PharmaSUG 2025 - Paper AP-132

Exploration and Insights from Open-Source R Packages' Dependencies, Programming Paradigms and Validation Strategies in Clinical Programming

Enki Wang, and Longfei Li, Sanofi

ABSTRACT

The R language has gained significant growth in popularity and adoption within clinical programming, largely driven by an expanding ecosystem of open-source R packages and a vibrant community.

This paper first explores the landscape of R package dependencies within curated pharmaceutical ecosystems such as pharmaverse, pharmaR, insightsengineering and openpharma. It summarizes available and appropriate dependencies and identifies the most used functions and packages categorized by their specific functionalities to facilitate a smoother transition to R for clinical programmers. Furthermore, the paper examines various programming paradigms, including comparisons between base R and Tidyverse R, standard evaluation versus non-standard evaluation, and different object-oriented programming approaches. It aims to guide informed decision-making regarding trade-offs in readability, maintainability and flexibility. Importantly, the paper highlights the validation strategies and tools necessary for developing robust, high-quality, and reliable packages. Finally, the paper demonstrates how interactive reports can be generated using Quarto dashboards and automatically deployed to GitHub Pages via GitHub Actions, ensuring regular updates.

The insights derived from this analysis will be valuable to developers and organizations seeking to embrace R. It provides practical suggestions for dependencies selection, coding paradigms, and validation approaches that align with the pharmaceutical industry's ecosystem and regulatory compliance.

INTRODUCTION

The pharmaceutical industry has experienced a notable shift toward adopting R for clinical programming. As a multi-paradigm programming language, R supports a variety of programming styles and offers a vast ecosystem of open-source packages. While this has promoted the application of R in clinical programming, it also presents challenges for clinical programmers.

From the perspective of programmers, this paper explores open-source R packages in pharmaceutical clinical programming, analyzing R package dependencies, programming paradigms, and validation approaches. It aims to provide practical suggestions and guidance to help clinical programmers adopt R and the open-source ecosystem more effectively and efficiently.

LANDSCAPE OF R PACKAGE’S DEPENDENCIES

OVERVIEW OF R PACKAGES

The packages that will be explored are from the end-to-end packages from the pharmaverse. The total number of 48 R GitHub Repositories is examined.

The entire analysis is processed by the following steps:

- Download packages from GitHub
- Explore the packages dependencies and functions
- Store the data for post-processing
- Summary data and Visualize results
- Generate the Quarto report

Table 1 lists example packages from pharmaverse.

repository	task	details
atorus-research/metacore	metadata	Provides an interface to read in various metadata sources and store in a standardized object
pharmaverse/metatools	metadata	Enables the use of metacore objects to build, enhance or check datasets using metadata - including removing/adding supplemental qualifiers from/to the parent SDTM domain
pharmaverse/aNCA	sdtm	(Pre-)clinical Non-Compartment Analysis (NCA) in a dynamic Shiny app
pharmaverse/datacutr	sdtm	Applying a datacut to SDTM
pharmaverse/sdtm.oak	sdtm	An Electronic Data Capture system and Data Standard agnostic solution that enables to develop SDTM datasets in R
pharmaverse/sdtmchecks	sdtm	Contains data check functions to identify SDTM issues that are generalizable, actionable, and meaningful for analysis
pharmaverse/admiral	ADaM	(ADaM In R Asset Library) - Modular framework to generate ADaM via R functions relying on community contributions
pharmaverse/admiralmetabolic	ADaM	Metabolism
pharmaverse/admiralonco	ADaM	Oncology
pharmaverse/admiralophtha	ADaM	Ophthalmology
pharmaverse/admiralpeds	ADaM	Pediatrics
pharmaverse/admiralvaccine	ADaM	Vaccines RELEASE EXPECTED ~2023

Table 1 Pharmaverse Packages

These packages are classified by task: metadata, sdtm, ADaM, tlg, esub, validation, synthetic, and utilities.

As Table 2 illustrates, the most used R packages are tailored for creation of table, listing, graph.

Task	Number of Packages
metadata	2
sdtm	4
ADaM	6
tlg	**19**
esub	2
validation	6
synthetic	3
utilities	6

Table 2 Number of packages by task

PACKAGE DEPENDENCIES

R package dependencies refer to other packages or external tools whose functionalities are used within a given package, and they are specified in the DESCRIPTION file under the Depends, Imports, or Suggests fields.

According to the R validation Hub's classification, the dependencies are divided into two categories:

- Statistical
- Non statistical

We will focus on the non-statistical packages that are classified into the following categories inspired by R validation Hub:

- Develop utility (document, test)
- Data manipulation (dplyr, tidyr, including string and date handling)
- Data Input / output (readxl, officer)
- Communication (knitr, rmarkdown, quarto)
- Modelling (non-statistical)
- Application Interface (Shiny)

The package dependencies are obtained by function `desc::desc_get_deps()`, which extracts the direct dependencies in the DESCRIPTION file. To detect function calls within R packages, it parses every scripts in the `R/` directory, using the function `getParseData(parse(file = file, keep.source = TRUE))` to collect all functions. The namespace of each function is then inferred by evaluating the code `environmentName(environment(fun))` after attaching the corresponding packages to the search path.

The number of dependencies by category among a total of 185 dependencies:

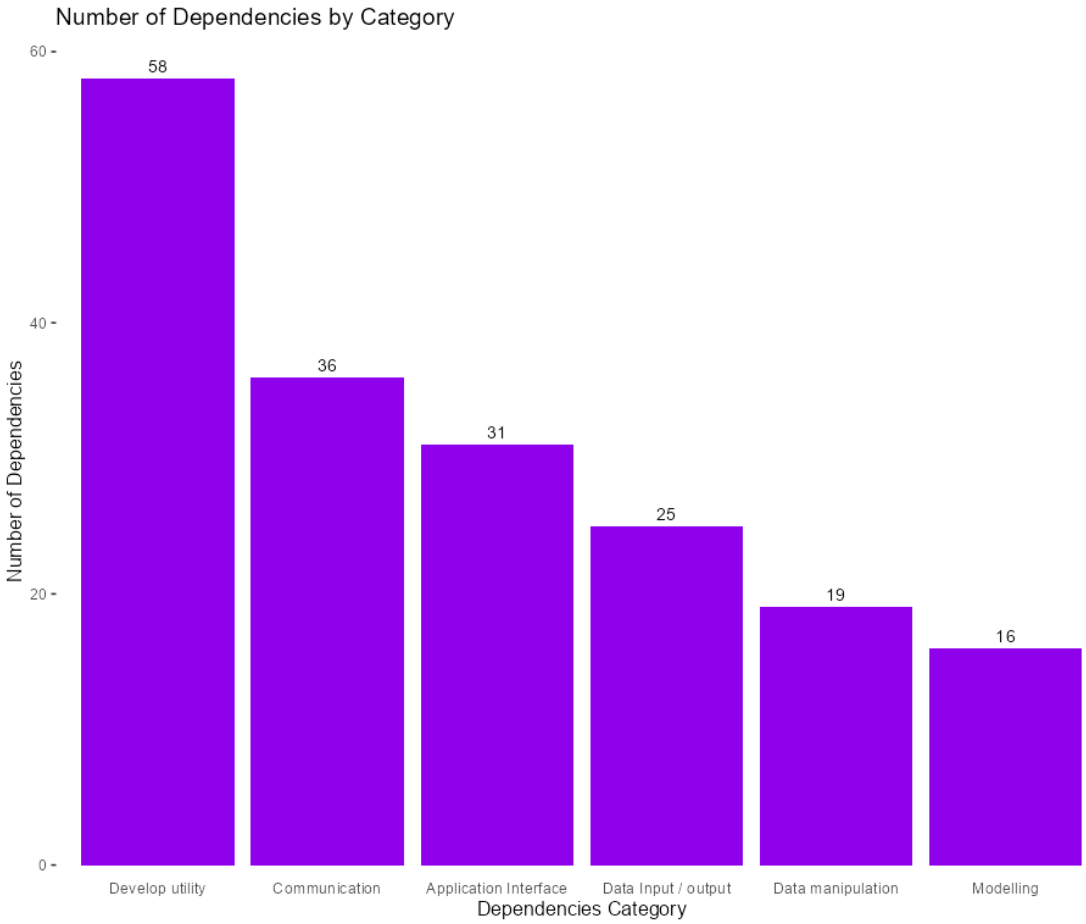

Figure 1 Number of Packages by Category

Top 10 direct dependencies by category among a total of 185 dependencies:

Category	Dependency	Frequency
Develop utility	testthat	43
	rlang	31
	purrr	28
	magrittr	25
	withr	25
	covr	18
	cli	17
	methods	17
	lifecycle	16
	tidyselect	16
Communication	knitr	39
	rmarkdown	37
	DT	12
	ggplot2	10
	kableExtra	8
	diffdf	6
	gt	6
	flextable	3
	grid	3
	huxtable	3
Application Interface	htmltools	7
	shiny	6
	miniUI	4
	shinyWidgets	4
	shinyjs	4
	rvest	3
	bslib	2
	golem	2
	sass	2
	shinyAce	2
Data Input / output	haven	9
	jsonlite	8
	readr	8
	xml2	7
	yaml	7
	readxl	6
	svglite	3
	DBI	2
	RSQLite	2
	arrow	2
Data manipulation	dplyr	37
	stringr	25
	tibble	23
	tidyr	22
	glue	14

	lubridate	14
	forcats	7
	stringi	6
	hms	4
	pillar	2
Modelling	stats	7
	survival	7
	broom	4
	car	3
	emmeans	2
	lme4	2
	MASS	1
	PKNCA	1
	TMB	1
	aod	1

Table 3 Top 10 Dependencies Per category

PROGRAMMING PARADIGMS IN R

R offers different programming paradigms, functional programming, object-oriented programming, and metaprogramming.

BASE R AND TIDYVERSE R

Base R provides the fundamental functionality and offers core functions for basic data manipulation, statistical analysis, and plotting. However, its historical evolution has led to inconsistent naming conventions, which may make it less intuitive for new users.

Tidyverse(`tidyverse::tidyverse_packages()`) is a cohesive collection of R packages designed to work together with a common API, which prioritizes ease of learning and convenience of usage.

The choice between Base R and Tidyverse is not merely a stylistic preference but a strategic decision with profound implications for team efficiency, onboarding, and long-term maintainability.

In this paper, we use R to iterate through the above open-source R packages and save the data in the CSV format for subsequent use.

The Example R package dependencies and function information:

admiral package dependencies

package	type	depends	version
admiral	Imports	admiraldev	>= 1.3.1
admiral	Imports	cli	>= 3.6.2
admiral	Imports	dplyr	>= 1.0.5
admiral	Imports	hms	>= 0.5.3
admiral	Imports	lifecycle	>= 0.1.0
admiral	Imports	lubridate	>= 1.7.4
admiral	Imports	magrittr	>= 1.5

admiral package function

package	pkg	fun	n
admiral	admiraldev	assert_data_frame	62
admiral	base	c	61
admiral	magrittr	%>%	59
admiral	rlang	enexpr	57
admiral	rlang	exprs	48
admiral	dplyr	mutate	46

Table 4 Example of R package information

The following visualization information is plotted using ggplot2 after the summary analysis.

Most Frequently Used Basic R Packages:

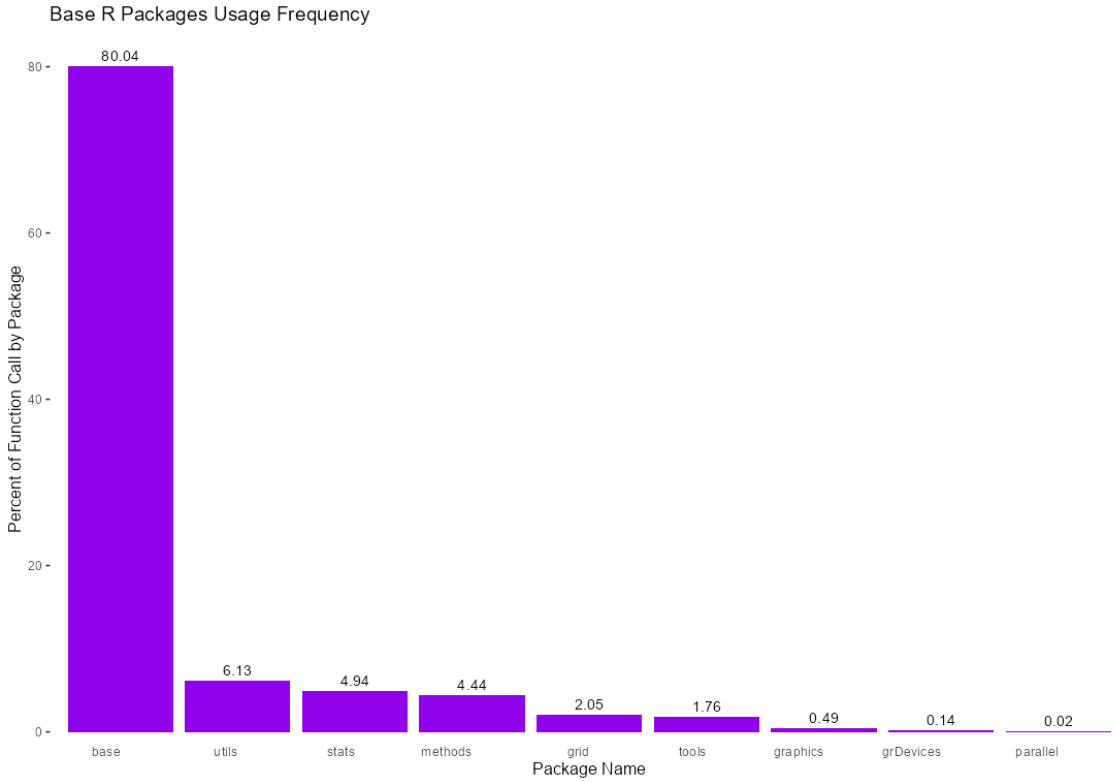

Figure 2 Base R Package Usage Frequency

Most Frequently Used Tidyverse Packages:

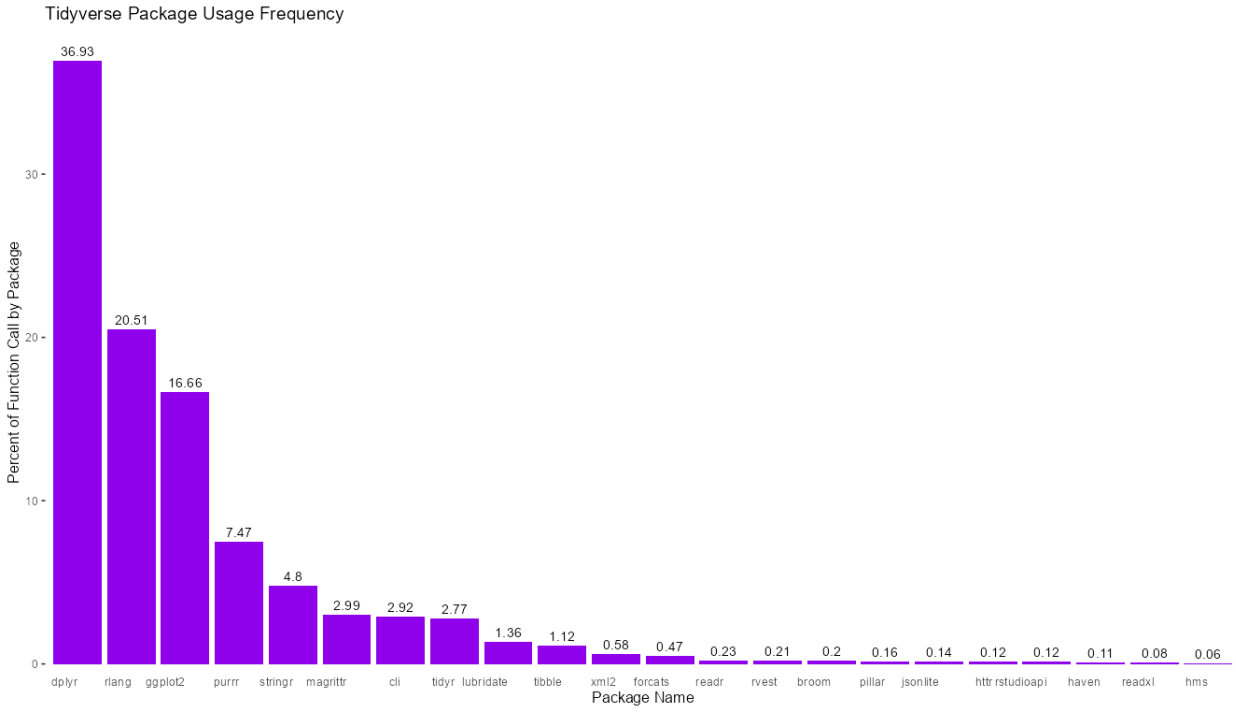

Figure 3 Tidyverse Package Usage Frequency

Most Frequently Used Basic R Functions among 1334 Total Functions:

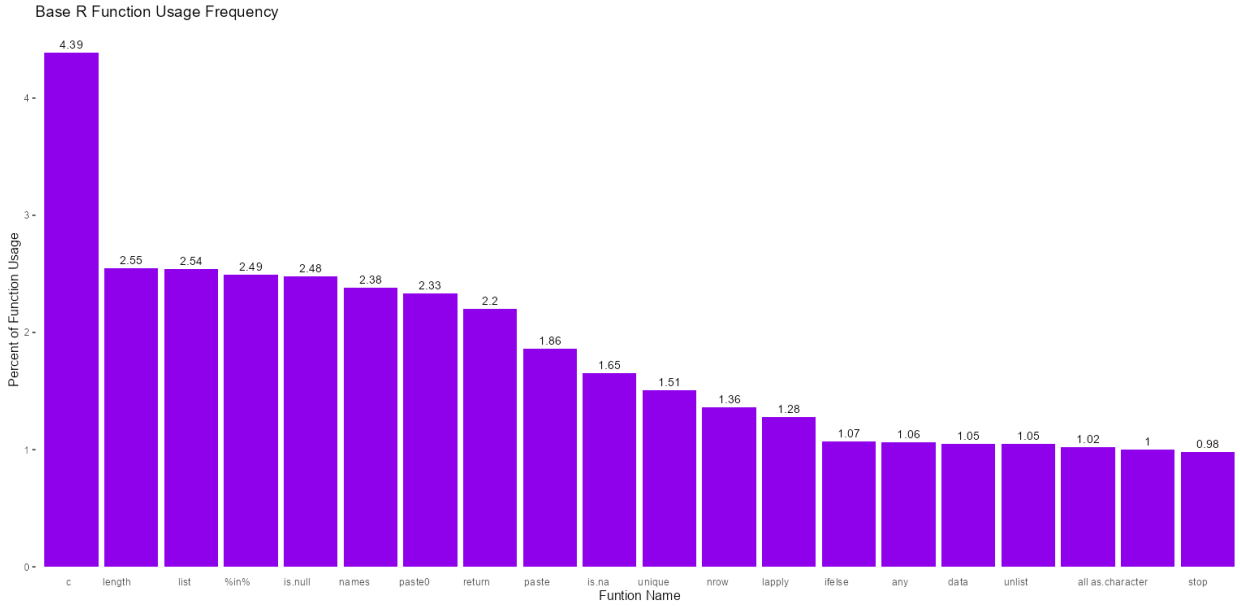

Figure 4 Base R Function Usage Frequency

Most Frequently Used Tidyverse R Functions aamong 1757 Total Functions:

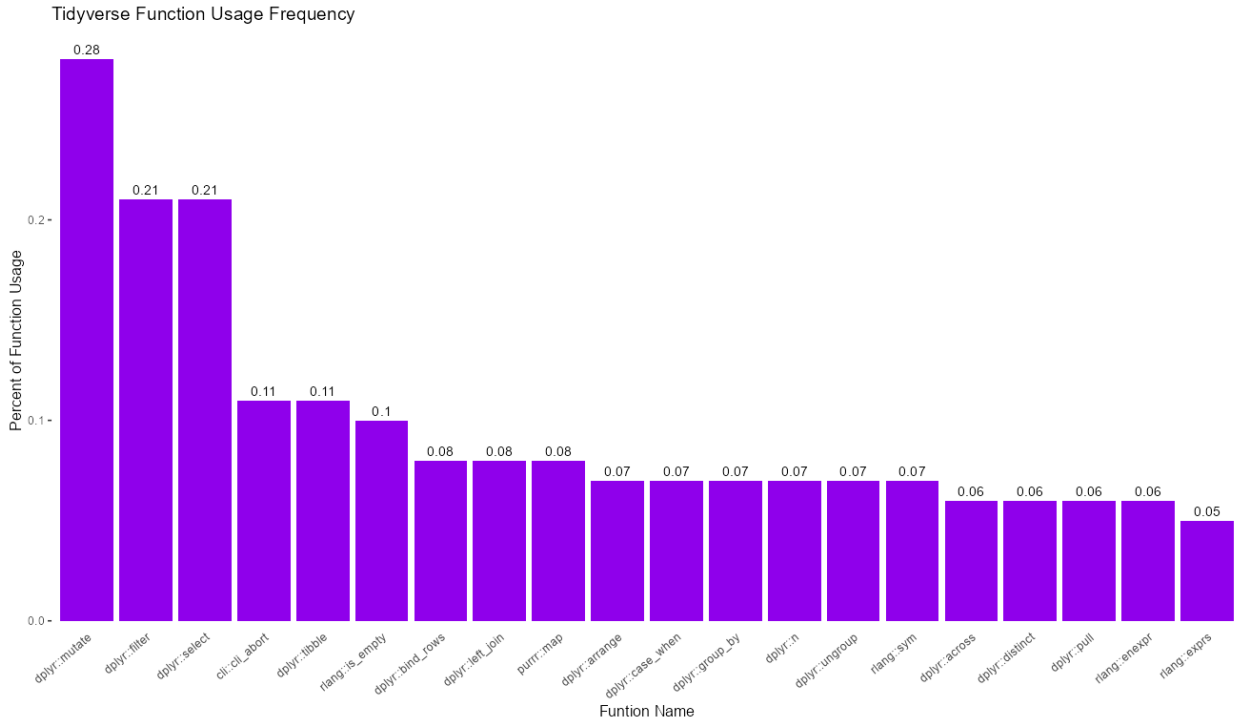

Figure 5 Tidyverse Function Usage Frequency

STANDARD EVALUATION AND NON-STANDARD EVALUATION

The main difference between the two programming approaches is how to pass the variables names in function parameters.

The below example, derived from pharmaverse R packages, illustrates the distinction between two programming paradigms from the perspective of function usage.

sdtm.oak	admiral
derive_seq(tgt_dat = apsc, tgt_var = **"APSEQ"**, rec_vars = **c("STUDYID","RSUBJID","SCTESTCD")**, sbj_vars = **c("STUDYID", "RSUBJID")**)	derive_var_obs_number(dataset = vs, by_vars = exprs(**USUBJID**, **VSTESTCD**), order = exprs(**VISITNUM**, desc(**VSTPTNUM**)), new_var = **ASEQ**)

Table 5 Dataset creation for sequence number

rtables	gtsummary

lyt <- basic_table() %>%
split_cols_by(**"arm"**) %>%
split_cols_by(**"gender"**) %>%
split_rows_by(**"country"**) %>%
analyze(**"age"**, afun = mean)
 	
tbl <- build_table(lyt, df)
	table <- tbl_summary(
	trial,
	include = **c(age, grade, response)**,
	by = **trt**, # split table by group
	missing = "no")

Table 6 TLG creation

The variable names in Non-Standard Evaluation can be used directly without quotation marks. This approach, which follows the tidyverse style, is more user-friendly, but it increases development and maintenance costs for programmers. After consideration of the trade-offs between user convenience and development complexity, developers should select a consistent programming paradigm aligned with the team's requirements and maintain this consistency.

OBJECT-ORIENTED PROGRAMMING APPROACHES

R supports multiple object-oriented programming paradigms, including S3, S4, and R6, each offering distinct advantages:

- S3 (Simple Object-Oriented Programming): This is the most common and flexible OOP system in R, relying on generic functions and method dispatch based on class attributes. It is straightforward to extend existing functions to work with new object types.
- S4 (Formal Object-Oriented Programming): S4 provides more formal class definitions, method signatures, and inheritance, offering stricter control and improved error checking. e.g., {**rtables**}.
- R6 (Reference Classes): R6 objects use reference semantics, allowing mutability and direct method binding. e.g., {**metacore**}.

OOP can significantly enhance code organization, reusability, and maintainability, especially for the complex data structures. The selection of an OOP system depends on the project's complexity, the development team's familiarity with the paradigm, and the desired level of formality and strictness required for code development.

VALIDATION STRATEGIES FOR R PACKAGES

Due to the highly regulated nature of the clinical programming domain, R packages must be validated to ensure accuracy, reproducibility and traceability, and to provide the documented evidence of the accuracy assessment for R. The recommended strategy is a risk-based approach from the Industry Initiatives of R Validation Hub.

Each package should be evaluated by specific criteria to calculate the risk assessment score and classify the risk level.

The following four criteria are described by the R validation Hub:

- Purpose
- Maintenance Good Practice (Software Development Life Cycle)
- Community Usage
- Testing

The company can customize risk metrics to ensure compliance with specific requirements.

For example, some criteria in the Sanofi {risk.assessr}:

- R CMD check
- Test coverage over 80%
- Number of dependencies <= 20
- Documentation
- License

There are several tools and initiatives to achieve its mission:

- {riskmetric} is a framework to quantify an R package's "risk of use" by assessing a number of meaningful metrics designed to evaluate package development best practices, code documentation, community engagement, and development sustainability.
- {riskassessment} is an R package containing a shiny front-end to augment the utility of the {riskmetric} package within an organizational context.
- thevalidatoR: GitHub Action that generates R Package Validation documentation

INTERACTIVE REPORTING AND DEPLOYMENT

Quarto is an open-source scientific and technical publishing system that empowers users to create dynamic reports. This project repository is hosted on GitHub and the analysis results of this paper will be generated by the Quarto dashboards with automated deployment via GitHub Pages and GitHub Actions. The interactive reports can contain the whole analysis data and results, significantly enhancing readability. This approach further improves communication and reproducibility. Furthermore, it can be continuously updated to keep pace with the evolution of R's open-source ecosystem.

CONCLUSION

The R programming language, together with the open-source ecosystem, offers significant opportunities and advantages for advancing clinical trial programming. This paper introduced the widely adopted R dependency packages, diverse programming designs that facilitate the streamlined adoption of R. Furthermore, it highlights the importance of actively engaging with the open-source community, leveraging insights from its collective experiences, and integrating knowledge into organizational practices.

REFERENCES

Hadley Wickham and Jennifer Bryan , R Packages (2e). Accessed 05 July, https://r-pkgs.org/.

Phil Bowsher, 2023. R Package Quality & Validation: Current Landscape. PharmaSUG 2023 - Paper SD-264.

pharmaverse developer. Accessed 05 July, 2025.https://pharmaverse.org/e2eclinical/developers/.

A Risk-based Approach for Assessing R package Accuracy within a Validated Infrastructure. Accessed 05 July, 2025. https://pharmar.org/white-paper/.

ACKNOWLEDGMENTS

Special acknowledgments are given to the open-source community in the pharmaceutical industry for their valuable contributions and inspiration.

The authors would like to thank the SDR team at Sanofi for their support and encouragement in exploring and adopting open-source R solutions

CONTACT INFORMATION

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

Enki Wang
Sanofi
enki.wang@sanofi.com

Longfei Li
Sanofi
longfei2.li@sanofi.com

APPENDIX A - GET DEPENDENCIES AND FUNCTIONS

Here the core function is used to get dependencies and functions for an R package.

The function `logrx:::get_used_functions(file)` can also be used for this purpose.

```
#' Analyze Package Dependencies and Function Usage
#'
#' This script installs development dependencies, loads the package, extracts
direct dependencies,
#' analyzes namespace exports/imports, and parses R source files to summarize
function usage.
#' Results are saved as CSV files in the specified result directory.

pkg_dir <- "./"
res_dir <- "./data"
pak::local_install_dev_deps(pkg_dir, ask = FALSE, dependencies = TRUE)
pkgload::load_all(pkg_dir)

# Extract package name and title
pkg_name <- pkgload::pkg_name(pkg_dir)
pkg_title <- desc::desc_get_field("Title")

# Get direct dependencies from DESCRIPTION, excluding R
direct_deps <- desc::desc_get_deps(pkg_dir) |>
  dplyr::rename(depends = package) |>
  dplyr::mutate(
    package = pkg_name,
```

```
    .before = 1L
  ) |>
  dplyr::filter(depends != "R")

# Get namespace exports
exports <- tibble::tibble(
  pkg = pkg_name,
  fun = getNamespaceExports(pkg_name)
)

# Get namespace imports
imports_01 <- getNamespaceImports(pkg_name) |>
  purrr::imap(\(x, idx){
    pkg <- NA_character_
    fun <- NA_character_
    if (idx == "") {
      pkg <- x[[1]]
      if (length(x) == 2) {
        fun <- x[[2]]
      }
    } else {
      pkg <- idx
      if (is.character(x)) {
        fun <- x
      }
    }
    tibble::tibble(
      pkg = pkg,
      fun = fun
    )
  }) |>
  purrr::list_rbind() |>
  dplyr::filter(!is.na(fun))

#' Parse R source file to extract function calls and their packages
get_function <- function(file) {
  df <- tryCatch(
    {
      df <- getParseData(parse(file = file, keep.source = TRUE)) |>
        dplyr::filter(nchar(text) > 0) |>
        dplyr::filter(token %in% c(
          "SYMBOL_FUNCTION_CALL",
          "SYMBOL_PACKAGE",
          "SPECIAL",
          "SYMBOL"
        ))
    },
    error = function(e) {
      NULL
    }
  )
  if (is.null(df)) {
    return(NULL)
  }
  # SYMBOL_PACKAGE for pkg::fun
  df1 <- df |>
    dplyr::semi_join(
```

```
      df |> dplyr::filter(token == "SYMBOL_PACKAGE"),
      by = "parent"
    )

  df2 <- df1 |>
    dplyr::arrange(parent, line1, col1, line2, col2) |>
    dplyr::filter(token %in% c(
      "SYMBOL_PACKAGE",
      "SYMBOL_FUNCTION_CALL",
      "SYMBOL"
    )) |>
    dplyr::mutate(token = dplyr::case_when(
      token == "SYMBOL_PACKAGE" ~ "pkg",
      token %in% c("SYMBOL_FUNCTION_CALL", "SYMBOL") ~ "fun"
    )) |>
    tidyr::pivot_wider(
      id_cols = parent,
      names_from = token,
      values_from = text
    ) |>
    dplyr::select(-parent)

  df3 <- df |> dplyr::setdiff(df1)

  df4 <- tibble::tibble(
    fun = character(),
    pkg = character()
  )

  text <- df3$text |> unique()

  for (s in text) {
    tryCatch(
      {
        fun <- NA_character_
        pkg <- NA_character_
        fun <- eval(as.symbol(s), parent.frame())

        if (is.primitive(fun)) {
          pkg <- "base"
        } else if (is.function(fun)) {
          pkg <- environmentName(environment(fun))
        }

        if (nchar(pkg) > 0) {
          df4 <- df4 |> dplyr::add_row(fun = s, pkg = pkg)
        }
      },
      error = function(e) NULL
    )
  }

  dplyr::bind_rows(df2, df4)
}

# Load dependencies to search path
ns <- direct_deps$depends
```

```
last_load_ns <- c("magrittr", "rlang")
ns1 <- ns[!ns %in% last_load_ns]
for (pkg in ns1) {
  tryCatch(
    {
      library(pkg, character.only = TRUE, quietly = TRUE)
    },
    error = function(e) NULL
  )
}
for (pkg in last_load_ns) {
  tryCatch(
    {
      library(pkg, character.only = TRUE, quietly = TRUE)
    },
    error = function(e) NULL
  )
}

# Parse all R files in package for function usage
imports_02 <- fs::dir_map(
  fs::path(pkg_dir, "R"),
  \(file)  {
    get_function(file)
  }
) |>
  purrr::compact() |>
  purrr::list_rbind()

imports <- dplyr::bind_rows(imports_01, imports_02)

# Summarize function usage by package
fun_stat <- imports |>
  dplyr::filter(pkg != pkg_name) |>
  dplyr::group_by(pkg, fun) |>
  dplyr::summarise(n = dplyr::n(), .groups = "drop") |>
  dplyr::arrange(-n, pkg, fun) |>
  dplyr::mutate(
    package = pkg_name,
    .before = 1L
  )

# Write results to CSV
write.csv(direct_deps, file.path(res_dir, paste0(pkg_name,
"_dependencies.csv")),
  row.names = FALSE
)
msg <- sprintf("%s is created", file.path(res_dir, paste0(pkg_name,
"_dependencies.csv")))
cat(msg, "\n")

write.csv(fun_stat, file.path(res_dir, paste0(pkg_name, "_functions.csv")),
  row.names = FALSE
)
msg <- sprintf("%s is created", file.path(res_dir, paste0(pkg_name,
"_functions.csv")))
cat(msg, "\n")
```