A bridge between Sponsor and CRO: My experience sharing an R package

Iolo Williams

iolo.williams@veramed.com

PHUSE SDE, Cambridge, 2026

Veramed

Agenda

01 Sponsor-Veramed collaboration

02 Overview of R packages

03 Existing methods to share R packages

04 Approach and solution to share the R package

05 Evolution of R package installation at Veramed

06 Key Learnings

Overview of Sponsor – Veramed collaboration

To comply with FDA guidance on safety reporting for investigational new drug studies, sponsors must conduct aggregate analyses on blinded, ongoing study data to identify potential safety signals that could necessitate unblinded review

The sponsor will conduct blinded analysis and delegate unblinded analysis to Veramed

To perform this, Veramed will need an R package developed from the sponsor, which I helped develop

R package implements a complex statistical method

R Packages: An Overview

Collection of R code, data, documentation and tests

Easy to share with R users

> 22000 public packages on Comprehensive R Archive Network (CRAN)

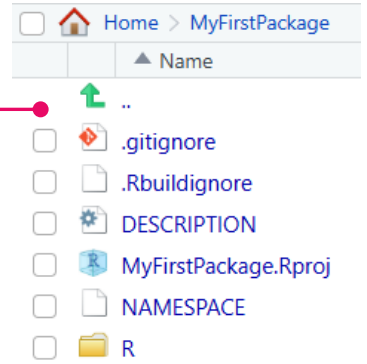
Made up of directories and files & follows a standard convention:

- DESCRIPTION and NAMESPACE files
- code in R/, data in data/, documentation files in man/, tests in tests/

Basic R package development

R code to create your first package

```
install.packages("devtools")
library(devtools)
create_package("MyFirstPackage")
```



Name	Type	Purpose
.gitignore	File	Files to be ignored by Git
.Rbuildignore	File	Files to be ignored when building the package from source
DESCRIPTION	File	Package metadata
MyFirstPackage.Rproj	File	File that makes this directory an RStudio Project
NAMESPACE	File	Declares functions that the package imports & exports
R	Directory	Contains the .R files

Essential *usethis* functions for package development

`use_r()` – create new .R script

`use_testthat()` – initialise testing infrastructure

`use_package()` – add dependencies to DESCRIPTION

Essential *devtools* functions for package development

`document()` – generate documentation files

`load_all()` – load package content

`check()` – run package checks

`install()` – install package locally

R package states

Source package

- When developing a package, we work in its source form
- Key components: DESCRIPTION, NAMESPACE, R/, man/, tests/
- Rtools required to build and compile source package

Binary package

- A pre-compiled version of the source package
- No compiling or building tools required to install
- Windows binary file ends in .zip (macOS – .tgz file)

R Packages

Can be distributed as either source or binary

How to share an R package?

Publish on CRAN

Common code sharing methods include GitHub and Managed File Transfer systems

Sponsors may not yet have policies to share code externally using these methods

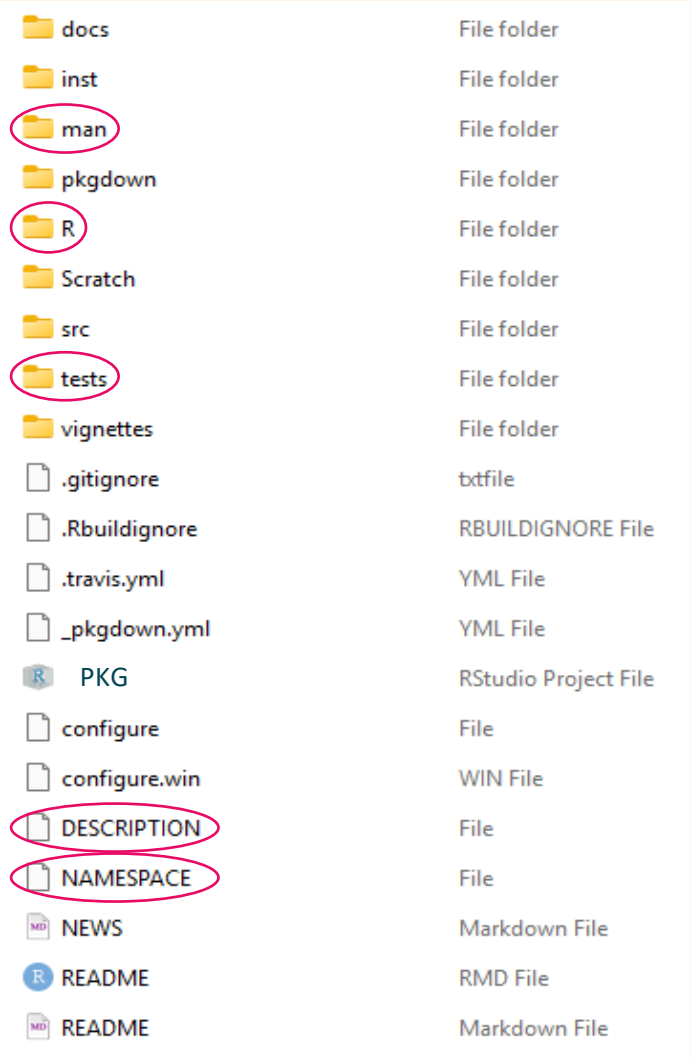
Innovative approach needed to distribute code

Initial approach: Source package installation

- As a contributor, I had access to the sponsor's package development environment
- Initial idea: install package from source in the target R environment
- Constraint: Rtools inaccessible in the target R environment

```
> pkgbuild::has_build_tools()
[1] FALSE
```

- Implication: installation not possible
- Next step: build and share a binary package



Solution: Build a binary package

Retrieve source package from sponsor's R environment

Build a binary package using local RStudio (with Rtools)

Transfer and install in the target R environment

Key benefit: avoids installation with Rtools in the target R environment

Implementation: building the binary package

R code to build a binary package

```
unzip("~/PKG.zip", exdir = "~/PKG")
setwd("~/PKG")
devtools::build(binary = TRUE)
```

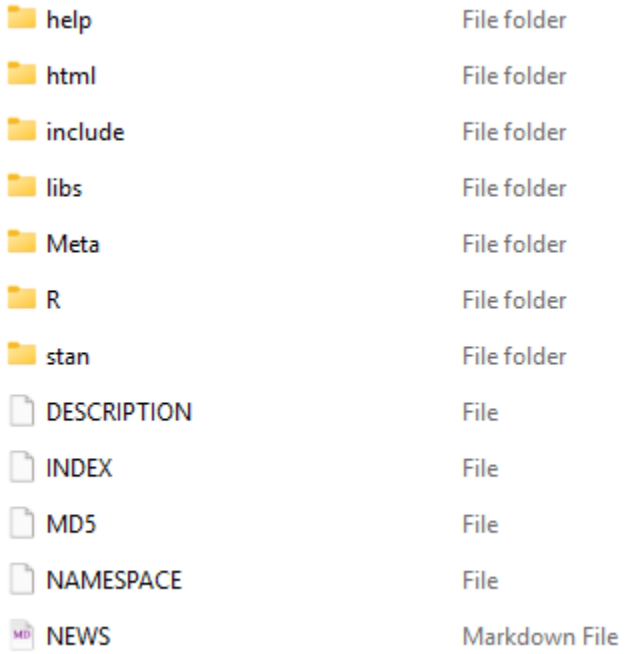
PKG_1.0.0 Compressed (zipped) Folder 1,560 KB

R code to install a binary package

```
install.packages("~/PKG_1.0.0.zip",
repos = NULL, type = "binary")
```

```
Installing package into 'Z:/Users/iolo.williams/R packages'
(as 'lib' is unspecified)
package  PKG  successfully unpacked and MD5 sums checked
```

Binary package structure



Additional complexity: Differing R versions

- Sponsor – R 3.6.2
- Veramed – R 4.4.0
- Differing package versions
- Example package output in Sponsor R environment:

```
p(x)= 0.247 (0.131, 0.349)
```

- Example package output in Veramed R environment:

```
p(x)= 0.248 (0.138, 0.377)
```

- Unlikely that R 4.4.0 can compile all packages from R 3.6.2

Package	R 3.6.2	R 4.4.0
rstan	2.21.5	2.32.7
Rcpp	1.0.10	1.0.14
dplyr	1.0.8	1.1.4
ggplot2	3.5.1	3.5.2
tidyr	1.2.0	1.3.1
purrr	0.3.4	1.0.4

Attempt to replicate the Sponsor's R environment

- *renv* package helps create reproducible environments for R projects
- The lockfile contains metadata on packages that can be reinstalled in a new environment
- Create a lockfile that captures package metadata in R 3.6.2
- Restore these dependencies in the target R environment

Essential *renv* functions to help reproduce R environments

`init()` – initialise renv in a project setting

`snapshot()` – update lockfile with current dependencies

`restore()` – restore dependencies from the lockfile

dplyr metadata from the Sponsor R environment

```
"dplyr": {
  "Package": "dplyr",
  "Version": "1.0.8",
  "Source": "Repository",
  "Repository": "RSPM",
  "Hash": "ef47665e64228a17609d6df877bf86f2",
  "Requirements": [
    "R6",
    "generics",
    "glue",
    "lifecycle",
    "magrittr",
    "pillar",
    "rlang",
    "tibble",
    "tidyselect",
    "vctrs"
  ]
},
```

```
> renv::restore()
- renv activated -- please restart the R session.
The following package(s) will be updated:

# RSPM ------------------------------------------------------------
- abind            [repo: CRAN -> RSPM; ver: 1.4-8 -> 1.4-5]
- askpass          [repo: CRAN -> RSPM; ver: 1.2.1 -> 1.1]
- backports        [repo: CRAN -> RSPM; ver: 1.5.0 -> 1.4.1]
- bayesplot        [repo: CRAN -> RSPM; ver: 1.12.0 -> 1.9.0]
- BH               [repo: CRAN -> RSPM; ver: 1.87.0-1 -> 1.75.0-0]
- bit              [repo: CRAN -> RSPM; ver: 4.6.0 -> 4.0.4]
```

```
- rstan            [repo: CRAN -> RSPM; ver: 2.32.7 -> 2.21.5]
- Rcpp             [repo: CRAN -> RSPM; ver: 1.0.14 -> 1.0.10]
- dplyr            [repo: CRAN -> RSPM; ver: 1.1.4 -> 1.0.8]
- ggplot2          [repo: CRAN -> RSPM; ver: 3.5.2 -> 3.5.1]
- tidyr            [repo: CRAN -> RSPM; ver: 1.3.1 -> 1.2.0]
- purrr            [repo: CRAN -> RSPM; ver: 1.0.4 -> 0.3.4]
```

Challenges of restoring R 3.6.2 packages with renv

Example: *renv* trying to install *MASS* version 7.3-57

```
Error: Error installing package 'MASS':
================================

* installing *source* package 'MASS' ...
** this is package 'MASS' version '7.3-57'
** package 'MASS' successfully unpacked and MD5 sums checked
** using staged installation
** libs
using C compiler: 'gcc.exe (GCC) 14.3.0'
gcc  -I"C:/PROGRA~1/R/R-45~1.2/include" -DNDEBUG     -I"C:/rtools45/x86_64-w64-mingw32.static.posix/include"
-O2 -Wall -std=gnu2x  -mfpmath=sse  -msse2  -mstackrealign   -c MASS.c -o MASS.o
MASS.c:37:23: error: unknown type name 'Sint'; did you mean 'int'?
   37 | VR_sammon(double *dd, Sint *nn, Sint *kd, double *Y, Sint *niter,
      |                       ^~~~
      |                       int
```

- Older *MASS* package version contains C code that modern R does not support
- Similar errors with *mgcv*, *nlme* packages (likely other packages have similar issues)
- Several packages in the R 3.6.2 lockfile cannot be installed on current R versions
- To fully reproduce the original environment, should we install R 3.6.2?

Should we install R 3.6.2?

Remains accessible

- Technically possibly
- Available from CRAN

Security concerns

- April 2024 – significant vulnerability found in R, affecting versions prior to 4.4.0.
- Installation of R 3.6.2 introduces a risk to information security

Solution?

Package could be redeveloped in a newer R version

Evolution of R package installation at Veramed

Autumn 2025

Veramed built new infrastructure to install packages for study teams to use

Risk Assessment:

- Identifies all packages (primary & dependencies)
- Creates a risk assessment for each package

Installation Qualification & Operational Qualification:

- Installs packages & runs inbuilt unit tests
- Verifies that packages are installed cleanly and run successfully

Deployment:

- IT run a script to bundle primary and dependency packages
- Package deployment can be made globally or to study specific teams

This will be the primary method to install the sponsor package in the future

Key Learnings

Package development is complex, but can be simplified

Multiple viable ways of sharing R code

Differences in R versions introduce hidden complexity

Reproducibility depends on environmental consistency

Resources

https://www.fda.gov/regulatory-information/search-fda-guidance-documents/investigator-responsibilities-safety-reporting-investigational-drugs-and-devices

https://www.datacamp.com/tutorial/r-packages-guide

https://r-pkgs.org/

https://cran.r-project.org/

https://rstudio.github.io/renv/

Thank you for listening

iolo.williams@veramed.com