Paper OS05

Leveraging on R packages for HTA Analysis & Reporting: Our Developer Journey

Jean Muller, MSD, Paris, France
Wenyu Li, MSD, Zürich, Switzerland

ABSTRACT

In the rapidly evolving landscape of Health Technology Assessment (HTA), R provides early access to new and/or advanced peer-reviewed methodologies essential for timely and efficient Analysis & Reporting (A&R) deliveries. This presentation will showcase our development of customized R packages designed to enhance our HTA A&R toolkit. The emphasis is on the importance of maintaining a systematic structure and adhering to good programming practices to establish standards and ensure long-term code maintainability. We discuss the advantages and drawbacks of the R package format, with a particular focus on the trade-off between package functionality and interdependencies. By sharing our journey in R development, the aim is to provide valuable insights to assist other R developers in traceable, reproducible, and reliable HTA A&R deliverables.

INTRODUCTION

Health Technology Assessment (HTA) is a systematic, multidisciplinary, and transparent process designed to evaluate the properties and effects of health technologies and interventions. Its primary function is to serve as a bridge between the world of clinical research and the complex realities of healthcare decision-making. HTA organizations around the globe use this evidence-based approach to inform reimbursement and coverage decisions, which have impacts on patient access to medicines, medical devices, and procedures.

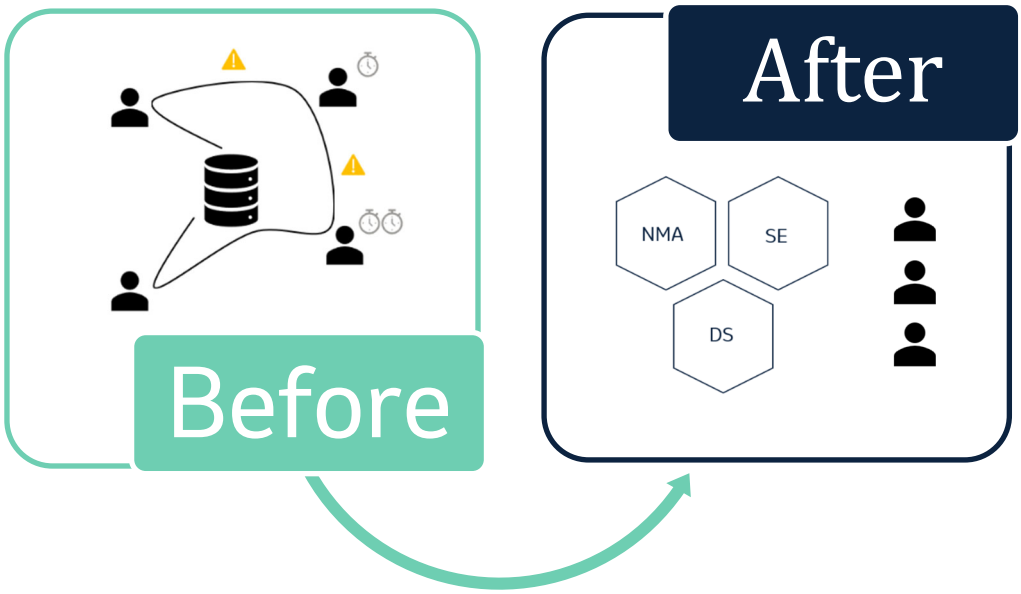
In this environment, the R programming language has emerged as a suitable tool for some HTA Analysis and Reporting (A&R). As a free and open-source language, R provides access to an ecosystem of packages that include recent peer-reviewed statistical methodology.

This paper details a developer journey focused on transitioning from a collection of R analysis scripts to a cohesive, standardized, and validated ecosystem of internal R packages which can be framed as a standards library (SL). This journey was initiated to simplify maintenance across projects. Building internal R packages, as opposed to other code structures, allows the team to leverage R tools (devtools, testthat, roxygen2, renv) supporting best practices such as standardization, documentation, and validation. This approach fosters an open-source culture that promotes collaboration and quality while also providing the option of properly sharing code to the R community when ready to do so.

This work has been developed from a statistical programming perspective, focusing on practical implementation challenges and solutions. The following sections will elaborate on this journey, beginning with our first steps, moving through the technical challenges encountered and our solutions, demonstrating practical applications with concrete examples, and concluding with reflections on the lessons learned and the future of HTA A&R in an era of technological change.

FROM FIRST STEPS TO SOFTWARE DEVELOPERS

We evolved from shared R scripts that were copied from one project to another to a cohesive, validated ecosystem of internal R packages purpose-built to support HTA submissions.



The shared drive system was born from a desire to share knowledge. Risks were mitigated because as a professional programming team, we already had strong, disciplined practices for every project. We enforced modularity through organized scripts, we ensured reproducibility by documenting our environments, we validated our outputs, and we collaborated on code. These practices worked well and ensured quality on a project-by-project basis. The manual effort required to maintain these good practices, however, became a barrier to scaling effectively. We looked both inside and outside our organization to see what solutions could help, and it became clear that R packages and the tools built around them should be explored.

We started by creating internal R packages (Wickham & Bryan, R Packages (2e), 2025), each dedicated to a specific analysis or delivery type (e.g., descriptive statistics, survival extrapolation, network meta-analysis), consolidating shared utilities and data into versioned, reusable packages with clear ownership. In parallel, the project-level scripts had systematic version logs (sessionInfo()) to support reproducibility across environments.

We continued to build using good software development practices by not repeating ourselves “Don’t Repeat Yourself (DRY)” (Hunt & Thomas, 2019), letting test drive development and considering documentation as a feature. Unit tests with testthat (Wickham, testthat: Get Started with Testing, 2011) verifying both success and failure allowed us to automate quality checks and scale testing with Continuous Integration (CI) running R CMD check and coverage regularly.

From a tooling perspective, we adopted Git with a feature-branch workflow (Atlassian), protected the main branch, and mandated peer-reviewed pull requests to maintain a high-quality, auditable change history. Plus, passing the automated checks was made a mandatory step for the code to be included in the main branch.

Clear project management using Jira/GitHub Projects established ownership, acceptance criteria, and a definition of done with review. Internal distribution through a secure Git server standardized release; the combination of Pull Requests (PR) and environment renv-like snapshots (Wickham & Ushey, renv: Project Environments, 2025) provided end-to-end assurance for HTA deliverables.

With these practices stabilized, we set a long-term pathway toward CRAN readiness for eligible utilities. Ultimately, the R ecosystem provided the automation and tooling necessary to scale our established culture of maintainable, testable, and reproducible software across the entire team, enabling faster and more reliable delivery.

GOOD PRACTICES

During this move from R scripts to package ecosystem, we listed good practices that added value for us in the long term.

STAKEHOLDER ENGAGEMENT AND PROJECT PLANNING

Good practice	Risk mitigated	Impact
Regular stakeholder discussions to refine requirements and prioritize features	Misaligned scope, rework due to misunderstood needs	High
Pilot studies of features in real analysis contexts	Building unusable or impractical features; late discovery of usability issues	High
Clear project management via Jira/GitHub Projects/MS Planner	Unclear ownership, missed deadlines, scope drift	High
Definition of Done (tests, docs, review)	Incomplete features slipping through	High

STRUCTURED DESIGN

Good practice	Risk mitigated	Impact
Adopt base R style guide and unified coding conventions	Inconsistent code, harder reviews, increased onboarding time	High
Apply Single Responsibility Principle (SRP) (Martin, 2002) (single, well-defined function responsibilities)	Tightly coupled code, hard-to-test and hard-to-change functions	Very high
Group related functions and use clear naming (e.g., utils-plotting.R, action-verb-*.R)	Scattered logic, low cohesion; duplicated effort	Medium
Long-term CRAN readiness for non-proprietary utilities	Noncompliance with ecosystem standards	Medium
Export a select number of functions	Breaking users' workflow by changing internal functions; non-stable Application Programming Interface (API)	High
Minimize dependencies (essential in Imports; optional in Suggests)	Dependency bloat, increased breakage surface, slow installs	Very high
Use explicit namespace calls (package::function())	Name masking and ambiguity leading to subtle bugs	Moderate
Object-Oriented Programming (S3/S4)	Unpredictable function behavior, code that is hard to maintain and extend	Very high
Unit tests with testthat combined with Github Actions or Jenkins	Silent regressions; fragile refactors	Very high

VERSION CONTROL

Good practice	Risk mitigated	Impact
Feature Branch Workflow for all changes	Untracked or conflicting changes on main; merge chaos	Very high

Protected main branch (no direct commits)	Instability in production-ready code; accidental hotfixes	Very high
Mandatory Pull Request (PR) with peer review	Low-quality, unreviewed code entering main	Very high
Automated checks on PR (R CMD check, tests, style)	Human error in manual checks; regressions	Very high
Small, atomic commits with descriptive messages	Hard-to-revert changes; poor change traceability	High
Delete merged feature branches	Repository clutter; confusion over active branches	Low

PROJECT LEVEL EXECUTION

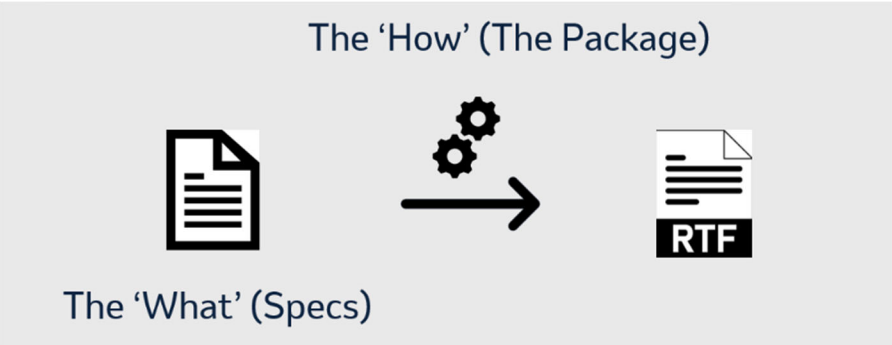
Good practice	Risk mitigated	Impact
Project-level snapshots of dependencies (like R's renv approach for reproducibility)	"Works on my machine" and non-reproducible results	Very high
HTML logs of exact dependency versions	Audit gaps; difficulty reproducing historic runs	High
Internal Production Git server for controlled distribution	Unauthorized access; inconsistent package versions	High

Collectively, these practices moved us to a disciplined workflow and allowed us to establish reliable guardrails that scale across teams and projects.

DOMAIN SPECIFIC APPLICATIONS

In this section, we discuss three R packages: Descriptive Statistics, Survival Extrapolation, Network Meta-Analysis. Each addresses a distinct analytical need. We outline the specific requirements, design choices, and challenges for each.

DS: DESCRIPTIVE STATISTICS (OVERALL CHALLENGES)



- Design Choice: Hard-coding variable names (e.g., AVAL, PARAMCD) is not recommended; functions should ideally allow users to map the specific variables in their ADaM datasets.
  ➔ Use defaults but provide flexibility to change variable names

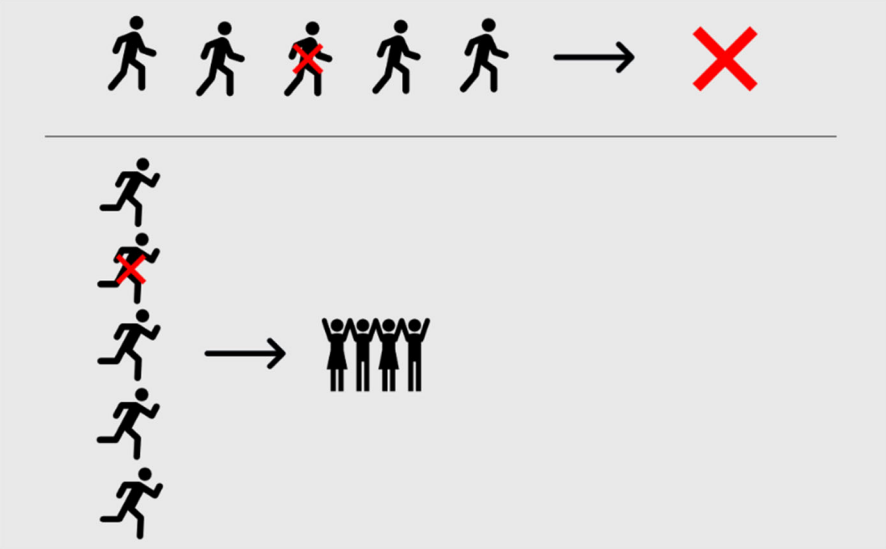
- Challenge: Figure generation has inherent differences across Operating Systems (OS).

➔ Leverage on vdiffr::expect_doppleganger() to solve the issue (Henry, Lin Pedersen, Luciani, Decorde, & Vaudor, 2025)

- Design Challenge: Analysis Specification. The key is to create a system where a statistician can specify the table contents without having to write complex R code. This separates the "what" (the table shell) from the "how" (the R code that generates it).
  ➔ Define requirements in machine readable file (Danenberg, Wickham, Csárdi, & Eugster, 2025)

- Programming Challenge: Table Formatting and RTF encoding. Translating a data frame of statistics into a multi-level, indented RTF table that exactly matches a predefined shell and ensures all special characters display correctly. This involves complex logic for headers, row indentation, and cell formatting.
  ➔ r2rtf package features
  ➔ Defensive programming and complex validation plan defined before development

SE: SURVIVAL EXTRAPOLATION

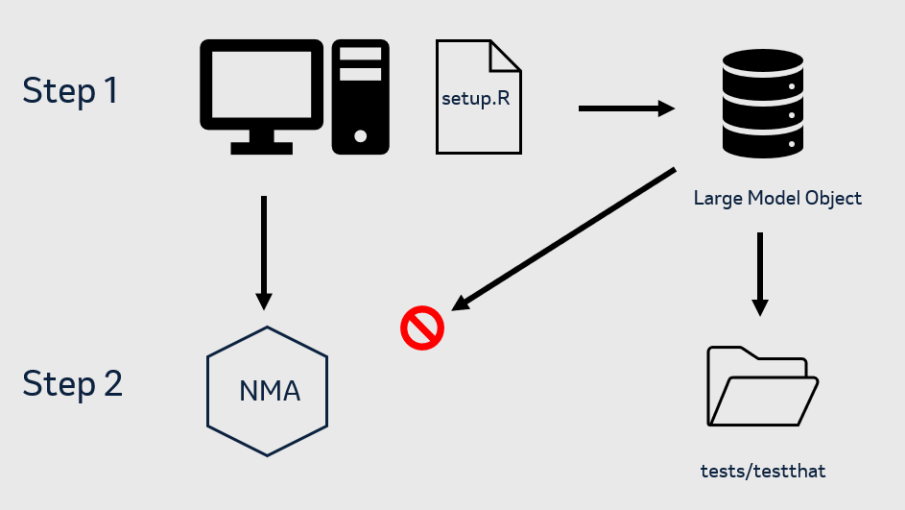


- Challenge: The package shouldn't just fit one model. It should be able to automate the full sequence: fit multiple distributions (Weibull, log-normal, etc.) and generate a standard set of diagnostic plots and summary tables. We initially used “for loops” which slowed down the workflow.
  ➔ Replace for loops with R’s apply/map family (e.g., lapply, Map) and, when appropriate, use the future.apply package (Bengtsson, 2021) to parallelize processing.

- Programming Challenge: Handling Model Convergence. Parametric survival models often fail to converge. Code must be robust enough to catch these errors, report the failure, and continue with the next model, rather than crashing the entire analysis.
  ➔ Leverage on tryCatch() functionality for robust workflow.

- Programming Challenge: Extrapolation & Visualization. A complex logic was needed to append the piece-wise model prediction to the Kaplan-Meier curve. Visualizing this extrapolation along with its confidence interval on a Kaplan-Meier plot requires careful data manipulation.
  ➔ Monitor engine development, as the desired option becomes available.

NMA: NETWORK META ANALYSIS



- Design Challenge: Data Integration. Design must be a robust process for combining trial Individual Patient Data (IPD) with diverse forms of Aggregated Data (AgD) from the literature (e.g., hazard ratios, counts, odds ratios). Standardizing the different data sources is required.
  - ➔ Set a standardized format for external data sources, start workflow with a data processing function assigning a class to the returning object

- Programming Challenge: Simplifying Bayesian Models. The multinma package is powerful but complex. The programming challenge is to create wrapper functions that simplify the process by providing sensible defaults for Bayesian settings (priors, iterations, etc) while still allowing advanced users to override them.
  - ➔ 2 layers of functions. A flexible verb layer allowing "…" input parameter to give users access to parameters of underlying functions and a reporting layer focused of approved scenarios

- Programming Challenge: Testing file management. Multinma modelling takes significant time to run and fit objects are large which results in a large package.
  - ➔ Leverage tests/testthat/setup.R file to build test models. Large files are available locally for testing but not part of the final package while still fully reproducible.

All three R Packages follow a similar high-level workflow of data ingestion, statistical modeling, and results display generation; the specific technical hurdles are unique to each domain. For descriptive statistics, the primary challenge lies in specification and presentation: translating a complex request into a perfectly formatted report. For survival extrapolation, the focus shifts to managing the modelling part of the workflow, from automating a sequence of model fits and handling convergence failures to visualizing long-term predictions. Finally, the network meta-analysis package confronts the most complex challenges in data integration and model management, requiring a framework to harmonize disparate data sources, Individual Patient Data (IPD) and Aggregated Data (AgD), and simplify the execution of computationally intensive Bayesian models.

While a consistent set of tools for validation, data handling, and reporting are the foundation of the A&R toolkit, each package must implement tailored solutions to address the unique scientific and programming problems of its specific application. The discussion then naturally becomes about the ecosystem of R packages.

DISCUSSION

Building a successful A&R toolkit is an exercise in ecosystem design, not just individual R package development. The goal is to create a set of cohesive, interoperable tools that feel like a single, unified system to the end-user, rather than a collection of isolated scripts. This requires consideration of the ecosystem's architecture and governance to sustain it.

ECOSYSTEM ARCHITECTURE

Package size and scope: Each package should be large enough to be useful on its own but small enough to be maintainable. The current division has a separate package to address a distinct and substantial analytical domain. This modularity prevents creating a single, monolithic package that is difficult to test, document, and update while providing flexibility for each team to implement domain specific solutions.

Interdependencies: The three-package structure (Descriptive Statistics, Survival Extrapolation, Network Meta-Analysis) provides a clear separation in terms of analysis. A key architectural question we are asking ourselves is whether the three packages should depend on a fourth, internal utility package. This utility package would contain all shared functionality, such as CDISC data import/validation functions, custom ggplot2 themes, and wrappers for the r2rtf package. An additional dependency would mean more constraints and an additional maintainer.

External dependencies: The ecosystem relies on powerful external packages like ggplot2, flexsurv, multinma, and r2rtf. This is a strength, but also a risk. An update to a dependency could break the code. This risk is currently managed through rigorous use of renv-like snapshots to lock down tested package versions at the project level, ensuring reproducibility.

GOVERNING THE ECOSYSTEM: KEY QUESTIONS FOR EXPANSION

As the A&R toolkit proves its value, requests for new features and packages arise. A clear governance process is needed to sustain growth and maintain quality. Before adding a new package to the ecosystem, the team could ask the following key questions:

- Is the scope justified? Does the new functionality represent a distinct, substantial domain of analysis? Or could it be added to an existing package? This prevents the proliferation of tiny, single-function packages that increase maintenance overhead.
- What are the new dependencies? Does the proposed package introduce new external dependencies? Each new dependency adds to the validation burden and increases the long-term maintenance risk.
- Does it overlap with existing functionality? Does the new package solve a problem that is already addressed elsewhere in the ecosystem? This ensures a "single way" to perform a given task, reducing user confusion and redundant code.
- Who is the customer and what is the interface? Will the new package be used by the same audience? Its user interface (the function names, arguments, and workflow) should be consistent with the existing packages to provide good user experience.
- What is the maintenance plan? Who is responsible for maintaining this new package? A package without a clear owner could be a liability. There must be a commitment to support it with testing, documentation, and updates.

CONCLUSION

Structuring R code into a well-managed ecosystem of internal packages, leveraging on modern software development practices, including disciplined version control, comprehensive automated testing, and robust dependency management, is an effective strategy for producing traceable, reproducible, and reliable HTA A&R deliverables plus it makes scaling easier.

While tools like Git and R packages are enablers, true success is rooted in a team culture that embraces collaboration, constructive peer review, and a sense of shared ownership over the codebase. The structures of open-source software development provide an excellent model for this, fostering transparency and collective improvement.

With the transition from shared R scripts to a structured R package ecosystem now complete, we look forward to the next phase: integrating Generative Artificial Intelligence (GenAI) into our development workflow as we are already seeing its capability to enhance the way we create, review, and maintain R packages.

REFERENCES

Atlassian. (n.d.). *Feature Branch Workflow.* Retrieved from Atlassian Git Tutorials: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow

Bengtsson, H. (2021). A Unifying Framework for Parallel and Distributed Processing in R using Futures. *The R Journal, 13*(2), 208--227. doi:10.32614/RJ-2021-048

Danenberg, P., Wickham, H., Csárdi, G., & Eugster, M. (2025). roxygen2: In-Line Documentation for R. Retrieved from https://roxygen2.r-lib.org/

Google DeepMind. (2025). *Gemini 2.5 Pro (25 March version) [Large language model]*. Retrieved from https://gemini.google.com.

Henry, L., Lin Pedersen, T., Luciani, T., Decorde, M., & Vaudor, L. (2025). vdiffr: Visual Regression Testing and Graphical Diffing. Retrieved from https://vdiffr.r-lib.org/

Hunt, A., & Thomas, D. (2019). *The Pragmatic Programmer: Your Journey to Mastery* (20th Anniversary ed., 2nd ed. ed.). Boston: Addison-Wesley Professional.

Martin, R. C. (2002). *Agile Software Development, Principles, Patterns, and Practices.* Upper Saddle River, NJ:

Prentice Hall.
Wickham, H. (2011). testthat: Get Started with Testing. *The R Journal*, 5--10. Retrieved from https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf
Wickham, H., & Bryan, J. (2025). Retrieved from R Packages (2e): https://r-pkgs.org/
Wickham, H., & Ushey, K. (2025). *renv: Project Environments*. Retrieved from https://rstudio.github.io/renv/.

ACKNOWLEDGMENTS

We would like to express our sincere gratitude to MSD for providing us with the opportunity to attend this conference and present our journey. We also extend our thanks to Qian Wang and Carl Herremans for their guidance and review. In addition, we are grateful for the collaboration and support of our HTA Statistics colleagues, whose contributions made the development possible.

CONTACT INFORMATION

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

Author Name: Jean Muller
Company: MSD
Affiliation: Statistical Programming
Email: jean.muller@msd.com

Author Name: Wenyu Li
Company: MSD
Affiliation: Statistical Programming
Email: wenyu.li1@msd.com