PharmaSUG 2026: ET-104

Unleash the R-volution: A Blueprint for Building Package Validation Capabilities in our own organization

Kevin Lee, ClinVia, Philadelphia, PA

ABSTRACT

As the use of R continues to expand in clinical trial programming, ensuring that R packages are validated for regulatory compliance, reproducibility, and traceability has become essential. This presentation provides a step-by-step blueprint for building robust R package validation capabilities within your organization.

We will begin by clarifying what R and R packages are such as base and recommended packages maintained by the R Core Team to community-contributed and in-house packages. The session then focuses on why validation matters, emphasizing principles such as accuracy, reproducibility, and data integrity to meet FDA and EMA expectations for reliable, auditable results.

A central part of the presentation introduces a **risk-based validation framework** that classifies packages according to their **purpose**, **maintenance quality**, **community usage**, and **testing rigor**. Tools like the **R Validation Hub's *riskmetric*** and **Sanofi's *risk.assessr*** packages will be demonstrated for risk scoring and assessment. We'll also cover unit testing practices using ***testthat*** and show how to generate comprehensive **validation reports** including purpose, environment, documentation, dependencies, and testing results.

Finally, the presentation outlines how to integrate validation into your computing environment through Installation (IQ), Operational (OQ), and Performance (PQ) Qualifications, ensuring traceable and reproducible results. Attendees will gain practical insights and best practices for establishing an internal, sustainable framework to confidently leverage R in regulated environments.

Introduction of R programming

R has become a central language for statistical programming, data manipulation, and visualization in clinical development. Its open-source ecosystem makes it attractive for rapid analysis and reporting, but that same ecosystem creates a validation challenge. Packages are continuously evolving, dependencies can shift, and different package types carry different levels of operational and regulatory risk.

In a submission-related works, the question is not simply whether a package works on a developer's laptop. The question is whether the organization can show, with evidence, that the package performs its intended function consistently, that its use is traceable, and that the surrounding environment is controlled. That expectation applies whether the package is a base R component, a recommended package shipped with R, a CRAN package, or an internal package created for a specific workflow.

What R Packages Are and Why They Matter

R packages are a bundle of functions, datasets, documentation, and compiled code that extends the base system. In practice, packages are the building blocks of modern clinical programming workflows, supporting data import, cleaning, analysis, table generation, and visualization.

As shown below, there are many types of packages such as base R, recommended packages maintained by the R Core Team, community-contributed packages such as CRAN packages, and internal packages developed within our own organization. The more a package influences analysis results, tables, or submission outputs, the more important it becomes to validate it before routine use.

Type	Description	Examples
Base packages	Core components of R, always included	base, stats, graphics, utils
Recommended packages	Officially supported, shipped with R installation	Matrix, lattice, survival
Contributed packages	Developed by the community; need to be downloaded from CRAN or GitHub	ggplot2, dplyr, caret, shiny

In-house packages	Developed within organizations for internal use	companytools, companyutils (custom)

Examples of packages commonly used in clinical programming include haven for SAS transport files, dplyr and tidyr for data manipulation, flextable for Word-ready tables, gtsummary and rtables for reporting, admiral for ADaM dataset creation, and ggplot2 for graphical output. These packages are powerful, but they also become part of the regulated computational chain and therefore merit formal evaluation.

Why R Packages Validation?

Validation is the process of showing that a system or component consistently produces results that meet predefined specifications. For R packages, the essential principles are accuracy, reproducibility, and traceability.

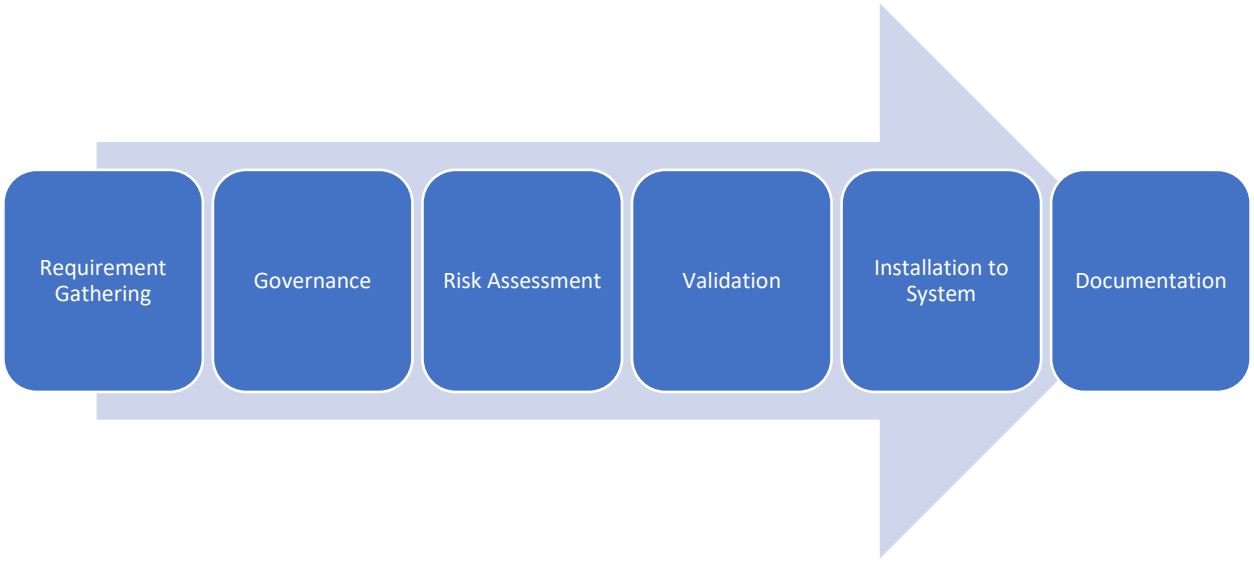
- Accuracy means the package performs the intended function correctly.
- Reproducibility means the same input and code produce the same result.
- Traceability means the organization can document what was used, what was tested, and what version was approved.

These principles matter because clinical trial programming contributes directly to regulatory submissions and decision making. An incorrect transformation, an untested edge case, or an undocumented dependency can lead to reporting errors, audit questions, or avoidable rework. Validation reduces these risks and helps demonstrate that the programming environment is controlled and fit for purpose.

Validation also supports expectations associated with electronic records and auditability. In practice, regulated teams often need to show what software was used, which version was installed, how it was qualified, and what evidence supports its approved use. A documented package validation process helps satisfy that need without forcing teams to treat every package as a one-off exception.

R Packages Validation Process

A structured and well-documented validation process is essential to ensure that R packages used in clinical trial programming meet regulatory expectations for accuracy, reproducibility, and traceability. The proposed R package validation process consists of six key stages: Requirement Gathering, Governance, Risk Assessment, Validation, Installation to System, and Documentation. Each stage plays a critical role in establishing a compliant and sustainable framework.



Requirement Gathering

The validation process begins with a formal collection of requirements to define the scope and purpose of the package within the organization. The following information could be documented for every package request:

- **Package Name and Version**: Explicitly identify the specific package and version number required for the environment.
- **Intended Use**: A clear description of why the package is needed and what business or scientific problems it solves.

- **Functional Requirements**: Detailed specifications of the features and capabilities that will be utilized by the team.

Governance

Effective governance ensures that new package requests are evaluated and authorized before committing validation resources.

- **Request Evaluation & Approval**: Each request is assessed to verify business needs, ensuring that no existing validated package already fulfills the requirement.
- **Source Integrity**: Packages must be sourced from trusted repositories, such as CRAN, Bioconductor, or an internal Posit Package Manager.
- **Operational Ownership**: The process defines clear roles, including a Process Owner from the business side and a Technical Steward from IT.
- **Inventory Management**: Approved requests are logged into a Master Package List to maintain a central source of truth and prevent duplicate validation efforts.
- **Sign-off**: A formal "Go/No-Go" decision is required by the Functional Lead or Department Head, and its implementation time.

Risk Assessment

A risk-based approach is essential for determining the level of validation effort required for a given package. The assessment framework evaluates packages based on four primary criteria:

- **Purpose**: Distinguishes between statistical, non-statistical, and data management tools.
- **Maintenance Good Practice**: Reviews the presence of vignettes, active maintenance, and bug-tracking systems.
- **Community Usage**: Analyzes package maturity and popularity (e.g., download counts) as a proxy for community testing.
- **Testing**: Examines the internal Software Development Life Cycle (SDLC) and existing unit testing within the package.

Risk is categorized as **Low**, **Medium**, or **High**. For example, a widely used, well-maintained CRAN package for non-statistical visualization is typically considered Low Risk. Conversely, internal or non-maintained statistical packages are classified as High Risk and require more extensive testing. Tools like the R Validation Hub's riskmetric or Sanofi's risk.assessr can be used to generate objective risk scores.

- R Validation Hub Validation Package
    - riskmetric: developed by R Validation hub, fostering community-driven to R package validation efforts.
    - Package Assessment
        pkg_ref(c("haven")) %>% pkg_assess()
    - Package Score
        pkg_ref(c("haven")) %>% pkg_assess() %>% pkg_score()
- Sanofi Validation Package
    - risk.assessr : developed by Sanofi R validation team
    - Package Assessment & Score
        risk_assess_package <- risk_assess_pkg( )

Validation

Validation involves confirming that the package consistently produces results that meet predefined specifications.

- **Validation Criteria**: Based on the gathered requirements, specific criteria are established to define what "success" looks like for the package.
- **Testing Scenarios**: Several testing methodologies may be employed:
    - **Unit Testing**: Using the testthat package to test individual functions.
    - **Verified Output Comparison**: Comparing package output against known, validated results.
    - **Double Programming**: Comparing results to an independently programmed analysis.
    - **Expectation Testing**: Verifying that the package correctly produces expected messages, warnings, or errors.
- **Results**: All validation codes and their results (Pass/Fail) must be recorded.

The typical R package unit testing could be done by testthat R package.

```
libraray(testthat)
library(haven)
```

```
test_that("Variable label stored as attributes", {{
df <- read_xpt("{dm_data}")
expect_equal(attr(df$STUDYID, "label"), "Study Identifier")  expect_equal(attr(df$SUBJID, "label"), "Subject Identifier for the Study")}})
```

Package Installation to System

Once validated, the package must be qualified for the specific production environment. This follows the standard GxP qualification model:

- **Installation Qualification (IQ)**: Verifies that the correct versions of R, RStudio, and the validated packages are installed properly.
- **Operational Qualification (OQ)**: Ensures that the R environment and its packages function as expected within the system.
- **Performance Qualification (PQ)**: Confirms that the packages execute correctly and produce accurate, consistent output under production-like conditions.

Validation Documentation

Thorough documentation is mandatory for regulatory compliance (e.g., 21 CFR Part 11) and auditable results.

- **R Package Validation Report**: This report compiles the assessment and testing results for the specific package functions. It could include the risk score and unit testing results.
- **System Qualification Report**: This document focuses on the R installation itself, ensuring the reproducibility of the entire environment

Lessons Learned

- We want to start with a clear inventory of packages used in production workflows and classify them by business impact. Do not wait until submission time to discover that an important dependency was never reviewed.
- We should use a standardized assessment template so that different reviewers apply the same criteria. This improves consistency and reduces subjectivity across projects.
- We should treat documentation as part of the deliverable. A good validation package should explain what was tested, why the package is considered acceptable, and what limitations remain.
- We should revalidate when versions change, dependencies change, or intended use expands. A package can be acceptable for one workflow and unsuitable for another.
- We should build a sustainable process by combining automation, testing frameworks, and governance. The most successful validation programs are not one-time events; they are part of the organization's normal software lifecycle.
- We always think in the terms of compliance and scientific integrity.

Conclusions

R packages are now foundational to clinical trial programming, but open-source flexibility must be paired with a disciplined validation strategy. A risk-based approach allows organizations to focus effort on where the impact is greatest while still maintaining documentation, traceability, and reproducibility. By defining intended use, assessing package risk, testing with tools such as testthat, and qualifying the surrounding R environment through IQ, OQ, and PQ, teams can create a sustainable framework for regulated work. The result is greater confidence in analysis outputs, stronger audit readiness, and a practical path for expanding R use in Biometrics.

REFERENCES

- R Core Team & The R Project in r-project.org
- R Validation Hub in pharmar.org
- Riskmetric Package in github.com/pharmaR/riskmetric
- Sanofi risk.assessr Package in github.com/pharmaR/risk.assessr
- testthat Package in testthat.r-lib.org
- CRAN in cran.r-project.org/package=testthat

CONTACT INFORMATION

Your comments and questions are valued and welcomed. Please contact the author at

Kevin Lee

Clinvia LLC
kevin@clinvia.com

TRADEMARKS