

Developing Open-Source Software

Nathaniel E. Helwig

Associate Professor of Psychology and Statistics
Core Member of the Data Science Initiative
University of Minnesota



Quantitative & Psychometric Methods Seminar
University of Minnesota, Twin Cities
November 12, 2025

Table of Contents

1. The Art of Coding
2. Good and Bad Practices
3. Publishing an R Package
4. Higher Level Languages

Table of Contents

1. The Art of Coding
2. Good and Bad Practices
3. Publishing an R Package
4. Higher Level Languages

If You Develop (and Publish) It, Users Will Come...

CRAN Package Downloads

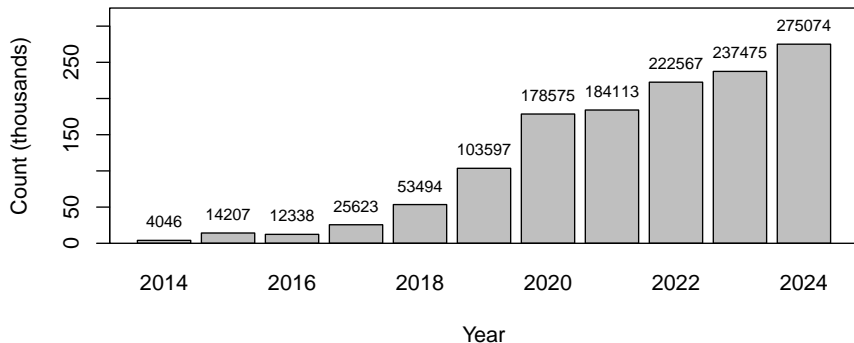


Figure 1: My CRAN Packages Downloads from 2014 to 2024.

Measuring the Impact of Software

The figure (on the previous slide) and below information was obtained from data published at: <http://cran-logs.rstudio.com>

From 2014 to 2024, I published 10 R packages on CRAN

- 1.3+ million downloads in total
- 191 different countries (US, CN, JP, DE, GB, KR, CA, FR, ...)
- 88 different OS (mingw32, linux-gnu, and many darwin versions)

Take Home Point:

R is a great way to disseminate your work to a wide audience!

What is Coding?

ChatGPT's Answer:

“Coding (also called programming) is the process of writing instructions that tell a computer what to do.”

Google Gemini's Answer:

“Coding, often used interchangeably with computer programming, is essentially the process of creating a set of instructions—known as code—that a computer can understand and execute to perform a specific task.”

Anthropic Claude's Answer:

“Coding is writing instructions for computers using programming languages! It's how we create software, apps, websites, and games.”

What is Vibe Coding?

ChatGPT's Answer:

“Vibe coding is a relatively new concept in software development that centers around using large language models (LLMs) and other AI tools to generate code from natural-language descriptions, rather than writing every line of code manually.”

Google Gemini's Answer:

“Vibe coding is a recently popularized concept and software development practice that involves using Artificial Intelligence (AI) tools, particularly Large Language Models (LLMs), to generate functional code from natural language prompts (i.e., describing what you want in plain language).”

Anthropic Claude's Answer:

“Vibe coding is a newer, playful term for a more relaxed, experimental approach to programming! It's basically coding by feel and intuition rather than rigorous planning.”

To Vibe or Not to Vibe Code?

This talk will be focused on “traditional coding”, i.e., not vibe coding.

- Discusses good and bad practices
- Pros and cons of different choices

I am not against vibe coding (I will talk a bit about it next month).

- I have effectively used vibe coding principles in my work
- IMHO you should understand traditional coding first...

Note that you can effectively combine traditional and vibe coding.

Scripts versus Functions

R (and most languages) distinguish between two types of code:

- **Scripts:** several lines of code intended to be run sequentially
- **Functions:** code that manipulates inputs to produce outputs

Most work could be done via script files only, but this isn't ideal...

- Functions are more portable and generalizable
- Functions are much easier to debug and edit

There is some art to deciding what to make a script versus a function

- Quantitative analyses consistent of several sequential steps
- Subtasks should be implemented via self-contained functions

Goals of this Talk

The primary goal of this talk to encourage you to develop and publish high quality code relevant to your own work.

Are you writing new code for manuscript or project?

- Make a coherent plan for writing your code
- Develop and publish a general purpose package

Do you have old (unpublished) code lying around?

- Make a coherent plan for editing your code
- Develop and publish a general purpose package

Table of Contents

1. The Art of Coding
2. Good and Bad Practices
3. Publishing an R Package
4. Higher Level Languages

Limited Information on Good Coding

Volumes have been written about how to write well:

- Strunk, W., & White, E. B. (1979). *The elements of style*. (3rd ed.). New York: Macmillan.
- Bem, D. J. (2019). Writing an empirical article. In R. J. Sternberg (Ed.), *Guide to publishing in psychology journals* (2nd ed., pp. 3-13). Cambridge University Press.

Comparatively little has been written about good coding:

- Nagler J. (1995). Coding Style and Good Computing Practices. *PS: Political Science & Politics*, 28(3), 488-492. doi:10.2307/420315
- Pruijm, R., Gîrjău, M.-C., & Horton, N. J. (2023). *Fostering Better Coding Practices for Data Scientists*. Harvard Data Science Review. MIT Press

The Four C's of Good Coding

Your code should be...

- **Correct** - triple check that all lines work as intended
- **Commented** - every (non-obvious) line should be commented
- **Clean** - easily readable, understandable, and editable
- **Concise** - simple and efficient (nothing more, nothing less)

Code that satisfies all four C's is considered “good code” in my book.

Order of importance: correct > commented > clean > concise

Writing Code that is Correct

Every line of code (in a script or a function) needs thorough vetting.

- Want a 100% guarantee that each line performs as intended
- Requires careful inspection of what is produced by each line

Reminders:

- Scripts should be tested in full from a clean environment
- Functions often abide by Vegas rules (what happens in a function, stays in a function)

When writing a function...

- Make sure to check all inputs (users WILL make mistakes)
- Use the `browser()` function to debug within R functions

Writing Code that is Commented

I cannot stress enough the importance of well-commenting your code.

- If you don't comment it, you will forget how/why you did things
- Lack of comments = lots of questions (for published code)

Both scripts and functions benefit from including thorough comments.

- Some folks think that commenting functions is less important
- See my above comments (which apply to scripts and functions)

Comments should explain intentions and actions of code.

- What each line/chunk of code is intended to accomplish
- Inputs/outputs produced by each function that is called

Writing Code that is Clean

Clean code requires careful consideration to style and consistency.

- Be intentional and consistent with names, spacing, formatting, etc.
- Deviations from normal standards will be noticed and scrutinized

Include white spaces to make your code more easily readable.

- `xbar <- mean(x)` versus `xbar<-mean(x)`
- `mod <- lm(y ~ x + z, data = mydata)` versus
`mod<-lm(y~x+z,data=mydata)`

Organize your script or function into sections with distinct goals.

- **Simulation Script:** Design, Analyses, Results
- **Data Analysis Script:** Preprocessing, Analyses, Results
- **Function Files:** Inputs, Manipulations, Outputs

Writing Code that is Concise

As a final step, optimize and streamline your code.

- Remove any unnecessary code (e.g., loading unused packages)
- Eliminate redundancies (e.g., centering already centered data)

Enhance your code for efficiency and scalability

- Initialize sufficient statistics and vectorize code whenever possible
- Use the `Rprof()` and `summaryRprof()` functions for profiling

Changes to code (for efficiency or brevity) need to be carefully vetted.

- Make sure each change produces same (intended) results
- Ideally you have written script files that test your functions

Table of Contents

1. The Art of Coding
2. Good and Bad Practices
3. Publishing an R Package
4. Higher Level Languages

Publishing an R Package on CRAN*

Steps required to publish R package on CRAN:

1. Prepare package (.R, .Rd, .Rda, DESCRIPTION, NAMESPACE)
2. Build package: `R CMD build grpnet`
3. Check package: `R CMD check grpnet --as-cran`
4. Submit package: <https://cran.r-project.org/submit.html>
5. Fix and resubmit (for first submissions)

Getting the first version of your package on CRAN can be effortful.

- CRAN Core Team Members thoroughly review first submissions
- Updates to package are much easier (if you pass checks in #3)

*CRAN = Comprehensive R Archive Network <https://cran.r-project.org/>

Publishing an R Package Elsewhere

CRAN is not the only place that you can publish R packages.

- GitHub is a very popular option for publishing code
- Software journals (e.g., JSS or The R Journal)

Things to consider when choosing a place to publish your code:

- Does your code get a permanent web address?
- Does your code get a digital object identifier (DOI)?

Publishing (and maintaining) code on CRAN is worth the effort.

- Requires more rigorous documentation, updating, etc.
- Benefit is that your code reaches a wider audience

Components of an R Package

An R package is a directory with four required files/folders:

- **DESCRIPTION** – file containing basic info about package
- **NAMESPACE** – file exporting functions and registering methods
- **man** – folder containing R documentation files (.Rd)
- **R** – folder containing R function files (.R)

Optional files and folders can be included:

- **ChangeLog** – file containing list of changes across versions
- **data** – folder containing example data (.Rda)
- **src** – folder containing source code (.c or .f90)

Note: there are several others that could be included (e.g., vignette)

Converting Code into a Package

To turn a collection of R files into an R package, you need to...

- Write the documentation files (.Rd)
- Organize the files into appropriate folders
- Create DESCRIPTION and NAMESPACE

You could consider using an “automated” approach for these steps.

- `roxygen2` automates aspects of package building
- `testthat` automates aspects of package testing

Learn from example by looking at good R packages (`car`, `MASS`, `mgcv`)

- Download the source code and explore the organization
- Helps understand DESCRIPTION and NAMESPACE files

Let's Look at Some Examples

From a structure perspective, my simplest R package is `Dykstra`

- One primary function and one S3 method (print)
- Package exists to implement a single algorithm

From a structure perspective, my most complex R package is `grpnet`

- 30+ R function files, 19 documentation files, 14 Fortran files
- Design was inspired by the popular R package `glmnet`

For a nice middle group, you can see my R packages `npreg` and `npctest`

- Both packages have several R functions and S3 methods
- But neither package uses external (C/Fortran) code

Table of Contents

1. The Art of Coding
2. Good and Bad Practices
3. Publishing an R Package
4. Higher Level Languages

Calling C and Fortran from R

R has the ability to call C or Fortran code (included in `src` subfolder)

- `.C` function for calling C/C++ code
- `.Fortran` function for calling Fortran code

Many packages in R use C or Fortran under the hood for heavy lifting.

- C is more popular (R is based on S, which is based on C)
- I prefer Fortran due to its intrinsic matrix functionalities

Whether you need to call C/Fortran will depend on your problem.

- R is rather fast if you can vectorize your code
- R is rather slow if you need nested iterations

What is the Benefit of Learning C/Fortran?

First, it can make your own code more efficient.

- When lots of iterative work is required, it will beat R
- It will make your code competitive with state-of-the-art

Second, learning C or Fortran is good for learning coding basics.

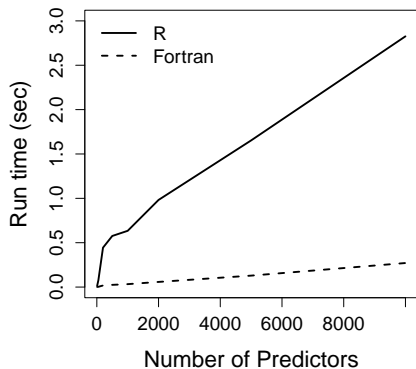
- Many principles of good coding apply to all languages
- Being multilingual is good for understanding others' code

Note: LLMs are very effective tools for understanding legacy code.

- Want to know what some Fortran code is doing? Ask a LLM
- Want to see that code translated to R or Python? Ask a LLM

Speed Comparison of R versus Fortran

N = 100



N = 1000

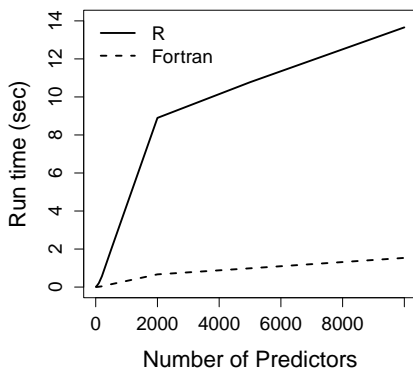


Figure 2: Comparison of fitting time for `grpnet()` function with `proglang = "R"` and `proglang = "Fortran"`.

Take Home Points

For most methodological papers, you could likely develop a package.

- Publishing script files as supplementary materials is fine
- Developing code into a package is harder—but better for science

If you are going to program something, then...

- Take the time to develop high quality code (and documentation!)
- Publish and maintain your code in an open-source repository

Quality open-source software and coding are essential for science.

- Many mistakes in scientific papers are due to coding errors
- “The code is fine as long as it works” is the wrong attitude