Tidy Tuesday: Creating a lab R package

Session 1: 2023-06-27

Ina Bornkessel-Schlesewsky

Table of contents

Basic principles of software engineering

Why software engineering?

- Connolly et al. (2023) argue that good data science requires quality software engineering
- many researchers develop their own software for their research
  - e.g. 56% of researchers in the UK according to a 2014 survey (Switters and Osimo 2019)
- many academics don’t have any experience in this area, making it more difficult for them to produce software that adheres to the 3Rs of software engineering: **readability**, **resilience** and **reusability**

The 3Rs of software engineering

- Code should be
  - **readable**: it is understandable by others, e.g. through use of comments and naming conventions
  - **resilient**: it fails rarely or, when it does, fails gracefully; requires testing for common errors (unit tests)
  - **reusable**: others can use the code as is without extensive rewriting

Importance of project scope

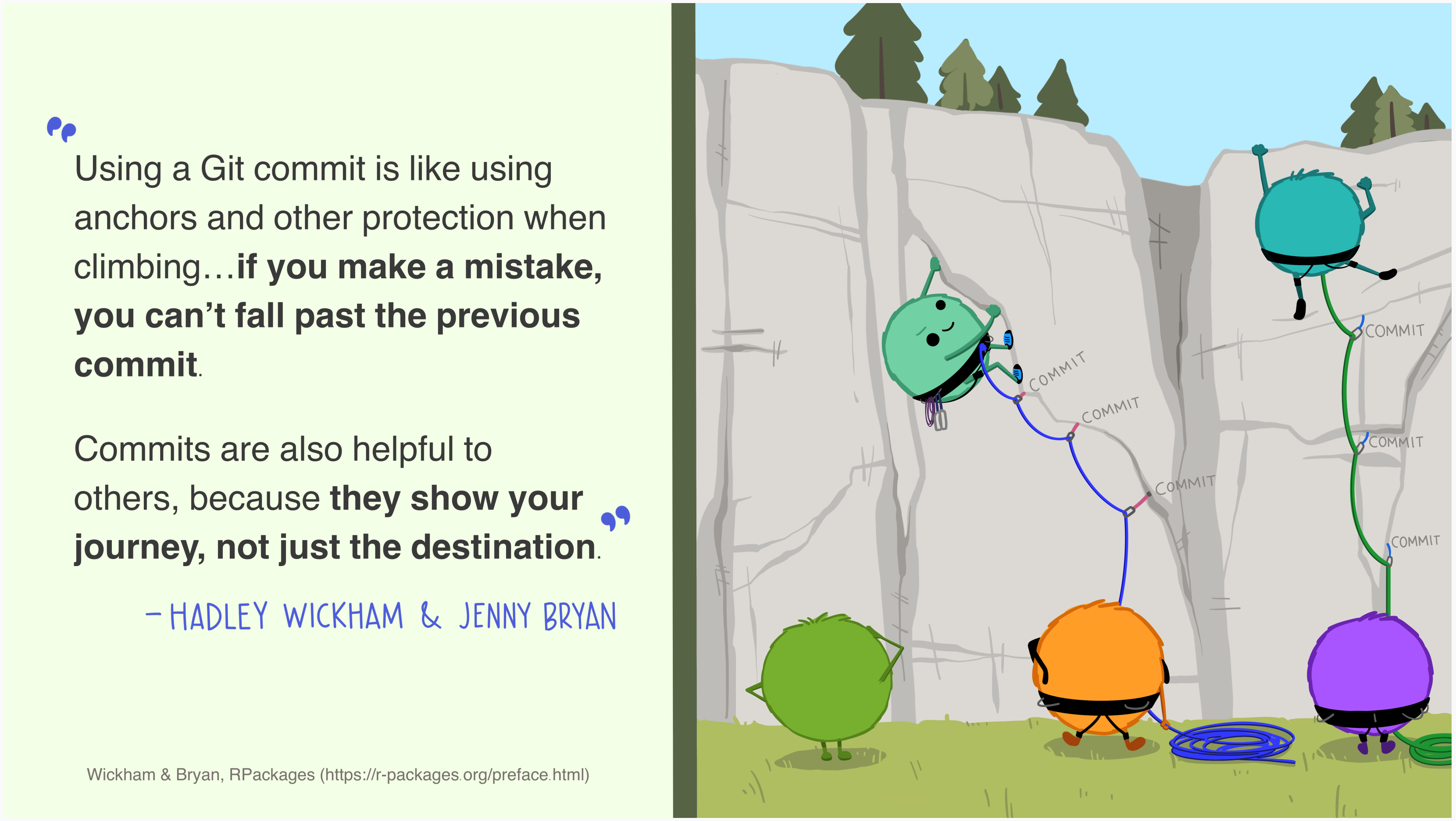
- researchers should have an awareness of good software engineering practices
- the level of software engineering rigour depends on the project scope:
    - **solo** (single researcher creates and uses the project)
    - **lab** (developers and users know each other and are in close contact)
    - **community** (developers have limited knowledge of the users)

Note: the potential of transitioning to a larger scope at a later point in the project is also an important consideration.

Engineering practices most relevant to data science

Version control

- tracks changes to artefacts in shared collections of files ("repositories")



"Using a Git commit is like using anchors and other protection when climbing…**if you make a mistake, you can't fall past the previous commit**.

Commits are also helpful to others, because **they show your journey, not just the destination**."

– HADLEY WICKHAM & JENNY BRYAN

Wickham & Bryan, RPackages (https://r-packages.org/preface.html)

Artwork by @allison_horst

Version control

- commonly used to undo a change that introduces an error, coordinate changes among multiple developers; branching capabilities to make changes in parallel and for managing experimental code
  - **solo project**: ensure code is not lost; ability to revert; enable experimentation and exploration of new ideas
  - **lab project**: resolve change conflicts
  - **community project**: more formal coordination to manage releases, develop new features, integrate code from other groups and handle urgent bug fixes

Computer programming or coding

- the process of writing detailed instructions so that a computer can perform a desired task
- encompasses a wide range of decisions such as:

a. choice of variables, functions and modules

b. documentation practices

c. the use of language features (e.g. for loop vs. map function)

d. choice of data structures (e.g. list vs array vs dict)

e. software licences

Computer programming or coding

- **solo project**: readability is improved through notes about decisions made (e.g. GitHub README or within a notebook) and use of consistent naming conventions to help understand earlier code
- **lab project**: readability and resilience are enhanced through agreement on common data structures and coding styles
- **community project**: code reviews

Lab R package project

- create an R package containing helpful functions for use by the lab
- practice several aspects of good software engineering:
  - readability through good documentation, use of a consistent style
  - version control with Git and GitHub
  - resilient code: how to test whether the code performs as expected and ensure that it “fails gracefully”
  - reusable code: turn commonly used code into functions that are reusable by both your future self and others in the lab

Note: we can potentially extend this to creating a lab Python package at a later date.

Roadmap for the project

Solo-level project

- create (a) function(s) that are useful in your data analysis workflow
- document your code appropriately
- use version control
- test your code

Lab-level project

- wrap the functions in a package for better reusability
- collaborate on writing code using version control and GitHub
- standardise code by agreeing on data structures and code style (cf. the Tidyverse style guide)

Creating functions in R

Anatomy of a function

```r
function_name <- function(arg1, arg2){
  function_body
}
```

- use the `function()` function to define a function
- define any arguments required in the round brackets
- the function body is is enclosed in curly brackets and can contain as many lines of code as needed; it should make reference to the arguments

A simple example: read csv and clean names

```r
squirrels <- read_csv(here("data","2018_squirrel_census.csv"))
head(squirrels)
```

```
# A tibble: 6 × 36
      X     Y `Unique Squirrel ID` Hectare Shift     Date Hectare Squirrel Num…¹
  <dbl> <dbl> <chr>                <chr>   <chr>    <dbl>                  <dbl>
1 -74.0  40.8 37F-PM-1014-03       37F     PM    10142018                      3
2 -74.0  40.8 37E-PM-1006-03       37E     PM    10062018                      3
3 -74.0  40.8 2E-AM-1010-03        02E     AM    10102018                      3
4 -74.0  40.8 5D-PM-1018-05        05D     PM    10182018                      5
5 -74.0  40.8 39B-AM-1018-01       39B     AM    10182018                      1
6 -74.0  40.8 33H-AM-1019-02       33H     AM    10192018                      2
# ℹ abbreviated name: ¹`Hectare Squirrel Number`
# ℹ 29 more variables: Age <chr>, `Primary Fur Color` <chr>,
#   `Highlight Fur Color` <chr>,
#   `Combination of Primary and Highlight Color` <chr>, `Color notes` <chr>,
#   Location <chr>, `Above Ground Sighter Measurement` <chr>,
#   `Specific Location` <chr>, Running <lgl>, Chasing <lgl>, Climbing <lgl>,
#   Eating <lgl>, Foraging <lgl>, `Other Activities` <chr>, Kuks <lgl>, …
```

A simple example: read csv and clean names

```r
squirrels <- read_csv(here("data","2018_squirrel_census.csv")) |>
  clean_names()
head(squirrels)
```

```
# A tibble: 6 × 36
      x     y unique_squirrel_id hectare shift     date hectare_squirrel_number
  <dbl> <dbl> <chr>              <chr>   <chr>    <dbl>                   <dbl>
1 -74.0  40.8 37F-PM-1014-03     37F     PM    10142018                       3
2 -74.0  40.8 37E-PM-1006-03     37E     PM    10062018                       3
3 -74.0  40.8 2E-AM-1010-03      02E     AM    10102018                       3
4 -74.0  40.8 5D-PM-1018-05      05D     PM    10182018                       5
5 -74.0  40.8 39B-AM-1018-01     39B     AM    10182018                       1
6 -74.0  40.8 33H-AM-1019-02     33H     AM    10192018                       2
# ℹ 29 more variables: age <chr>, primary_fur_color <chr>,
#   highlight_fur_color <chr>,
#   combination_of_primary_and_highlight_color <chr>, color_notes <chr>,
#   location <chr>, above_ground_sighter_measurement <chr>,
#   specific_location <chr>, running <lgl>, chasing <lgl>, climbing <lgl>,
#   eating <lgl>, foraging <lgl>, other_activities <chr>, kuks <lgl>,
#   quaas <lgl>, moans <lgl>, tail_flags <lgl>, tail_twitches <lgl>, …
```

Wrap into a function

```r
read_and_tidy <- function(filename){
  read_csv(filename) |>
    clean_names()
}

squirrels <- read_and_tidy(here("data","2018_squirrel_census.csv"))
head(squirrels)
```

```
# A tibble: 6 × 36
      x     y unique_squirrel_id hectare shift     date hectare_squirrel_number
  <dbl> <dbl> <chr>              <chr>   <chr>    <dbl>                   <dbl>
1 -74.0  40.8 37F-PM-1014-03     37F     PM    10142018                       3
2 -74.0  40.8 37E-PM-1006-03     37E     PM    10062018                       3
3 -74.0  40.8 2E-AM-1010-03      02E     AM    10102018                       3
4 -74.0  40.8 5D-PM-1018-05      05D     PM    10182018                       5
5 -74.0  40.8 39B-AM-1018-01     39B     AM    10182018                       1
6 -74.0  40.8 33H-AM-1019-02     33H     AM    10192018                       2
# ℹ 29 more variables: age <chr>, primary_fur_color <chr>,
#   highlight_fur_color <chr>,
#   combination_of_primary_and_highlight_color <chr>, color_notes <chr>,
#   location <chr>, above_ground_sighter_measurement <chr>,
#   specific_location <chr>, running <lgl>, chasing <lgl>, climbing <lgl>,
#   eating <lgl>, foraging <lgl>, other_activities <chr>, kuks <lgl>,
#   quaas <lgl>, moans <lgl>, tail_flags <lgl>, tail_twitches <lgl>, …
```

A more realistic example: objective

- read in 45 EEG data files, perform some preprocessing (e.g. restrict to ROI and isolate prestim activity) and combine into a data frame

```r
eeg_files <- Sys.glob(here("oberon_erp_data", "*.csv"))
head(eeg_files, n=3)
```

```
[1] "/Users/ina/Library/Mobile
Documents/com~apple~CloudDocs/Documents/00.01_projects/00.01.02_research_lab/2023_TT_lab_package/oberon_erp_data/01_erps_critical_epochs.csv"
[2] "/Users/ina/Library/Mobile
Documents/com~apple~CloudDocs/Documents/00.01_projects/00.01.02_research_lab/2023_TT_lab_package/oberon_erp_data/02_erps_critical_epochs.csv"
[3] "/Users/ina/Library/Mobile
Documents/com~apple~CloudDocs/Documents/00.01_projects/00.01.02_research_lab/2023_TT_lab_package/oberon_erp_data/03_erps_critical_epochs.csv"
```

```r
read_csv(eeg_files[1]) |>
  glimpse()
```

```
Rows: 108,570
Columns: 9
$ ...1      <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17…
$ epoch     <dbl> 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, …
$ condition <chr> "DET/C/1/B/65", "DET/C/1/B/65", "DET/C/1/B/65", "DET/C/1/B/6…
$ win       <chr> "-200..0", "-200..0", "-200..0", "-200..0", "-200..0", "-200…
$ wname     <chr> "prestim", "prestim", "prestim", "prestim", "prestim", "pres…
$ subj      <chr> "01", "01", "01", "01", "01", "01", "01", "01", "01", "01", …
$ channel   <chr> "F7", "F5", "F3", "F1", "Fz", "F2", "F4", "F6", "F8", "FT7",…
$ mean      <dbl> -1.7257816, -0.1292891, -0.6471173, -4.8748872, -8.1013147, …
$ sem       <dbl> 0.4031690, 0.3947023, 0.4854820, 0.5933222, 0.5038471, 0.636…
```

A more realistic example: function(s)

```r
# Define region of interest for N400 analysis
electrodes <- c("C3","C1","Cz","C2","C4",
                "P3","P1","Pz","P2","P4",
                "CP3","CP1","CPz","CP2","CP4")

isolate_prestim <- function(df){
  df |>
    filter(wname == "prestim") |>
    select(!c(win,wname))
}

read_and_preprocess <- function(filename){

  eeg_full <- read_csv(filename) |>
    clean_names() |>
    select(-x1) |>
    mutate(subj = as.character(subj)) |>
    filter(wname %in% c("n400","prestim"),
         channel %in% electrodes)

  eeg_full |>
    filter(wname == "n400") |>
    left_join(
      isolate_prestim(eeg_full),
      by = c("epoch","condition","subj","channel"),
      suffix = c("_n400", "_prestim")) |>
    separate(condition, into = c("cat","canon","position",
                                 "speaker","passage"))
}
```

A more realistic example: application

```r
all_wins <- eeg_files |>
  map_df(read_and_preprocess)

glimpse(all_wins)
```

```
Rows: 449,535
Columns: 14
$ epoch        <dbl> 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 1…
$ cat          <chr> "DET", "DET", "DET", "DET", "DET", "DET", "DET", "DET", "…
$ canon        <chr> "C", "C", "C", "C", "C", "C", "C", "C", "C", "C", "C", "C…
$ position     <chr> "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1…
$ speaker      <chr> "B", "B", "B", "B", "B", "B", "B", "B", "B", "B", "B", "B…
$ passage      <chr> "65", "65", "65", "65", "65", "65", "65", "65", "65", "65…
$ win          <chr> "300..500", "300..500", "300..500", "300..500", "300..500…
$ wname        <chr> "n400", "n400", "n400", "n400", "n400", "n400", "n400", "…
$ subj         <chr> "01", "01", "01", "01", "01", "01", "01", "01", "01", "01…
$ channel      <chr> "C3", "C1", "Cz", "C2", "C4", "CP3", "CP1", "CPz", "CP2",…
$ mean_n400    <dbl> -2.1563104, -1.6411366, -3.0616647, -4.7654576, -4.499718…
$ sem_n400     <dbl> 0.4382589, 0.4016568, 0.3726864, 0.3299565, 0.5261620, 0.…
$ mean_prestim <dbl> -3.34017926, -5.08264843, -3.78956231, -1.85922794, 0.547…
$ sem_prestim  <dbl> 0.3282207, 0.2806286, 0.3385096, 0.3175025, 0.5741826, 0.…
```

Issues 1: documentation

```r
# function to isolate prestimulus activity
# from EEG window averages
# df: a data frame
isolate_prestim <- function(df){
  df |>
    # only retain rows pertaining to the prestim window
    filter(wname == "prestim") |>
    # drop columns referencing the window / window name
    select(!c(win,wname))
}

# function to read and preprocess EEG window average data
read_and_preprocess <- function(filename){

  eeg_full <- read_csv(filename) |>
    # clean name of unnamed column that is labelled as "1"
    clean_names() |>
    # remove this column
    select(-x1) |>
    # ensure that subject number is a string
    mutate(subj = as.character(subj)) |>
    # only keep N400 and prestim time windows
    # as well as electrodes of interest
    filter(wname %in% c("n400","prestim"),
         channel %in% electrodes)

  eeg_full |>
    # isolate N400 windows
    filter(wname == "n400") |>
    # join corresponding prestim data
    left_join(
      isolate_prestim(eeg_full),
      by = c("epoch","condition","subj","channel"),
      suffix = c("_n400", "_prestim")) |>
    # separate complex condition label into multiple columns
    separate(condition, into = c("cat","canon","position",
                                 "speaker","passage"))
```

Issues 1: documentation

- the comments added to the code on the previous slide help to document the function
- there are, however, more principled ways to document functions, which also allow for the documentation to show up as “help” for the function (e.g. through `?read_and_preprocess`)
- we will look at these in future sessions

Issues 2: generality of use

- the current version of the function is quite specific to one particular experiment
- for reusability, it would need to be made more general, allowing it to apply to more use cases
- will look at how to do this in a future session

Your turn!

Ahead of our next session

1. Find some code that you reuse regularly (or might want to reuse in future)
2. Wrap it into a function (and check that it works)
3. Document your function using comments
4. Consider whether / how others in the lab might use your function
5. Bring your code (and thoughts re. 4) to our next session!

References

Bornkessel-Schlesewsky, Ina, Isabella Sharrad, Caitlin A. Howlett, Phillip M. Alday, Andrew W. Corcoran, Valeria Bellan, Erica Wilkinson, et al. 2022. “Rapid Adaptation of Predictive Models During Language Comprehension: Aperiodic EEG Slope, Individual Alpha Frequency and Idea Density Modulate Individual Differences in Real-Time Model Updating.” *Frontiers in Psychology* 13 (August): 817516. https://doi.org/10.3389/fpsyg.2022.817516.

Connolly, Andrew, Joseph Hellerstein, Naomi Alterman, David Beck, Rob Fatland, Ed Lazowska, Vani Mandava, and Sarah Stone. 2023. “Software Engineering Practices in Academia: Promoting the 3Rs, Resilience, and Reuse.” *Harvard Data Science Review* 5 (2). https://doi.org/10.1162/99608f92.018bf012.

Switters, Jon, and David Osimo. 2019. *Recognising the Importance of Software in Research: Research Software Engineers (RSEs), a UK Example.* Luxembourg: Publications Office of the European Union.