PharmaSUG 2023 - Paper SI-139

# Integrating Practices: How Statistical Programmers Differ and Align Within User Groups

Valeria Duran, Radhika Etikala, and Haimavati Rammohan, SCHARP at Fred Hutchinson Cancer Center, Seattle, Washington

## ABSTRACT

An expectation in the pharmaceutical industry is that every entity within an organization is aligned on day-to-day practices. Our organization strives to streamline processes to achieve traceable and reproducible outputs for Statistical Programmers coding in different languages, such as R and SAS®. Although the fundamentals in programming practices are aligned, the day-to-day workflow deviates at departmental levels: stakeholders have different expectations for how data is processed and how workflows should proceed. This paper takes a closer look at (1) Statistical Programming practices, specifically with Validation, Version Control, Coding Practices, and Verification, (2) the tools and techniques that have contributed to the success of our analogous workflow, and (3) future work towards further alignment. The insight from this paper has implications for how different entities within a workplace can work towards the same goals while meeting regulatory standards.

## INTRODUCTION

In statistical programming, SAS has been a predominant tool used by researchers and practitioners. SAS has established itself as the gold standard for statistical programming with its proprietary source code and extensively researched and implemented methods. Over the years, SAS has been able to adapt and refine its language to meet the needs and expectations of healthcare industry professionals. However, in recent times, the open-source programming language R has gained significant traction in the industry. R's widespread availability of documentation sources, the ability to create packages tailored to users' specific needs, and the fact that it is free to have all contributed to the increasing popularity of R. Our organization employs a combination of SAS and R statistical programmers, a blend that strengthens our statistical analysis capabilities. These strengths include but are not limited to programming validation, verification, version control, and code maintenance. This paper describes the implementation of the SAS and R programming paradigm in our organization, its impact, and future steps to continue strengthening our organization.

## STATISTICAL PROGRAMMING PRACTICES WITHIN THE ORGANIZATION

In this section, we will focus on the most common programming practices used by statistical programmers within the organization. Programmers are responsible for a variety of tasks, including software validation (Performance Qualification) using SAS and R programming languages, version control, dataset and reports generation, verification, maintaining code, creating, and reviewing data specifications for derived datasets, and maintaining or creating template codes and scripts. We hope that this paper will help statistical programmers and organizations to develop high-quality software systems efficiently and produce reliable datasets and reports for analysis.

Validation of software is an essential aspect of statistical programming, as it ensures that the software meets the required quality standards. Risk assessment is another critical stage, which involves identifying and managing potential risks associated with the software development process. In addition, requirements gathering is a vital step in statistical programming, where the requirements are collected from stakeholders to ensure that the software meets their needs. We will discuss the best practices for developing test cases and test code to ensure the efficient functioning of the software.

Version control is an important aspect of statistical programming, which involves managing changes to the codebase. We will discuss two popular version control systems - Git and SVN - and their benefits for statistical programmers.

Coding practices are also critical in statistical programming, and we will focus on the coding practices in two widely used statistical programming languages - R and SAS. We will discuss the best practices for writing clean, efficient, and maintainable code in these languages.

Finally, we will explore different modes of verification that statistical programmers can use to ensure the quality of their datasets and reports. We will discuss techniques such as data validation, data verification, report verification, and code reviews.

## VALIDATION

There are two common software applications used for data analysis and statistical modeling: R and SAS. In our organization programmers use the same validation methods for both languages to ensure the validity of their results.

There are three distinct stages of validation, IQ (Installation Qualification), PQ (Performance Qualification), and OQ (Operational Qualification); these are important steps in ensuring the quality and reliability of SAS and R code. These stages of validation provide a systematic approach to verifying the code and systems are functioning correctly, and that they are meeting specified requirements. The focus of this paper will be on PQ validation. PQ validation is the second stage of software validation and is used to verify that code and systems are performing as expected. This includes writing expected requirements for the software performance, test cases to demonstrate that each requirement, and test code to verify the success of the test cases and therefore requirements. This ensures that the software package is meeting specified performance requirements, and that the system is responding correctly to different inputs and conditions. SAS programmers specifically, base SAS functions and procedures are used for PQ validation. These functions and procedures are designed to test the performance of the SAS code and systems, and to ensure that they are meeting the requirements and specifications set out in the PQ validation process. In contrast, R programmers use Valtools for PQ validation, designed to help meet regulatory authority requirements, such as those imposed by the FDA, by automating the validation of R software through useful templates and unit testing, as described in the Pharmaceutical Users Software Exchange (PHUSE) Valtools site (Hughes, et al., 2021).

In both languages, PQ validation consists of four components: risk assessments, requirements, test cases, and test code. Each component is vital to validating and ensuring our systems and code are up to standard. The descriptions of each component are as follows:

1. Risk assessments:

As with every validation effort, a risk assessment strategy helps mitigate potential business and compliance risks within a given process, system, or environment. The risk for a requirement/function can broadly be categorized as:

a. **Critical** – An error could be detrimental to system functionality.
    a. If a system can continue to function relatively normally, even if the function is completely compromised, then it is a low critical risk.
    b. If a system fails to complete a primary requirement, then it is a high critical risk.
b. **Detectable** – The ease of detecting an issue arising with a particular function is inversely related to the risk.
    a. The lower the ease of detectability, the higher the risk.
    b. High chances of detectability correspond to lower risk.
c. **Probable** – The probability of an issue arising with a particular function is correlated to the risk.
    a. Low probability means there is little chance that the function will fail.
    b. High probability means there is a high chance that the function will fail.

Our organization has a risk assessment matrix that is used across statistical programming departments, but the implementation might look different depending on the needs of the department and of the specific program. Table 1 is a generalized risk assessment matrix:

| **Overall Risk** | *Likelihood* | | |
|---|---|---|---|
| *Impact* | **Low** | **Medium** | **High** |
| **Low** | L | L | M |
| **Medium** | L | M | H |
| **High** | M | H | H |

**Table 1. Risk assessment matrix.**

2. Requirements:

Requirements are statements that describe what a software system must do. They specify the functional, non-functional, and performance characteristics of the software system. Requirements are all important aspects of validation. Functional requirements ensure that the code performs the expected operations, while non-functional requirements ensure that the code is maintainable and portable. Performance requirements ensure that the code can manage large datasets and multiple users, while also running within a specified amount of time.

Risk assessments are established for each requirement, and a description of the potential effects of the risk are created. Table 2 is an example of risk assessments for different requirements. The risk for text file ingestion is medium; therefore, a mitigation plan is provided. No mitigation plan is needed for the text file output, which is considered low risk.

| Requirement Name | Requirement ID | Risk Assessment |
|---|---|---|
| Data Ingestion – Text Files | 1.1 | Medium Risk – Base R by default will guess column types. Mitigate by evaluating test cases with a variety of variable classes. |
| Data Output – Text Files | 1.2 | Low Risk |

**Table 2. Risk assessment.**

3. Test Cases:

Test cases are a set of tests that are designed to validate the requirements of the software system. They are used to validate the software system's behavior and ensure that it produces the expected results.

Requirements provide a clear understanding of what the software system must do, and test cases are used to validate that the software system meets these requirements. Together, requirements and test cases ensure the reliability, accuracy, and quality of the software system.

4. Test Code:

Test code is an important part of validation as it helps to ensure that the code meets the requirements and passes the test cases. Test code plays a crucial role in identifying any issues in the code and allowing developers to fix them before deployment. Figure 1 is a test code example.

```r
# Create a list of numbers
num_list <- c(1, 2, 3, 4, 5)

# Calculate the average using R code
avg <- mean(num_list)

# Check if the average is correct using an assertion
assert_that(avg == 3)
```

**Figure 1. Test code example.**

In this example, the requirement is to calculate the average of a list of numbers correctly. The test case is to create a list of numbers and check if the R code returns the correct average for that list. The test code uses an assertion to check if the calculated average is equal to the expected value of 3.

Figure 2 shows a high-level overview of the validation process:

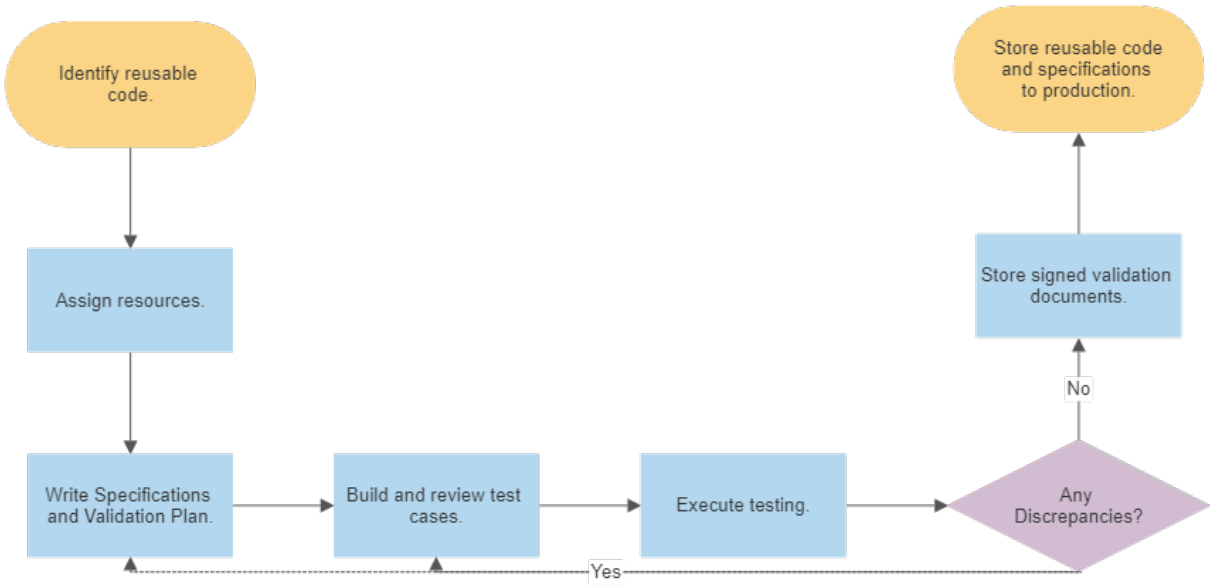


**Figure 2. Validation process.**

## VERSION CONTROL

Version control systems (VCS) are software systems that manage and track changes to a set of files over time. They allow teams to collaborate on software development projects by tracking changes to the code, maintaining a history of modifications, and managing different versions of the code. SVN (Subversion) and Git are two of the most popular version control systems in use today.

### SVN

Subversion (SVN) is a centralized version control system that was created to manage software development projects. In SVN, all changes to the codebase are stored on a central server, and

developers must check out a copy of the code from the server to make changes. When a developer has updated their local copy of the code, they can then commit those changes back to the central server.

One of the advantages of SVN is that it is easy to use, especially for smaller development teams. The central repository makes it easy for developers to collaborate, and it is also easy to revert to a previous version of the code if necessary. It also provides tools for resolving conflicts and for managing the repository.

### GIT

Git is a version control system that tracks any changes to shared files, including reports sent to stakeholders and both input and output data. It provides an effective tool to work collaboratively and ensure that any modifications made to code and data are traceable. Git can be installed and maintained on local machines, benefiting programmers who maintain repositories – which can be thought of as storage folders containing the files – on both local and secured internal servers. Git provides privacy and support to maintain files used for internal purposes.

Programmers also use GitHub, a cloud-based platform which hosts the Git software. GitHub's web interface allows users to visualize and interact with files and any changes through graphics. GitHub provides the required functionality to host R packages, whether created for data processing within an organization or analysis packages to produce results shared outside of the organization. The combined functionality of Git and GitHub can be leveraged to accommodate a spectrum of data privacy and confidentiality requirements.

## CODING PRACTICES

### R

#### *R Reproducibility – DataPackageR*

DataPackageR, an R package hosted by ROpenSci, an organization which aims to foster open and reproducible research through reusable software, most notably in the form of R packages, allows users to preprocess and tidy raw data into versioned and packaged analysis-ready data sets (Finak, et al., 2018). It is currently used to address reproducibility by one department at SCHARP which primarily employs R. The package streamlines the process of sharing all necessary support and development documentation and files with stakeholders. Data packages built by DataPackageR hold raw data, Rmarkdown preprocessing scripts, Quality Control reports, and analysis-ready datasets derived from the preprocessing scripts. Installing a DataPackageR created data package allows the user to view the preprocessing scripts and QC reports as vignettes, access the built data objects, and view data versioning of the built objects.

During the data package building process, the user can toggle which scripts to have active for package build and add additional scripts. This is useful since, in our paradigm, study data will typically arrive in a staggered timeframe and require constant data repackaging. Rebuilt data objects will have a new hash key – a unique identifier associated with a unique piece of data – to determine data versions. In contrast, static data objects will remain with the same hash key throughout the package rebuilds. Figure 3 shows the data package skeleton structure, Figure 4 shows a high-level process flow using DataPackageR, and Output 1 shows an example of the Data Digest file content after building the package using DataPackageR.

```
PackageName root
|--- data # Holds analysis-ready data objects
|
|--- data-raw # Holds preprocessing scripts, QC reports, preprocessing-dependent files
| |--- documentation.R # Roxygen documentation for data objects.
|
|--- datapackager.yml # Configuration file
|
|--- DESCRIPTION # Package description. Adds a DataVersion string for data set version.
|
|--- inst
| |--- extdata # Holds raw data files
| |--- doc # Holds processed vignettes. Accessible via vignette(PackageName).
|
|--- vignettes # Holds data-raw scripts that are processed into vignettes.
|
|--- R
|
|--- DATADIGEST # Holds md5 hash of each data object.
|
|--- man # Holds autogenerated documentation as rd files.
```

**Figure 3. Package skeleton structure.**

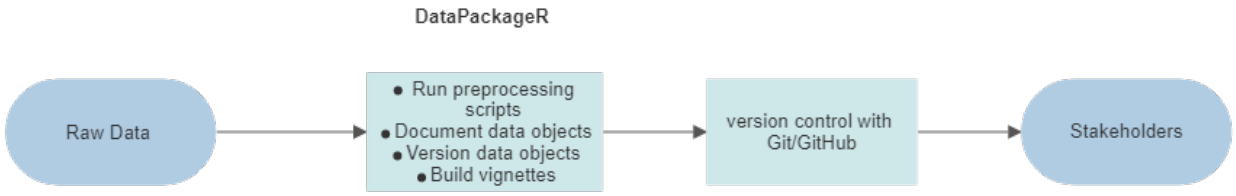


**Figure 4. Process flow with DataPackageR.**

```
DataVersion: 0.1.1
dataset_1: e1de5z44385d13e7f8129k1922w29c6d
```

**Output 1. Data Digest output file from DataPackageR.**

### *Company-Specific R Packages*

Frequently, study data are received in a standardized format. For our organization, this is often true of immuno-assays which are regularly processed by our R programming team. Data for common assay types are often transmitted similarly regardless of source. Previously, our programming teams would script and verify novel code for each assay delivery which led to the need for redundant verification activities and could have been more efficient. We noted these inefficiencies and identified a need to formalize commonly used functions into company-specific R packages.

In addition to decreasing redundancy, creating reusable company-specific R packages could ensure that all incoming data, regardless of the study or source, is processed the same way every time. This, in turn, decreases turnaround time, as functions and processes would not start from scratch build each time. R programmers across teams throughout our organization now share validated R packages for assay processing as well as a supplemental utility package. These packages allow R Statistical Programmers to

streamline assay data processing. R programmers also have department-specific R packages that address their partners' specific needs.

### *Data Specifications*

Data specifications are used to define the structure, format, and content of data. Specifications serve as a communication tool between functional teams which may include the data management team, statisticians, and programmers as well as others. Data specifications ensure that everyone is working from the same data definitions and formats, which is critical to producing accurate and reliable results.

Data specifications are also important for supporting good team relations as they provide a clear and consistent understanding of the data requirements for the study. This allows all teams to work together more effectively and efficiently with fewer misunderstandings, which is particularly important in large, complex clinical trials.

In addition, data specifications are used for data manipulation, derivation, and analysis. They provide a blueprint for the creation of the database and the data management plan, which includes the data cleaning, coding, and analysis procedures. Data specifications also provide guidelines for the creation of analysis datasets, tables, figures, and listings, which are used for statistical analysis and reporting.

### *Template Processing Scripts*

Template processing scripts provide a standardized and efficient approach to data manipulation and analysis. These scripts are pre-written code that can be easily adapted to different datasets and analysis requirements, which saves programming time and reduces the risk of errors. We use a repository to maintain the template processing scripts.

In addition to standardizing code/script across multiple studies, template processing scripts can also help ensure reproducibility of results. By using a standardized script, others can easily reproduce the analysis and results. This is particularly important in clinical research where reproducibility and traceability are essential.

## SAS

### *Legacy Code and Company-Specific Macros*

As with most of the industry, SAS has been the preferred programming language for manipulating and analyzing clinical trials data within our organization, consequently, much of the legacy code is in SAS and we are still expected to maintain robust SAS-programming capabilities. As the demand for more studies, quicker turnover and more robust programming increased, we began to explore options for greater efficiency. Company-specific macros were developed to reduce the coding effort for functions that would be used across multiple groups and a code repository was created. Examples include but are not limited to SAS macros for qualitative statistics (to calculate frequencies, mean, median, percentiles, interquartile range, etc.), proc report templates, proc compare reports.

### *Data Specifications*

In our organization data specifications are created and maintained through collaboration between the programmer and the statistician. The nuance of this collaboration differs slightly between groups depending on team needs and requirements. In some groups, the statistician creates and maintains the data specifications with feedback and updates from the programmer, while in other groups, the responsibilities are reversed, and the programmer creates and maintains the specifications with feedback provided by the statistician.

The data specification details, and the dataset structures are determined in part by the required content of the TLF mock shells and, depending on the purpose of the analyses, conform to CDISC ADaM standards. Most analysis datasets follow the BDS structure.

## VERIFICATION

Verification, like validation, is determined by a risk assessment matrix. However, the verification process details differ by User groups and whether they are using SAS, R, or both. We will describe the risk assessment matrix and the different verification forms.

Table 3 summarizes the user groups, and the type of verifications usually conducted.

| | **User Groups** | | |
|---|---|---|---|
| | **Predominantly SAS** | **SAS and R** | **Predominantly R** |
| **Code** | Legacy code in SAS | Legacy code in SAS, now moving to R | Legacy code in R |
| **Validated environment** | SAS version 9.4 | SAS version 9.4<br>Base R version 4.0.4 | Base R version 4.0.4 |
| **Version Control** | SVN | | git |
| **Verification level determined by a risk-based matrix** | Double programming<br>Targeted checks<br>Code review | Double programming/ code review | Double programming/ code review |

**Table 3. Summary of verification by user groups.**

### Risk Assessment Matrix

All high-risk datasets are double programmed. The verification level of the TLF programs is determined by the projected impact erroneous data could have on the endpoint analyses as well as the perceived likelihood for the occurrence of that error. Table 4 is a verification risk assessment matrix.

| **OVERALL RISK** | | **LIKELIHOOD (of datapoint correctness)** | | |
|---|---|---|---|---|
| | | **Low** | **Medium** | **High** |
| **IMPACT (Study and Endpoint Risk)** | **Low** | Data review | Data review | Code review + Data review |
| | **Medium** | Data review | Code review + Data review | Independent verification + Data review |
| | **High** | Code review + Data review | Independent verification + Data review | Independent verification + Data review |

**Table 4. Verification risk assessment matrix.**

**Impact on study and endpoint:**

a. **Low risk**: Correcting errors requires light documentation and has limited impact on SCHARP and our collaborators.

Examples: Exploratory endpoints for trials of any phase. Pre-clinical studies. Observational studies.

b. **Medium risk**: Correcting errors requires cross-center effort (e.g., SCHARP and Lab Center/Leadership Operations Center) but errors, if corrected promptly, are unlikely to have long-term consequences.

Examples: Primary and secondary endpoints for phase 1 trials.

c. **High risk**: Correcting errors requires cross-center effort. Consequences of errors may be long-term and/or impact the reputation and standing of SCHARP and our network partners.

Examples: Primary and secondary endpoints for high-profile studies such as efficacy trials or trials in sensitive populations (e.g., infants). Projects where processed datasets are delivered to non-network collaborators or shared with the public (e.g., with a manuscript). Datasets which will inform changes to the design of ongoing trials.

**Impact on programming task:**

a. **Low risk**: Minimal or simple programming, so errors are unlikely.

Examples: Refreshing datasets which were previously processed, with no changes to the requirements. Datasets where the lab mostly or completely derives the analysis variables, such as pass-through datasets or animal pharmacokinetics studies where time and drug levels variables are provided by the lab.

b. **Medium risk**: Programming complexity increases, but data structures are well-known, well-documented, or simple, and processing errors are due to misuse of code rather than unforeseen aspects of data.

Examples: Initial processing using validated code libraries or template scripts. Revised processing of datasets which were previously processed differently.

c. **High risk**: Programming difficulty increases, and data structures are unfamiliar or complex. Errors are due to unexpected or unknown aspects of data, or the need for new or complex code.

Examples: New processing code for new or established assays. Any new complex dataset requiring custom programming (e.g., new pharmacokinetics analysis datasets which combine drug timing information from multiple CRFs). Initial processing using custom code.

## Double Programming / Independent Verification

Double programming is the most rigorous method of verification amongst R and SAS Statistical Programmers. This strategy requires both a primary programmer and a verification programmer complete a programming task independently from a common specification and compare results. Both programmers use the same input files and program the code outlined in the Data Specifications but differ in the use of functions. For example, if the primary programmer uses company-defined SAS macros, then the verifier codes in Base SAS or SAS SQL. Within our R environment, the primary programmer may use a verified template data processing code, if available, and may also code using R's Tidyverse syntax.

In contrast, the secondary programmer may use base R scripts to verify the primary programmer's code. This ensures a standardized verification approach, as the two unique code syntaxes should lead to the same results. If questions or concerns arise, the programmers can use the same sources for help, such as the study's respective statisticians or documentation. Once programming finishes from both ends, the outputs are compared for discrepancies. Any discrepancies found are resolved prior to finalizing the processed dataset.

## Targeted Checks

Targeted checking is employed when it is determined that only a few high-risk outputs, parameters, variables derived from raw data, or a low-risk programming and medium risk impact of data issues, are verified. Targeted checks are similar to double programming / independent verification, differing only in the volume of data being checked, i.e., key data points alone are verified. This method is often used for tasks that are relatively simple, but still require a high degree of accuracy.

## Code and Data Review

In some lower-risk programming situations, code and data review may be more appropriate than full independent programming. With this type of verification, two programmers can work on different studies simultaneously and then alternate the review process. Programmers conduct code reviews followed by data reviews to ensure that the code used to process study data performs as expected and that the processed data used for analysis meets quality checks. Below is a checklist a programmer follows while performing a review:

Code Review:

- Code runs
- Code follows programming principles
- Output matches study's data package objects
- Data package installs without errors
- Processing code matches processing steps in Data Specifications
- Figures, tables, and captions align with study expectations in reports
- Log files are clean (without error or warning messages or unnecessary notes)
- Functions consistently yield expected results (joins, sum, filter)

Data Review:

- Data is complete (e.g., all expected values are in data)
- Data matches what is expected from the Data Specifications and Data Transfer Plans

If there are any discrepancies, the reviewing programmer will follow up with the primary programmer to clarify questions or comments. Once resolved, the data and code review are considered complete.

## SAS AND R ANALOGOUS WORKFLOW

This section focuses on the workflow process of SAS and R programming languages. While these two programming languages have differences in syntax and implementation, their workflow processes are similar. One key difference to highlight concerning data processing is the type of data process; when processing non-assay data, such as clinical study data, SAS is the predominant language, and the CDISC data submission standards are followed. This contrasts with assay data processing, which only processes raw data to derived data. In this section, we will discuss the similarities and differences between the SAS and R programming workflows, and how they are used in statistical programming practices within the organization. We hope that this will enable readers to gain insights into how they can improve their statistical programming practices and optimize their use of SAS and R programming languages. Figure 5 is a flow diagram showing a high-level overview of the process when working with assay data:

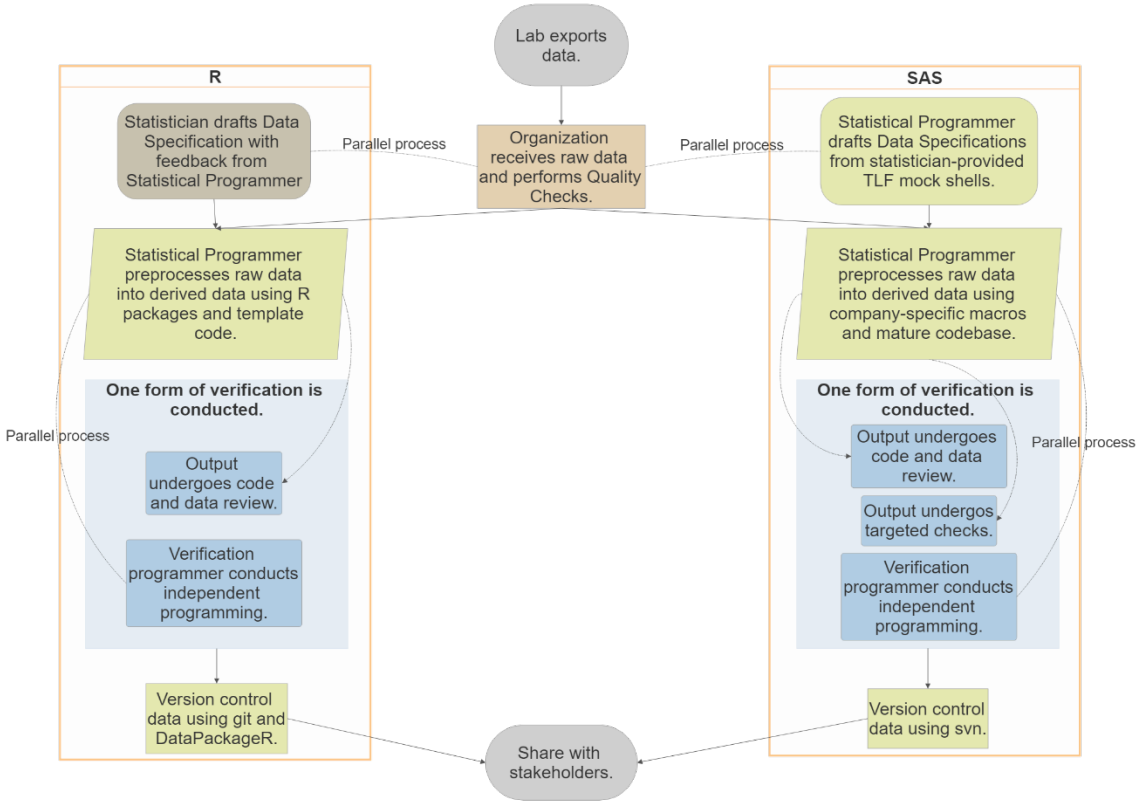


**Figure 5. Programming process.**

## FUTURE GOALS

We continue to look toward the future and focus on further improvements to our statistical programming practices. By implementing these goals, our organization can continuously enhance its programming practices and remain current with the latest developments in the field. These enhancements include expanding R packages and template code, adopting DataPackageR in other departments, increasing training amongst SAS and R programmers, continuing SAS and R validation, building and adopting more tools for use regardless of language, and formalizing our verification procedures across languages.

### R USER IMPLEMENTATIONS

#### Expand R Packages

R Statistical Programmers are currently expanding our library of R packages used for incoming assay data. We have validated packages that are tailored to our historically most frequent incoming assay data. However, as we receive new data from other types of assays, we aim to expand our validated package suite to support these assays. Priority of package creation will depend on incoming data frequency and current available documentation.

#### Expand Template Code

There is an ongoing need to further expand our template code. Our organization maintains a substantial amount of SAS macros and R template scripts. Existing processes often depend on template code; therefore, continued maintenance of those packages is imperative. However, handling code-base can be difficult, as it may be poorly documented, and lack sufficient modern validation. Additionally, the code might have dependencies, especially with package versions; when these packages are updated and incompatible with the code, the code is broken and requires the user to update with each application. Creating new template code will help streamline and align processes and ensure better confidence in code performance. Template code also provides higher visibility since multiple people will be using it; this, in turn, will allow for more frequent refactoring to stay up to date with current procedures.

### Expand DataPackageR Usage

Currently, DataPackageR is used by a limited subset of our programming teams. However, due to the ability to track data object versioning through the package using the hash key, we would like to expand DataPackageR to other departments. This would also allow other departments and stakeholders to keep track of data object versions, increasing reproducibility and verification efforts.

### Training

Although R Statistical Programmers perform daily tasks similarly across departments, specific procedures might vary by department. One goal to allow continuous learning and growth in the organization is to allow R Statistical Programmers to rotate through all departments. This is useful as R Statistical Programmers who usually work with, for example, preclinical data could learn the practices and procedures used for higher phase clinical trials. Programmers who usually go through double programming could learn to conduct code and data reviews and vice versa. This will allow more flexibility and provide more redundancy among skillsets which will contribute to better team resilience.

## CONTINUING R VALIDATION

An ongoing effort exists to expand and validate the R ecosystem within our organization. Base R version 4.4 validation was completed at the end of 2022, and efforts are ongoing to validate more current versions of R. Additionally, validation of other R packages consistently used in the organization, such as Tidyverse, is highly prioritized. However, the extent of validation still needs to be determined as other external resources have already produced robust documentation for these packages. As we continue to train SAS programmers to use R and further expand the use of R across our organization, ensuring that all packages are validated, and that our approach for validation aligns with requirements for regulatory submission are of high priority. Another high priority is validating packages used by individual departments giving users more confidence in the output.

## BUILDING MORE LANGUAGE-AGNOSTIC TOOLS

Building language agnostic tools that allow data and code to be easily transferred between SAS and R, can help to reduce the time spent on data conversions and data preparation, increase productivity and collaboration, and make data analysis more accessible to a wider range of users. Here are examples of language agnostics tools:

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Jupyter supports multiple programming languages including R and SAS.

Git is a version control system that is used to manage and track changes to code. Git supports multiple programming languages including R and SAS.

Quarto is an advanced version of R Markdown with syntax alignment towards Pandoc that supports multiple languages such as R, Python, Julia, and more through Jupyter notebooks. Using Jupyter kernels when creating documents with Quarto allows the user to choose their programming language while still being able to output the same document. This Quarto is extremely useful for SAS and R programmers who wish to align on report formatting and increases reproducibility amongst the languages.

## FORMALIZING VERIFICATION ACROSS LANGUAGES

Programming teams in our organization follow slightly different verification process based on the complexity of data and output and intended purpose of the output. We are working to align the verification process across languages by establishing standards for verifying data that are applicable to both SAS and R. This includes a clear definition of what data should be verified, how it should be verified, and what the acceptable levels of accuracy and consistency are.

Automated verification tools can be used to check the accuracy of data produced in SAS and R. For example, automated tests can be run to ensure that data meets specified accuracy and consistency standards.

Cross-checking results between SAS and R can help to ensure that the data produced in both languages is consistent and accurate. For example, if a data analysis is performed in SAS, the same analysis can be performed in R, and the results can be compared to ensure that they are the same.

Documenting the verification processes used for data produced in SAS and R can help to ensure that the verification process is consistent and repeatable.

Regularly reviewing the verification processes used for data produced in SAS and R can help to identify and resolve any problems or inconsistencies.

## CONCLUSION

This paper describes considerations and strategies for teams working to bring R into a traditionally SAS-predominant environment. The synergy of R and SAS in statistical programming can provide a powerful tool for data analysis if the benefits and limitations of each are acknowledged and appropriately addressed in the development of standard procedures and a strategy for application. While SAS provides a reliable platform for data submissions, R offers a wide range of open-source techniques that allow for easier collaboration. Together, these tools allow for more efficient support of clinical trials and other exploratory studies. Further, as the industry becomes more accepting of the use of open-source languages for submissions, it is imperative that programming practices are considered and developed to meet all requirements.

Additionally, this paper outlines the statistical programming practices within an organization that employs both SAS and R and how teams can differ and align within that organization depending on the language of choice. We discussed the roles and responsibilities of statistical programmers, workflow processes of SAS and R programming languages, and best practices for efficient and maintainable coding.

Our analysis highlighted the importance of software validation, version control, and verification of datasets and reports to ensure the quality and compliance of the organization's outputs. Moreover, we identified several areas for future work, such as expanding R Packages and template codes, cross-training SAS and R programmers, continuously conducting software validation and following coding best practices, building more language-agnostic tools, and formalizing processes across programming languages. Implementing these enhancements will allow our organization to grow and improve practices continuously, thus further improving the quality of our deliverables to our stakeholders.

Overall, this paper provides insights into statistical programming practices that can benefit any organization that relies on statistical programming for data analysis and decision-making. By leveraging the strengths of both SAS and R and adopting best practices for software validation and version control, statistical programmers can support more effective and efficient data analysis, leading to better decision-making and improved outcomes.

## REFERENCES


FDA. (2002, January). *General Principles of Software Validation: Guidance for Industry and FDA Staff*. Retrieved from https://www.fda.gov/regulatory-information/search-fda-guidance-documents/general-principles-software-validation

FDA. (2003, September). *Part 11, Electronic Records; Electronic Signatures - Scope and Application: Guidance for Industry*. Retrieved from https://www.fda.gov/regulatory-information/search-fda-guidance-documents/part-11-electronic-records-electronic-signatures-scope-and-application

FDA. (2015, May). *Statistical Software Clarifying Statement*. Retrieved from chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.fda.gov/media/161196/download

Finak, G., Mayer, B., Fulp, W., Obrecht, P., Sato, A., Chung, E., . . . Gottardo, R. (2018). DataPackageR: Reproducible data preprocessing, standardization and sharing using R/Bioconductor for collaborative data analysis. *Gates Open Research*, 16.

Hughes, E. (2021, 01 19). *R Package Validation Framework*. Retrieved from Posit: https://posit.co/resources/videos/r-package-validation-framework/

Hughes, E., Miller, E., Vendettuoli, M., Eshghi, P., & Gans, M. (2021). *valtools: Automate Validated Package Creation*. Retrieved from https://github.com/phuse-org/valtools

Stutzman, P. (2016). Handling Interim and Incomplete Data in a Clinical Trials Setting. *PharmaSUG 2016 Conference*, (p. 13). Seattle, WA. Retrieved from PharmaSUG 2016 Conference: https://www.pharmasug.org/proceedings/2016/IB/PharmaSUG-2016-IB12.pdf

Vendettuoli, M., Zhang, E., & Zou, R. (2023). Strategies for Code Validation at Statistical Center for HIV/AIDS Research and Prevention (SCHARP) (accepted). *PharmaSUG 2023 Conference*, (p. 12). San Francisco.


## ACKNOWLEDGMENTS


We would like to thank Amber Randall and Paul Stutzman for their guidance and review.


## RECOMMENDED READING

- *R Package Digest reference manual: https://cran.r-project.org/web/packages/digest/digest.pdf*
- *R Package Valtools GitHub repository: https://github.com/phuse-org/valtools*
- *R Package DataPackageR website: https://docs.ropensci.org/DataPackageR/index.html*
- *Quatro within RStudio guide: https://quarto.org/docs/tools/rstudio.html*
- *Jupyter notebook guide: https://docs.jupyter.org/en/latest/start/index.html*
- *Introduction to Git & GitHub https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners*

## CONTACT INFORMATION

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

Valeria Duran
Statistical Center for HIV/AIDS Research & Prevention (SCHARP) at Fred Hutchinson Cancer Center
vduran@scharp.org

Radhika Etikala
Statistical Center for HIV/AIDS Research & Prevention (SCHARP) at Fred Hutchinson Cancer Center
retikala@scharp.org

Haimavati Rammohan
Statistical Center for HIV/AIDS Research & Prevention (SCHARP) at Fred Hutchinson Cancer Center
hrammoha@scharp.org