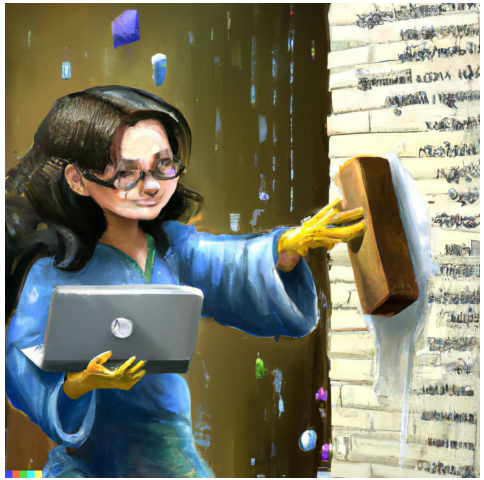
Clean Code:
Why and how to use it

WWZ, 05.02.2024

Dr. Anthea Alberto

Research and Infrastructure Support (RISE)



What exactly is *clean code*?

"I like my code to be **elegant and efficient.** The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well**."

- Bjarne Stroustrup, inventor of C++ (emphasis mine)

Cited in Martin, R. (2015): Clean Code. Upper Saddle River, NJ: Prentice Hall.

What exactly is *clean code*?

"Clean code is simple and direct. Clean code reads like **well-written prose.** Clean code never obscures the designer's intent but rather is full of crisp abstractions and straightforward lines of control."

-Grady Booch, author of Object Oriented Analysis and Design with Applications *(emphasis mine)*

Cited in Martin, R. (2015): Clean Code. Upper Saddle River, NJ: Prentice Hall.

What exactly is *clean code*?

There is no one clear-cut definition.

The concept is primarily tied to software engineering and refers to code that is *intuitively understandable.*

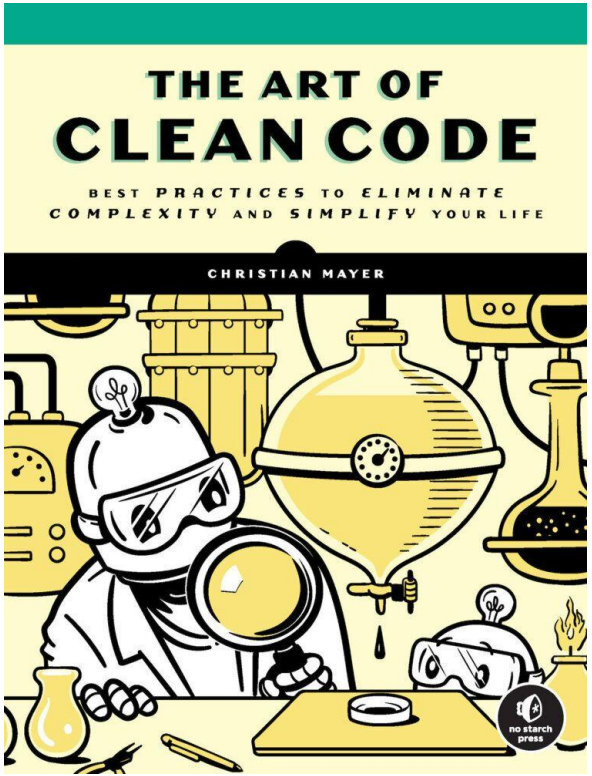
Some core principles are **DRY** (don’t repeat yourself), **YAGNI** (you ain’t gonna need it) and **KISS** (keep it simple, stupid).

The ultimate goal of clean code is to work more efficiently by reducing the time spent trying to understand what you or someone else have done before.

Why you should use clean code

- It saves time & increases efficiency
  - Clean code will reduce the effort of trying to understand a script when coming back to it after a while
- It makes collaboration easier
  - There’s no need for project partners to explain their code when you can just run & understand it yourself
- Having well organized scripts will make it easier to go back for reference
  - You’ll likely reuse your code a lot during your PhD (and later)
- It is important for **reproducibility**
  - Being mindful of your code while working on a project will save time later when putting together reproduction materials

Principles of clean code



Inspiration & recommended reading:

Mayer, C. (2022): The Art of Clean Code. San Francisco: No Starch Press, Inc.

Code for people, not machines

- Getting your code to work is one thing, but it should be *understandable by humans* too!

- “Always assume that others will read your source code.” (Mayer, 2022, p. 56)

- This is a good rule to stick to even if you’re certain you’re the only one who will ever look at your script again

Use the right names

- Descriptive variable names help you make sense of your (and other people's) code later

- Names should be meaningful, unambiguous, pronounceable and easy to spell

- It's tempting to use placeholder names like *x, df* or *dat,* but these should be avoided in scripts you want to reuse

Not ideal

```
82  x <- spd_names[,c(1,4)]
83  x$surname <- trimws(x$surname, which = "both")
84  y <- spd_cab[,c(2,6)]
85  y$surname <- trimws(y$surname, which = "both")
86  colnames(x)[1] <- "name"
87  colnames(y)[1] <- "name"
88  z <- rbind(x,y)
89  z <- unique(z)
90  z$surname[9] <- "Bennetor"
```

Adhere to standards and be consistent

- Use capital letters or underscores to distinguish words
- Either use numbers directly or spell them out
- Avoid mixing styles, it will just make your life harder

Don’t mix styles

```
42  list.files()
43  load("german_cabinets.Rda")
44  load("spd_fraktionsvorsitz.Rda")
45  load("spd_fsekretaer.Rda")
46  load("spd_parteivorsitz.Rda")
47  load("spd.minp.Rda")
48
```

Use comments (& avoid unnecessary comments)

- Comments are a good way of explaining bits of code
- They can either be for yourself to understand your code later down the line or for your collaborators
- However, you shouldn’t overcomment and clutter your script that way
  - Exceptions are scripts for teaching purposes
- I personally recommend having some sort of “lab journal” (e.g. a word document) where you report on different things you tried, alterations you made to certain parameters etc.

Don’t repeat yourself (DRY)

- Clue’s in the title: try to avoid repetitions
- Rule of thumb: don’t copy-paste the same bits of code for (sort of) repeated tasks
- Try to use for-loops, functions or apply() to perform repeated tasks
- This avoids clutter and makes your code easier to amend
- However: “You can either spend ten minutes doing something by hand or waste two hours failing to automate it.”
    - Be economical with your time too, not just your code

Sometimes, copy-paste cannot be avoided (although I am sure there are ways to streamline this more)

```
18  spd1$text <- gsub("\\HERAUSGEBER.*","",spd1$text)
19  spd1$text <- gsub("[\r\n]", " ", spd1$text)
20  spd1$text <- gsub("  ", " ", spd1$text)
21  spd1$text <- gsub("- ", "", spd1$text)
22
23  full$text <- gsub("\nSeitenanfang.*","",full$text)
24  full$text <- gsub("[\r\n]", " ", full$text)
25  full$text <- gsub("  ", " ", full$text)
```

Here, a loop would have saved a lot of space

```
91  z$surname[28] <- z$name[28]
92  z$surname[16] <- z$name[16]
93  z$surname[38] <- z$name[38]
94  z$surname[52] <- z$name[52]
95  z$surname[58] <- z$name[58]
```

You ain’t gonna need it

- This one is difficult for me: sometimes I work very hard to make certain models work, only to find that I don’t need them
- It is recommended that unnecessary code be deleted, and for good reason
- However, if you think you might still need code chunks at some point in the future, either
  - put them at the very end of the script, in a dedicated section, or
  - create a script where you collect these bits of code (and use comments to explain what they do and where they came from)

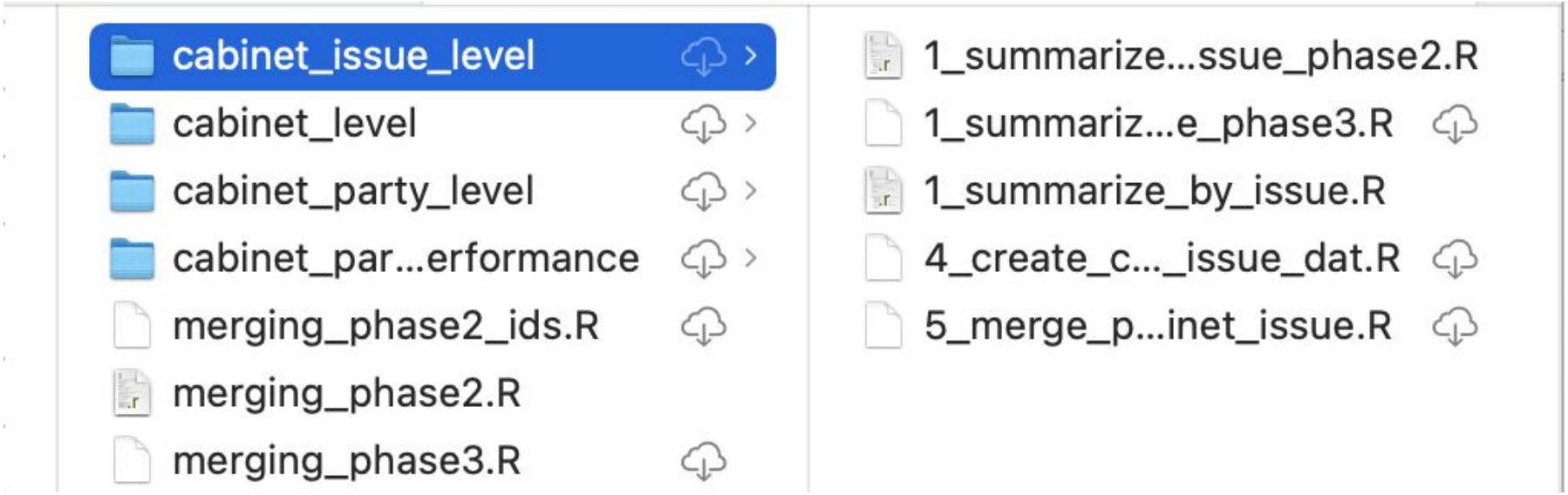
Some practical tips and tricks

- Your scripts should have meaningful names too

- Structuring your scripts: RStudio has some nice built-in features for that (numbering, collapsing chunks etc.)

- A tidy folder structure can help avoid a lot of frustration

Well intentioned, but not ideal :/



Some practical tips and tricks

- Have an additional script open were you can play around with functions without cluttering your main script
  - Or use the console

- Leave empty lines between code chunks

Exercise

In the folder *clean* on GitHub, you can find a subfolder named *cc_exercise.*

In there you'll find two "bad scripts" of mine - try and make them better, keeping in mind the principles you learned about today.

Time: approx. 20 mins

How to practise

- The boy scout rule: *Leave the campground cleaner than you found it.*
  - Try and keep your code clean *over time*
- Take an outsider view of what you're doing right now: is it intuitively understandable?
- Explain what your script is supposed to do: does the explanation match the code?

And lastly…

- People are different and there is no one true way of writing clean code. Don’t try to optimize *everything.*
- Some techniques will work for others but not for you, and that’s fine!
- The important thing is that you’ll find a way of working efficiently and reducing stress factors.

And lastly…

- Try to achieve *flow* when coding*:* being in a very concentrated state of mind and working fluently.
- Don’t interrupt flow in order to clean up your script, you can always come back later.
- Don’t push yourself to write “perfect” code from scratch: it’s always a process.
- “Premature optimization is the root of all evil (or at least most of it) in programming.” - Donald Ervin Knuth

Resources

Style Guide (old) and best practices from tidyverse

PEP 8 - Style Guide for Python Code

Tips, tricks, and philosophies on computational work by Claire Duvallet (mostly on file structure)

Q&A

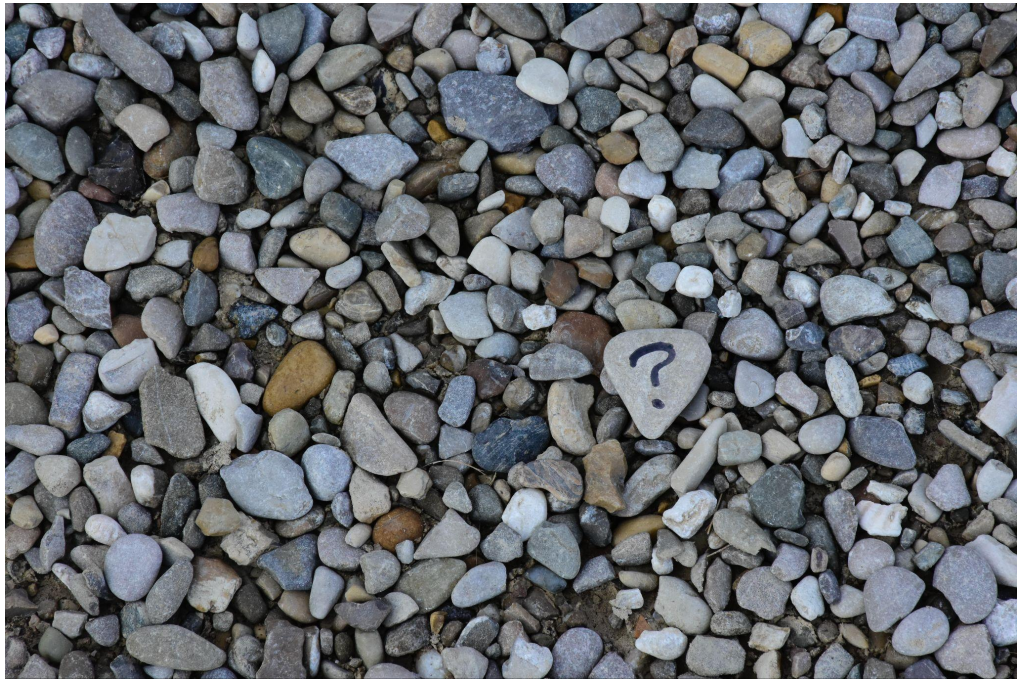


Image: Ana Municio on Unsplash

Thank you for your attention!

Please send any feedback you might have either to rise@unibas.ch or gsbe-wwz@unibas.ch !

Also, if you are interested in specific crash courses, workshops or want advice on your own projects, send an email to rise@unibas.ch or any of the individual team members.