helpful for making the review process systematic and avoiding accidental omissions, especially when someone is working as a lead for their first study or handling their first study within an organization. A standardized and reviewed checklist helps in clearly understanding the end-to-end process. Additionally, it can be used as a standard training document, not only for lead programmers but also for programmers, to ensure consistent quality.

Checklists can be created in multiple formats, such as client-specific, study-specific, or therapeutic area–specific templates. It is always a good practice to include different categories within a checklist so that leads can easily tailor and use it based on the study's requirements. For example, by preparing separate checklists for each category such as General checks, SDTM, ADaM, and TLFs, lead programmers can filter the relevant set of tasks and ensure that nothing is missed.

A well - structured checklist acts as a quick reference guide, promotes consistency across deliverables, and reduces the risk of last-minute errors. It also serves as a documentation tool for audit readiness, providing clear evidence of the steps taken during validation. Furthermore, checklists encourage a risk-based approach by prioritizing critical checks, allowing leads to focus on areas that have the highest impact on data integrity and compliance.

Incorporating these checklists into routine workflows not only improves quality but also enhances efficiency and accountability within programming teams.

HOW TO CREATE A CHECKLIST?

A checklist can be efficiently created using an Excel sheet with three main columns:

1. Category: This column specifies the type of checks. Common categories include:
    - General: General checks applicable to all deliverables
    - SDTM: Checks related to Study Data Tabulation Model datasets
    - ADaM: Checks for Analysis Data Model datasets

- TLFs: Checks for Tables, Listings, and Figures

2. Checks: This column lists the specific tasks or validations that need to be performed under each category. Notes: This section can be used to pass additional information to leads, such as the availability of macros that support specific checks, or any applications/tools that can be used to perform them.

3. Status: This column tracks the progress of each check. Typical status options include:
   - Done
   - Not Done
   - NA (Not Applicable)

It is important to note that the exact checks may vary across organizations. For example, one organization might have a batch program to run all SAS programs in a directory, while another may use different automation methods. Therefore, checklists should be tailored to the specific processes and tools used within the organization.

GENERAL CHECKS

In all submissions - whether SDTM, ADaM, or TLFs - there are certain general checks that a lead programmer must perform before final delivery. These checks act as a foundation for quality assurance and help ensure consistency across outputs. Including a dedicated category for general checks in the review process is highly beneficial, as it provides a systematic approach and minimizes the risk of overlooking critical steps. By incorporating these checks into every submission workflow, leads can maintain compliance, improve efficiency, and deliver outputs with confidence.

Below are some examples of general checks that can be considered. Based on experience, if there is a possibility that certain issues may occur due to being overlooked, it is always beneficial to include them in the checklist. It is also a good practice to review the checklist periodically and update it by adding new checks or removing any that are no longer relevant, based on specific processes and standards. This approach helps keep the checklist flexible and adaptable.

Category	Checks	Status
GENERAL	The programming tracker is up to date.	DONE
GENERAL	Ensure all clarifications and resolutions are properly documented in the tracker, closed, and that responses are fully addressed or implemented in the programming deliverables.	DONE
GENERAL	Batch programs are updated to include all programs in the directory before sequential batch run.	DONE
GENERAL	Check the timestamp to ensure everything properly updated	DONE
GENERAL	Check for hard-coded values in programs.	DONE
GENERAL	All comparison reports are clean.	DONE
GENERAL	There are no characters outside the ASCII character set (0 to 127)	NOT DONE
GENERAL	Production and validation Logs are clean.	NOT DONE
GENERAL	Sign-off document is completed.	NOT DONE

CHECKS FOR TO SDTM, ADAM AND TFL

In addition to general checks, incorporating category-specific checks for SDTM, ADaM, and TLFs is essential to ensure that all unique requirements for each deliverable type are thoroughly validated before submission.

Below are some checks for SDTM. Similarly, a checklist can be developed for ADaM.

Category	Checks	Status
SDTM	Data issues are tracked and shared.	
SDTM	All partial dates are correctly populated.	
SDTM	Raw variables with length >200 are properly handled.	
SDTM	In TS domain Data Cutoff Date and description are updated.	
SDTM	All required SDTM production datasets were created in the recent batch run.	
SDTM	All required SDTM validation datasets were created in the recent batch run.	
SDTM	All comparison reports are clean.	
SDTM	P21 validation is completed, and all SDTM-related P21 issues have been justified.	
SDTM	Annotated CRF has been finalized, bookmarked and flattened.	
SDTM	Dataset names, Define, CRF, and supplemental documentation are all in lowercase.	
SDTM	Define.xml file has been generated.	
SDTM	SDRG document has been created.	
SDTM	Created supplemental data documentation as required.	
SDTM	Ensure "Inherit Zoom" is applied to all PDF files such as Annotated CRF and supplemental data documentation	
SDTM	P21 validation done with XPT files and Define.xml.	
SDTM	Ensure all required SDTM datasets are in the submission folder.	
SDTM	No datasets with zero observations.	

For TLFs, along with general checks, additional validations should ensure that the Data Cutoff Date is updated to the latest value, the MedDRA version is consistent throughout, and that there are no blank pages and page breaks are properly applied.

SAS MACROS TO STREAMLINE FINAL REVIEW

HARD CODING CHECKS

Hardcoding refers to embedding fixed values directly into the program logic instead of deriving them dynamically from source data. While sometimes unavoidable, excessive hardcoding can lead to inconsistencies, errors during updates, and increased maintenance effort. Therefore, it is essential for lead programmers to review and minimize hardcoding wherever possible. Using automated tools or macros to identify potential hardcoding patterns can be helpful. If certain hardcoding is unavoidable, ensure that it is communicated and documented in the program header or comments, along with a clear justification.

A macro to identify hardcoded subject IDs within SAS programs serves as an example of this approach. Developing a macro using the following methodology enables efficient detection of hardcoded subject-level values.

- The macro scans SAS program files for occurrences of subject identifiers present in study datasets.
- Subject IDs are first extracted from either from SDTM.DM or ADAM.ADSL dataset.
- The macro then checks all SAS programs in a user-specified directory for hardcoded references to these subject IDs.
- So, the Folder path containing SAS programs to be validated and the library where the DM or ADSL dataset is stored could be passed as parameters.
- Upon execution, if hardcoding is detected, the macro generates a detailed diagnostic message that includes:
    - Dataset name
    - Program (script) file name
    - Directory path
    - Exact line number containing the hardcoded value

Below is a sample log message generated when hardcoding has detected in the ADSL dataset at line 323.

```
================================================================================================================
Message    : Hardcoding Found!

Sas File   :  ADAM.ADSL
Script Line:  323

Directory  : c:/test/adam
Script     : if saffl='Y' and subjid='103-002' then dthfl='Y';
================================================================================================================
```

CHECKING LOG ISSUES

Checking log issues after generating individual programs and after a batch run is essential to ensure that no programming errors go unnoticed. A practical approach is to create a macro that can be invoked within each program to automatically scan the log for critical issues. Additionally, after a sequential batch run, the macro can generate consolidated reports summarizing common issues such as ERRORS, WARNINGS, UNINITIALIZED, INVALID, and REPEAT. These reports allow the lead programmer to quickly verify that all programs are free from major problems. Including notes such as “numeric value converted to character” can also help identify potential data inconsistencies. This automated process improves efficiency, supports audit readiness, and is particularly useful during reruns, as it ensures systematic log checks without manual intervention.

ENSURE ALL DATASETS ARE GENERATED IN THE RECENT BATCH RUN

In clinical programming workflows, consistency and completeness are critical. Before validation checks can be performed, it is essential to confirm that all expected datasets and reports have been successfully generated by the batch production program. Missing outputs can lead to incomplete validation and potential compliance issues. This step ensures that the foundation for subsequent checks is solid.

Generating a macro using the following approach will help automate this test:

- Inventory Creation: The macro first retrieves a list of all expected datasets based on study specifications.
- Output Verification: It then compares the actual outputs generated by the batch program against this inventory to identify any missing items.
- Timestamp Validation: The macro captures the last modified date of the batch program and ensures that all outputs reflect the latest updates.
- Issue Logging: Any discrepancies, such as missing or outdated outputs, are flagged for corrective action before validation activities commence.

This macro-driven approach reduces manual effort, ensures completeness, and provides a robust foundation for subsequent validation checks.

ENSURE PROGRAMMING TRACKER IS UPTO DATE

The programming tracker serves as a central reference for monitoring the status of dataset development, validation, and associated deliverables. An outdated tracker can lead to inconsistencies, missed updates, and delays in reporting. Therefore, it is essential to verify that the tracker reflects the latest program modifications and dataset generation dates before proceeding with validation or submission activities.

Generating a macro using the following approach will help automate this check:

- Extract Tracker Information: Read the programming tracker to capture expected program names, last update dates, and associated datasets.
- Compare with Actual Program Metadata: Retrieve the actual last modified timestamps of programs from the study directory.
- Validate Consistency: Ensure that the tracker's update date matches or is later than the actual program modification date. Any discrepancies indicate that the tracker is not current.
- Flag Issues: Log any mismatches for review and corrective action, ensuring that the tracker remains an accurate source of truth for project status.

This automated approach minimizes manual effort, improves compliance, and ensures transparency in programming workflows.

VALIDATION DATASET CREATED AFTER PRODUCTION DATASET

In clinical programming, validation datasets serve as an independent check to confirm the accuracy of production datasets. To maintain process integrity, validation must occur after production is complete. If validation datasets are created before or at the same time as production datasets, it raises concerns about whether the validation reflects the final production output. Therefore, it is essential to verify the sequence of dataset creation.

Generating a macro using the following approach will help automate this check:

- Extract Metadata: Retrieve details of both production and validation datasets from sashelp.vtable, including dataset names and creation dates.
- Map Corresponding Datasets: Match each validation dataset to its corresponding production dataset based on naming conventions (e.g., SDTM → V_<DOMAIN>, ADaM → V_A_<DS>).
- Compare Creation Dates: For each pair, compare the Date Created values. If the production dataset has a creation date greater than the corresponding validation dataset, it indicates that validation was created before production was finalized.
- Flag Discrepancies: Log any such cases for review and corrective action to ensure compliance with validation standards.

This automated check enforces proper sequencing, reduces manual oversight, and ensures that validation reflects the final production output.

MANAGING RAW VARIABLES WITH LENGTH GREATER THAN 200.

Character variables with lengths greater than 200 in raw datasets can lead to potential truncation during SDTM conversion. Monitoring these variables across data cuts is essential to ensure compliance and prevent data loss.

Generating a macro using the following approach will help automate this check:

- Squeeze Variable Lengths: Compute the actual maximum length observed for each raw variable and adjust its length to match this value.
- Create Baseline Inventory: Store the list of variables and their observed maximum lengths for the current data cut.
- Compare Across Data Cuts: When new data is introduced, repeat the process and compare with previous results to detect:
    - New variables exceeding length 200
    - Increased observed maximum length for existing variables
- Flag and Document Changes: Log any new or increased lengths for review and ensure proper documentation and update.

DASHBOARD INTEGRATION

R provides a powerful and flexible alternative for automating and streamlining the final review process in clinical data programming. Unlike SAS, R enables the integration of interactive dashboards, making reviews more systematic, user-friendly, and visually insightful. Similar to SAS macros, modular R functions can be developed for specific tasks such

as checking log issues, validating that production datasets are created after QC, and performing automated comparisons. This modular approach enhances efficiency, reduces manual effort, and ensures consistency across reviews.

R SHINY BASED LEAD REVIEW DASHBOARD

Building on this concept, the lead review dashboard will be implemented using R Shiny, incorporating a secure authentication framework via the shinymanager package to manage user access and enforce data-level permissions. The application will dynamically render content based on user credentials, ensuring that leads access only the relevant studies, metrics, and reports. Core packages such as shiny and shinydashboard will form the backbone of the application, while dplyr, tidyr, and data.table will enable efficient data processing. Interactive tables will be delivered using DT, and visual summaries will leverage ggplot2 with plotly for interactive exploration. The dashboard will also support exporting reviewed outputs in Excel format using openxlsx or writexl.

COMPREHENSIVE REVIEW TABS

To make the review process structured and comprehensive, the dashboard will include separate tabs for different review components:

- **Checklist** Tab: Hosts checklists with dropdowns for status updates, integrating automated checks such as log issues, hardcoding detection, and comparison reports each linked to modular R functions. Manual checks (e.g., verifying clarifications, ensuring responses are documented, and confirming sign-off completion) will also be captured within the same tab to maintain end-to-end traceability.

- **Additional Specialized Tabs (as required):** Dedicated tabs can be created for specific review needs, such as a comparison reports tab to display production versus validation dataset matching, or a logs diagnostics tab summarizing warnings, errors, and unresolved notes.

This flexible design ensures the dashboard adapts to varying study requirements while maintaining a consistent, audit-ready workflow.

NOTIFICATIONS AND TRACEABILITY

To enhance collaboration and close-out efficiency, the solution can incorporate email notifications to distribute final reports and status summaries to study programmers using packages such as mailR or sendR. Combined with Excel exports and role-based access, the framework provides clear traceability, supports compliance, and promotes accountability across teams.

Overall, this modular architecture promotes scalability and reusability across studies, simplifies maintenance for future enhancements, and strengthens review quality. By combining automated checks, interactive visualizations, structured checklists, and secure access, the approach improves efficiency, consistency, and regulatory readiness.

AI ENABLED SUPPORT FOR VALIDATION CHECKS

AI tools such as ChatGPT and Microsoft Copilot can assist with certain non-confidential aspects of validation. While they cannot be used to review datasets, logs, or code directly, they can support lead reviewers in a few practical ways that do not violate confidentiality.

GENERATING QUICK CHECKLISTS

When a project does not have a ready checklist, AI can create a draft structure based on simple prompts.
Examples include generating checklist items for SDTM, ADaM, or TLF reviews.
This helps by:

- Providing an initial template quickly
- Ensuring major categories are not missed
- Allowing the lead to refine the checklist as per study needs

This saves setup time and helps maintain consistency across reviews. Below is an example prompt used to generate a draft checklist for ADaM final review:

```
"Create a draft final review checklist for ADaM datasets, covering data structure,
derivation logic, traceability to SDTM, and compliance with CDISC standards"
```

REVIEWING NON-CONFIDENTIAL METADATA

Although sensitive data cannot be uploaded, AI can analyze screenshots of:

- Folder names
- Dataset names
- Timestamps
- File structures

Using this information, AI can help identify:

- Whether validation datasets were created after production
- Missing or unexpected files in a directory
- Outdated file timestamps
- Naming inconsistencies
- Since only metadata is shared, this remains within confidentiality limits.

CLARIFYING GENERAL DOMAIN QUESTIONS

AI can provide quick, high-level guidance without any study-specific information.
Examples include asking:

- What checks are expected for ADaM BDS datasets?
- What issues commonly appear in AE listings?
- How to verify MedDRA version consistency?

This allows leads to confirm best-practice expectations before validating detailed implementation. While AI can be used as a guiding tool, findings should always be verified against relevant reference documents. AI can also be asked to provide links to related guidance or standards to support verification and ensure accuracy.

USING POWERSHELL AS A PRODUCTIVITY TOOL

PowerShell can be an excellent companion for a lead programmer. It helps reduce both time and manual effort by automating several routine checks and file-handling tasks that are typically part of the final review process.

WHAT IS POWERSHELL?

PowerShell is a Windows-native scripting environment designed for task automation and system management. It combines a command-line shell with a powerful scripting language, letting you read and manipulate files, folders, and metadata (e.g., timestamps) at scale, call external tools, and write outputs to formats like CSV or Excel via COM.

EXAMPLES OF POWERSHELL-AUTOMATED CHECKS AND VALIDATIONS

A. Build Order & Data Freshness Checks
   - Validation-after-Production check: Compare the modified dates of PROD vs. VALIDATION datasets and flag any case where validation is older than the production dataset it's supposed to verify.
   - Batch recency confirmation: Confirm that all required datasets were regenerated in the latest batch run (e.g., all have timestamps ≥ the batch controller script's timestamp).

B. Deliverable Completeness & File Integrity Checks
   - Expected vs. actual inventory: Compare a list of expected SDTM/ADaM/outputs (from CSV/Excel) against the current folder contents to identify missing, extra, or duplicate files.
   - Zero-byte/incomplete files: Detect files with size = 0 or unusually small sizes indicative of failed writes or early terminations.

C. Naming & Structure Hygiene
   - Prefix and pattern enforcement: Check that validation datasets use the expected prefix (e.g., v_), enforce upper/lowercase rules (e.g., ADSL not adsl), and ensure only allowed extensions exist.

D. Tracker Synchronization
   - Auto-update "Date of completion": Read dataset timestamps from the PROD folder and write the latest modified time into the tracker's "date of completion" column per domain.
   - Staleness checks: Flag any tracker entries where the recorded completion date is earlier than the current file timestamp (indicating the tracker is outdated).

E. Log Review & Risk Flagging
   - Bulk log scans: Parse .log files to summarize counts of ERROR, WARNING, UNINITIALIZED, and other risky notes (e.g., "numeric values have been converted").
   - Run-level summary: Produce a one-page CSV highlighting programs that need attention before sign-off.

F. Archival Controls
   - In-place archiving: Copy/move current PROD/VALIDATION contents into Archive under each folder, preserving structure and writing a manifest of archived files.

With the help of AI tools like Copilot or ChatGPT, leads can quickly generate PowerShell scripts tailored to their study needs, even without deep scripting experience. Incorporating PowerShell into the review workflow is a practical way to

automate routine tasks, reduce manual effort, and improve the efficiency of final checks. Its ability to handle file-based validations, timestamp comparisons, and tracker updates makes it a valuable addition to any lead programmer's toolkit.

CONCLUSION

By creating a proper checklist and automating validation checks through macro-based solutions, programming leads can ensure data quality, process consistency, and regulatory compliance across clinical deliverables. These macros provide a systematic approach to verifying critical aspects such as dataset generation, validation sequencing, QC timing, and variable length management, reducing manual effort and minimizing errors. A well-defined checklist helps perform systematic manual reviews, while automation ensures no unexpected gaps occur from the lead's end. AI can assist with generating checklists, interpreting non-confidential metadata, and providing quick domain guidance, while PowerShell automates tasks such as timestamp verification, folder completeness checks, and tracker updates. Together, these practices enable smooth workflows and high-quality deliverables, strengthening confidence in the integrity of clinical data.

REFERENCES

https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/Connect/EU/Hamburg/PAP_CT13.pdf.

ACKNOWLEDGMENTS

We thank our colleagues, mentors, and reviewers for their guidance and constructive feedback throughout the development of this paper. Additionally, we acknowledge the assistance of AI tools, such as Copilot, in organizing ideas and refining content, which contributed to a more efficient writing process.

CONTACT INFORMATION

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

Author Name: Jisna Joseph
Company: Catalyst Flex
Address: 8B Tower 1, TransAsia, Cyber Park Infopark, Phase 2, Kochi, Kerala 682303
Work Phone:
Email: jisna.joseph@catalystcr.com
Website: https://catalystcr.com/

Co-Author Name: Bimal Thomas
Company: Catalyst Flex
Address: Second Floor, Nila Building, Technopark Campus, Thiruvananthapuram , Kerala, 695581
Work Phone:
Email: bimal.thomas@catalystcr.com
Website: https://catalystcr.com/