Reproducible Research as Software Design

How principles of software design might make research more reproducible and useful

Christopher Gandrud
Presented at Siteman Cancer Center Biostatistics,
Washington University School of Medicine, St. Louis
2022-01-31

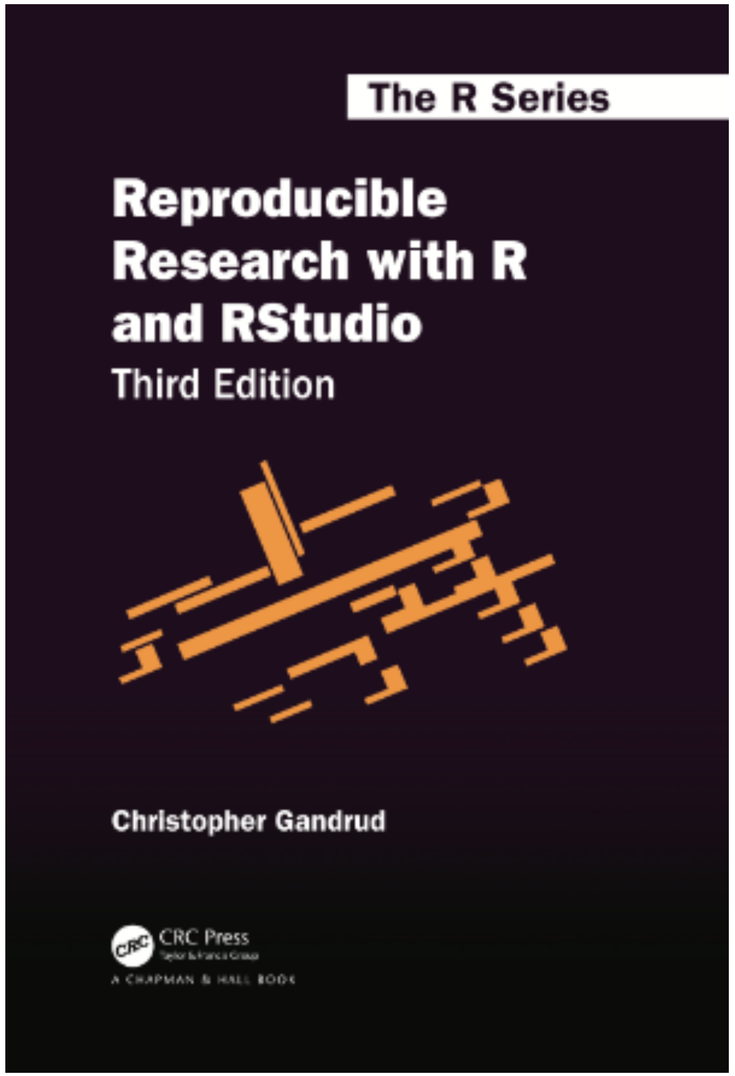
Who am I?

Christopher Gandrud

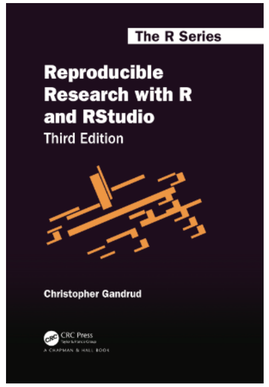
Director (acting) of Economics & Experimentation at Zalando SE (Europes largest online fashion retailer)

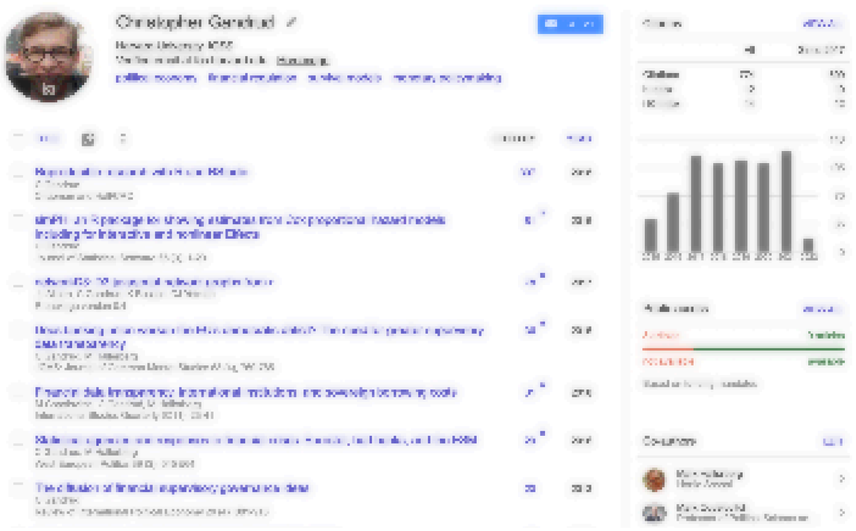
Christopher Gandrud

>10 years thinking about reproducible research

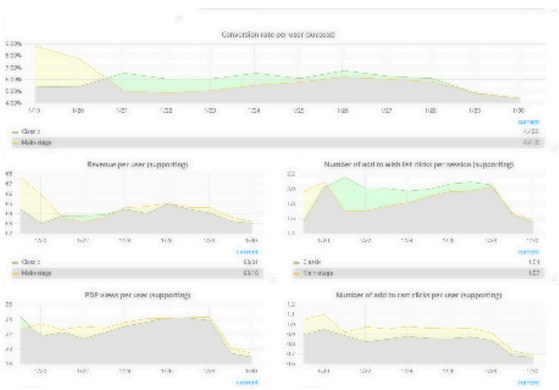


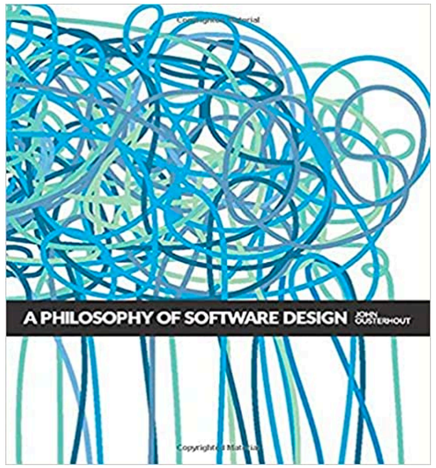
This Talk . . .





Researcher





A Philosophy of Software Design
(Ousterhout 2018)

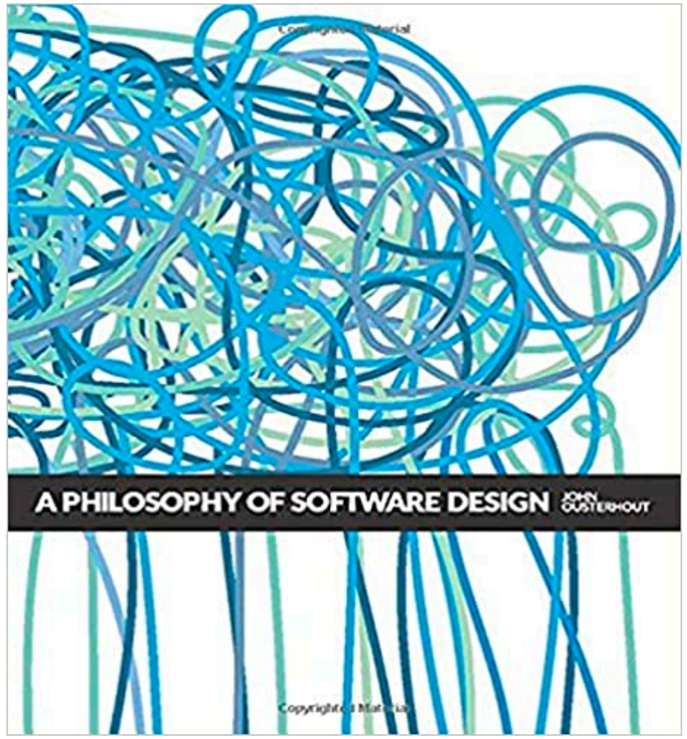
Building Systems to Do Research at Scale

This talk . . . comes from a confluence of experiences

I will draw generously from

A Philosophy of Software Design

(Ousterhout 2018)



The (usual) beginning:

Can’t Reproduce

ABSTRACT

The Influence of Hidden Researcher Decisions in Applied Microeconomics*

Researchers make hundreds of decisions about data collection, preparation, and analysis in their research. We use a many-analysts approach to measure the extent and impact of these decisions. Two published causal empirical results are replicated by seven replicators each. We find large differences in data preparation and analysis decisions, many of which would not likely be reported in a publication. No two replicators reported the same sample size. Statistical significance varied across replications, and for one of the studies the effect's sign varied as well. The standard deviation of estimates across replications was 3-4 times the typical reported standard error.

JEL Classification: C81, C10, B41

Keywords: replication, metascience, research

Corresponding author:
Nick Huntington-Klein
California State University
800 N. State College Blvd.
Fullerton, CA, 92831
USA

E-mail: nhuntington-klein@fullerton.edu

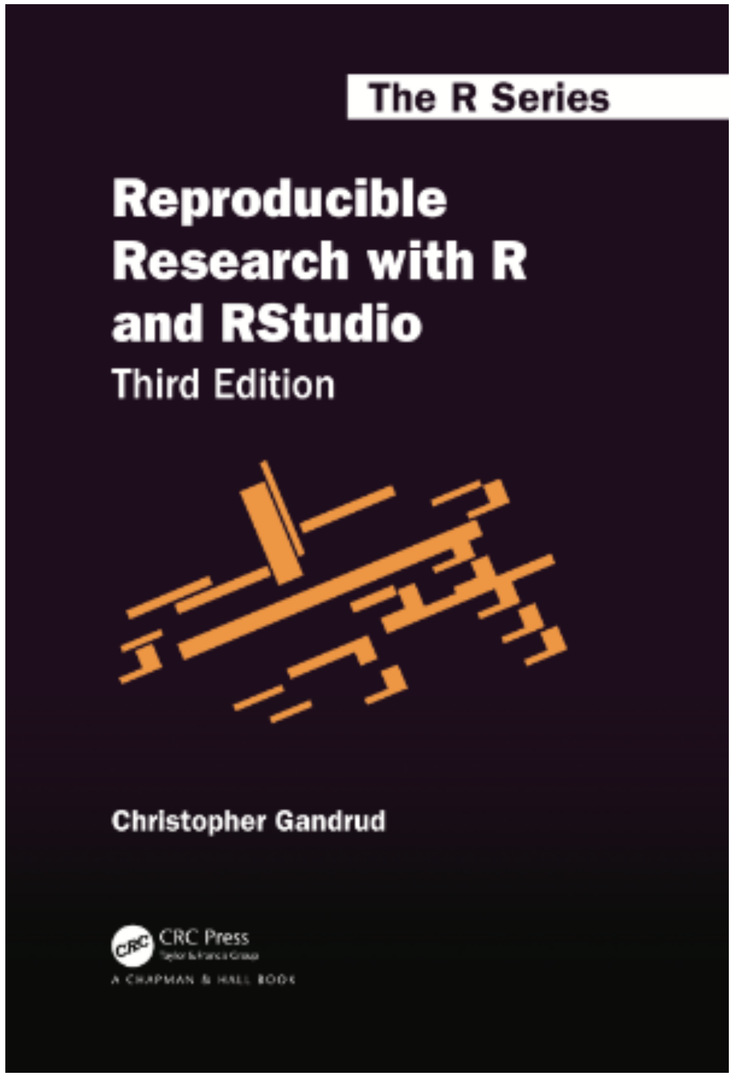
What's going on?

My perception is that the need for reproducible research is well understood, but unsuccessful replications are not rare.

What might be the solution?

#1 Practical tip for reproducible research

Document Everything!



Let’s document more?

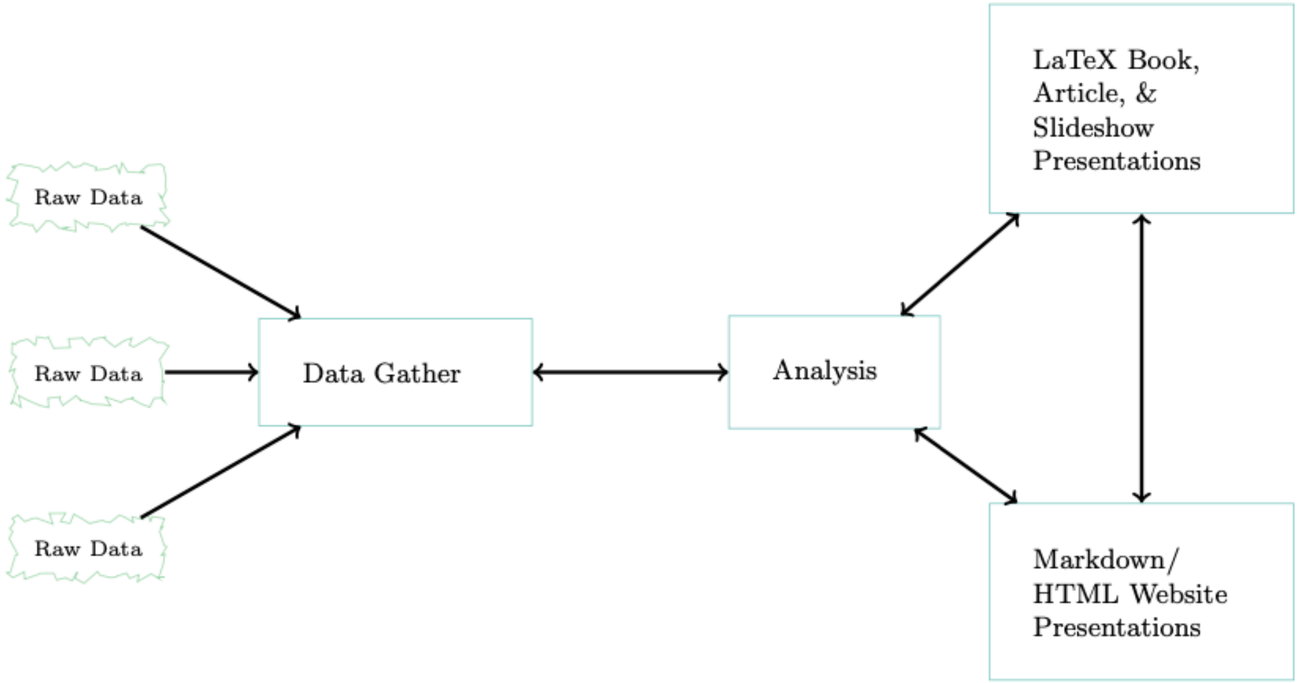
¯_(ツ)_/¯

We might be missing something.

“Quantitative research is to a non-trivial extent software development.”

Reproducible Research for R & RStudio (2020)

This is to a non-trivial extent software development.



Quantitative researchers have adapted many tools from software development

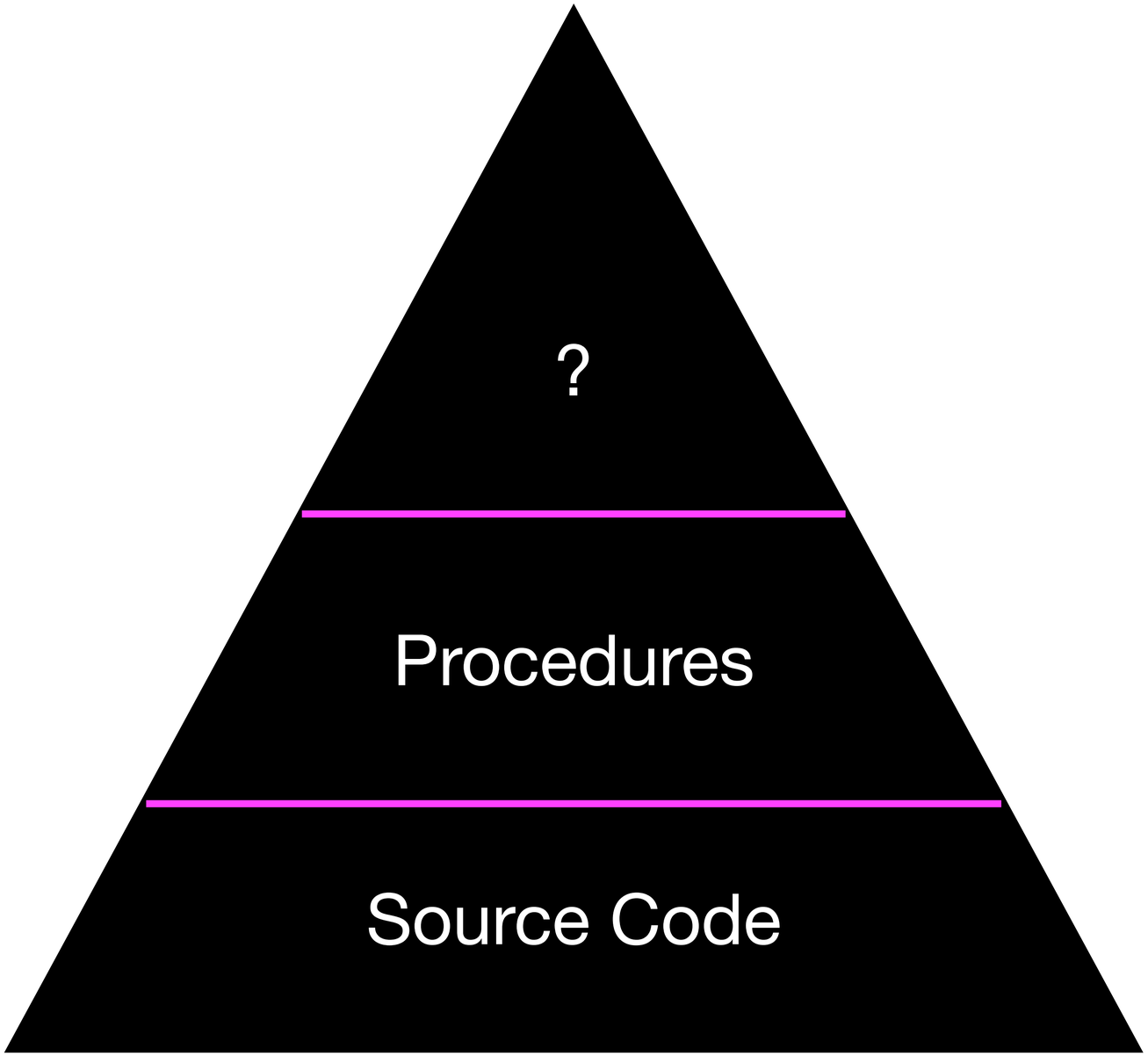
Quantitative researchers have adopted many procedures from software development

git

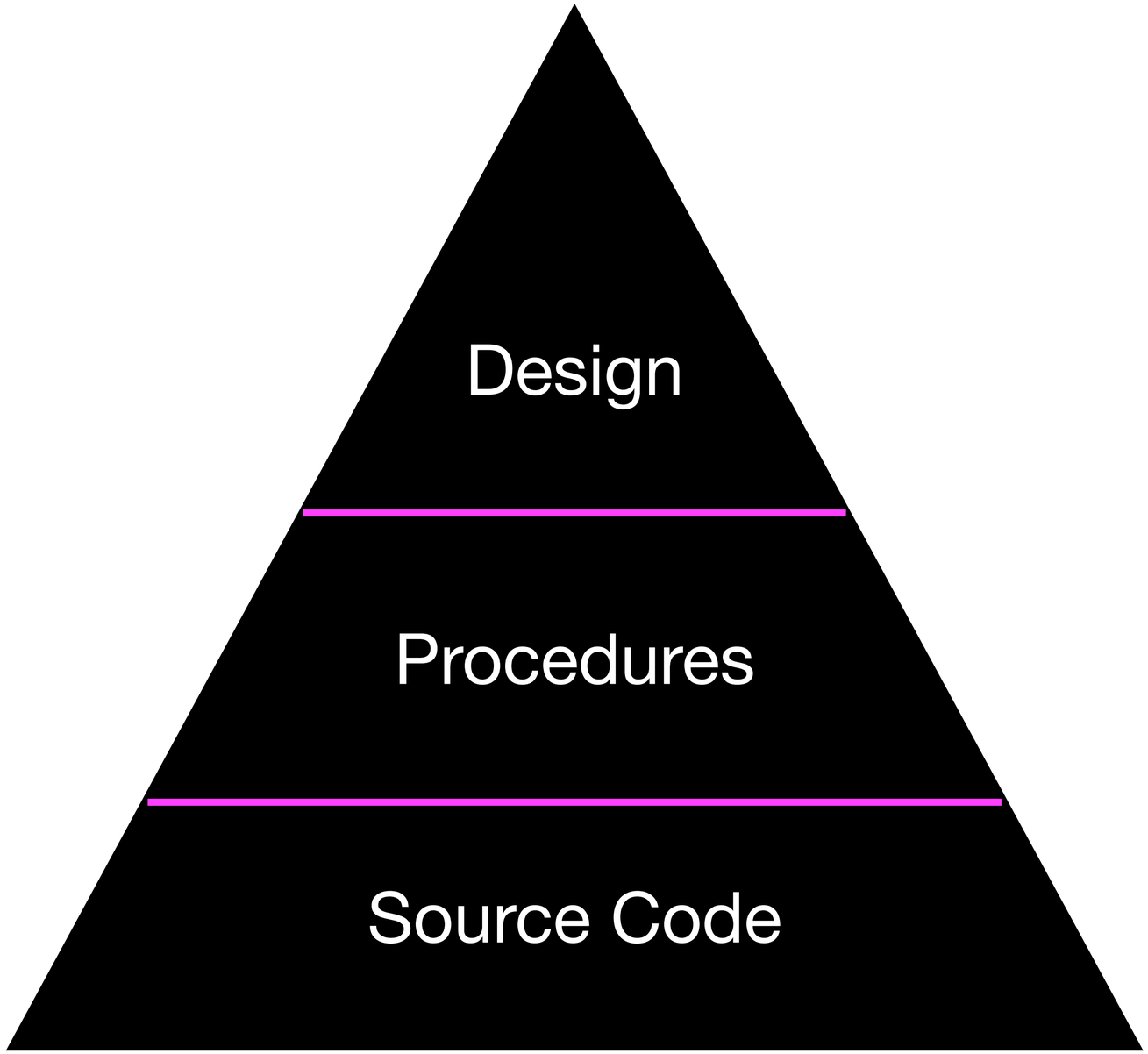
python™ Package Index

Tools and procedures might not be enough.

A hierarchy of software development



A hierarchy of software development



What should researchers know about software design?

“It’s All About Complexity.”

Ousterhout (2018, 1)

“The greatest limitation in writing software is our ability to understand the systems we are creating. As a program evolves and acquires new features, it becomes complicated, with subtle dependencies between components. Over time, complexity accumulates, and it becomes harder and harder for programmers to keep all of the relevant factors in their minds as they modify the system.”

Ousterhout (2018, 1)

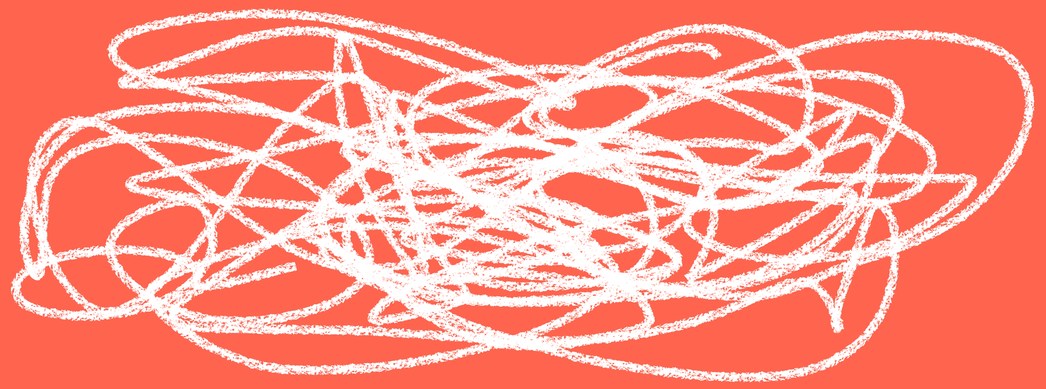
“The greatest limitation in writing software is our ability to understand the systems we are creating. As a program evolves and acquires new features, it becomes complicated, with subtle dependencies between components. Over time, complexity accumulates, and it becomes harder and harder for programmers to keep all of the relevant factors in their minds as they modify the system.”

Ousterhout (2018, 1)

“The greatest limitation in writing software is our ability to understand the systems we are creating. As a program evolves and acquires new features, it becomes complicated, with subtle dependencies between components. Over time, complexity accumulates, and it becomes harder and harder for programmers to keep all of the relevant factors in their minds as they modify the system.”

Ousterhout (2018, 1)

It might *not* be that we don’t have enough documentation, but . . .



. . . the systems we are trying to document are **too complex** to document well.

“The greatest limitation in creating reproducible research is our ability to understand the systems we are creating. As a piece of research evolves and acquires new features, it becomes complicated, with subtle dependencies between components. Over time, complexity accumulates, and it becomes harder and harder for researchers to keep all of the relevant factors in their minds as they modify the research.”

Ousterhout modified for quantitative research

What is software complexity?

“Complexity is anything related to the structure of a software system that makes it hard to **understand** and **modify**.”

Ousterhout (2018, 5)

Many forms of complexity

Hard to understand how a piece of code works.

```
datapackage_install <- function(path,
                                load_file,
                                full_meta = FALSE,
                                ...)
{
  . <- NULL
  # Determine how to load the data package and place it in a tem
  # Is the file from a url?
  if (grepl('^http', path)){
    URL <- path
    temp_path <- digest(URL) %>% paste0('temp_', .)
    download(path = temp_path, url = URL)

    # Unzip if .zip
    if (grepl(pattern = 'zip', x = URL)) {
      temp_path_2 <- paste0('Second', temp_path)

      unzip(temp_path, exdir = temp_path_2)

      zipped_path <- list.files(temp_path_2)
      meta_zipped <- paste0(temp_path_2, '/', zipped_path,
                            '/datapackage.jsonl') %>%
                  fromJSON()
      dkg_zipped_name <- meta_zipped$name

      if (file.exists(paste0 zipped_name)){
        unlink(c(temp_path, temp_path_2), recursive = TR
```

Many forms of complexity

Take a lot of effort to make a small improvement.

Many forms of complexity

Not clear where changes would be made to modify the software.

Many forms of complexity

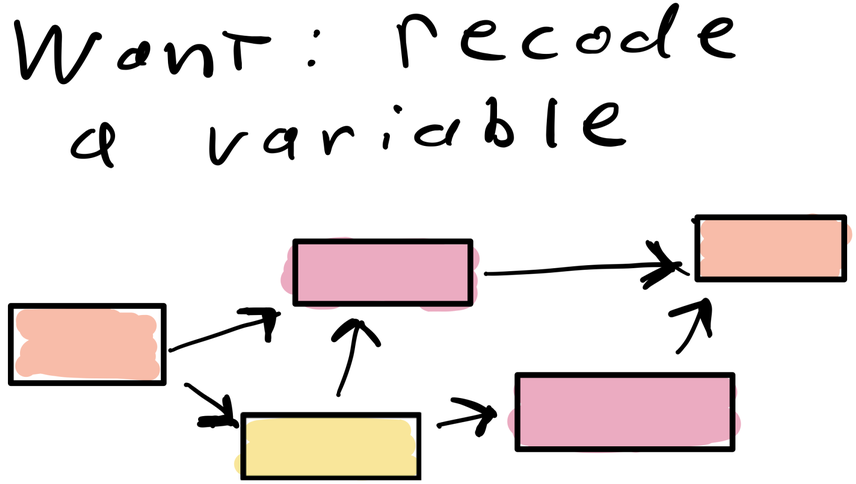
Hard to fix a bug without introducing more bugs.

How to spot complexity

Spotting Complexity

Change test: does a small change require modifications in many different places?

(Likely also introducing bugs)

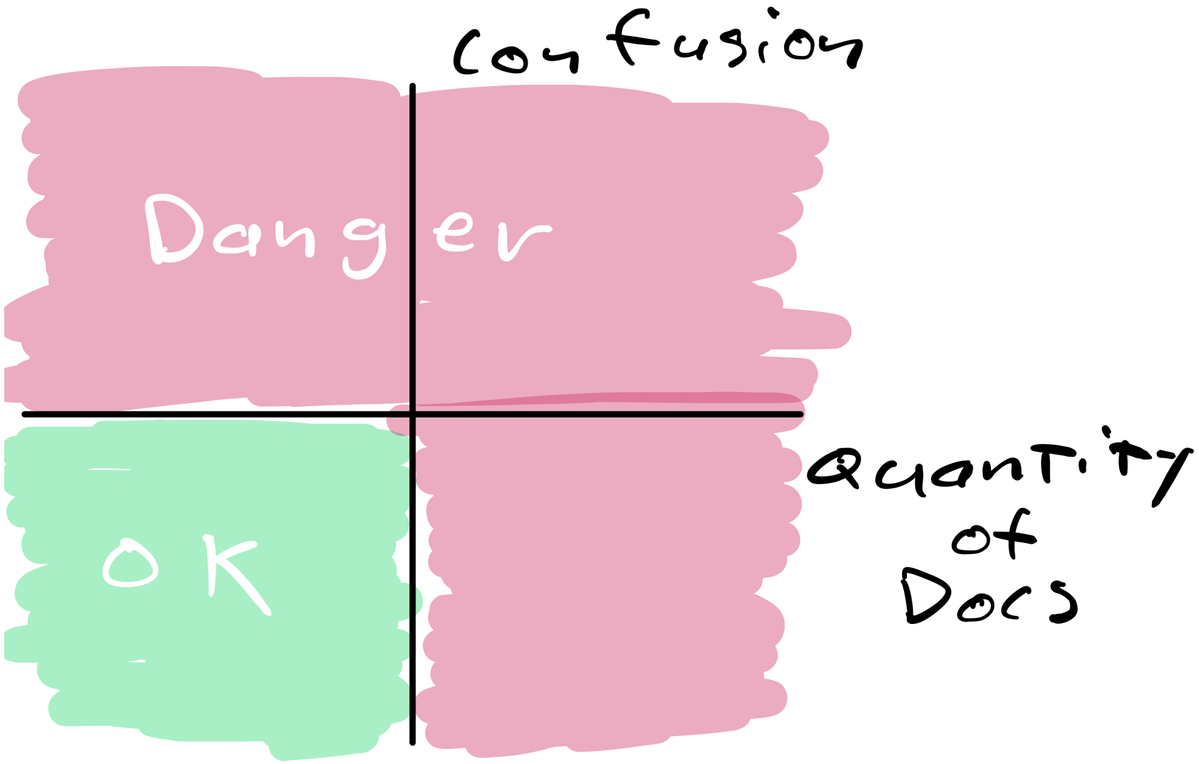


Spotting Complexity

Documentation test: Is a lot of documentation needed to understand/ modify the code?

Even if there is a lot of documentation, is there still a lot of open questions/confusion?

Tip: this test easier for the reader to assess than the docs writer.

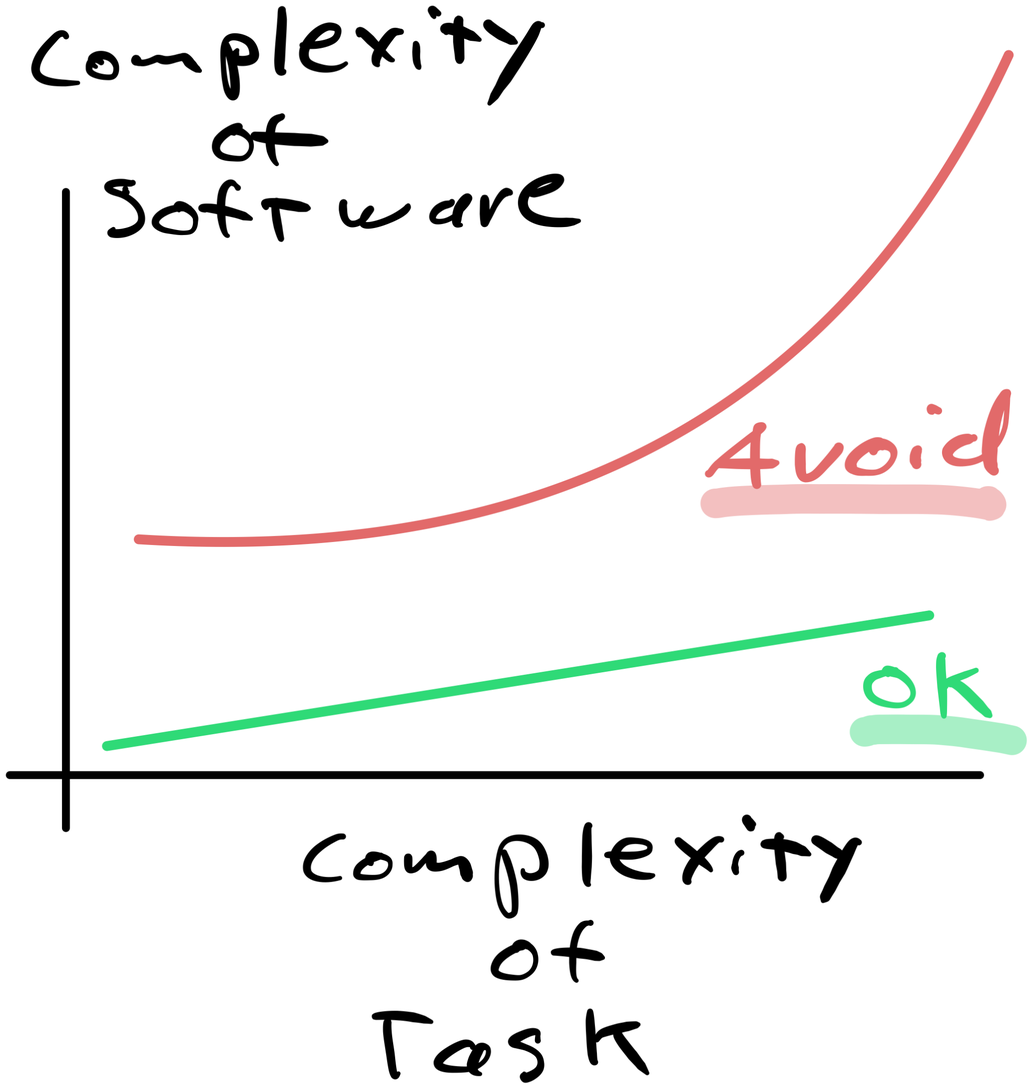


Do our research software artefacts need to be this complex?

“We are researching inherently complex things.

It’s not surprising our research software is complex.”

— Devil’s Advocate


Complexity of Software
Avoid
ok
Complexity of Task

Side note

Writing good software is like good scientific writing generally.

Make key relationships clear to the reader/user.

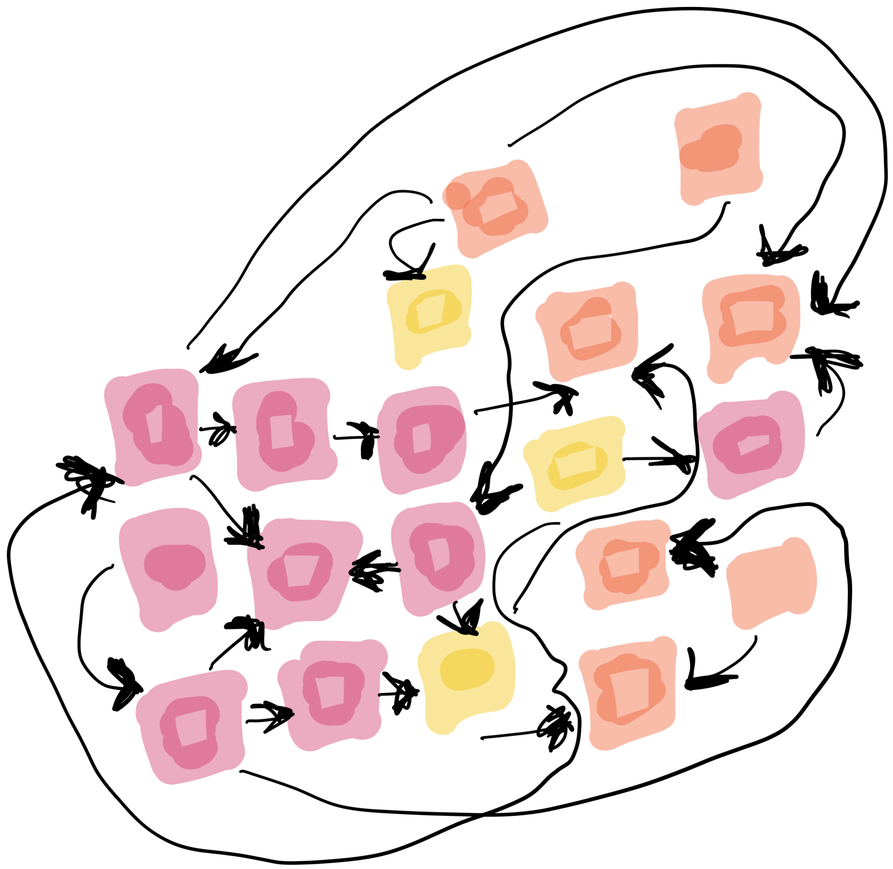
"Hard" skills
writing
words in
code for
User

"Soft" skills
writing
words in
sentences
for Reader

What (proximate) causes complexity?

Proximate Cause of Complexity (1)

Dependencies: many dependencies that are not simple and obvious



Proximate Cause of Complexity (2)

Obscurity: important information is not obvious.

Dependencies

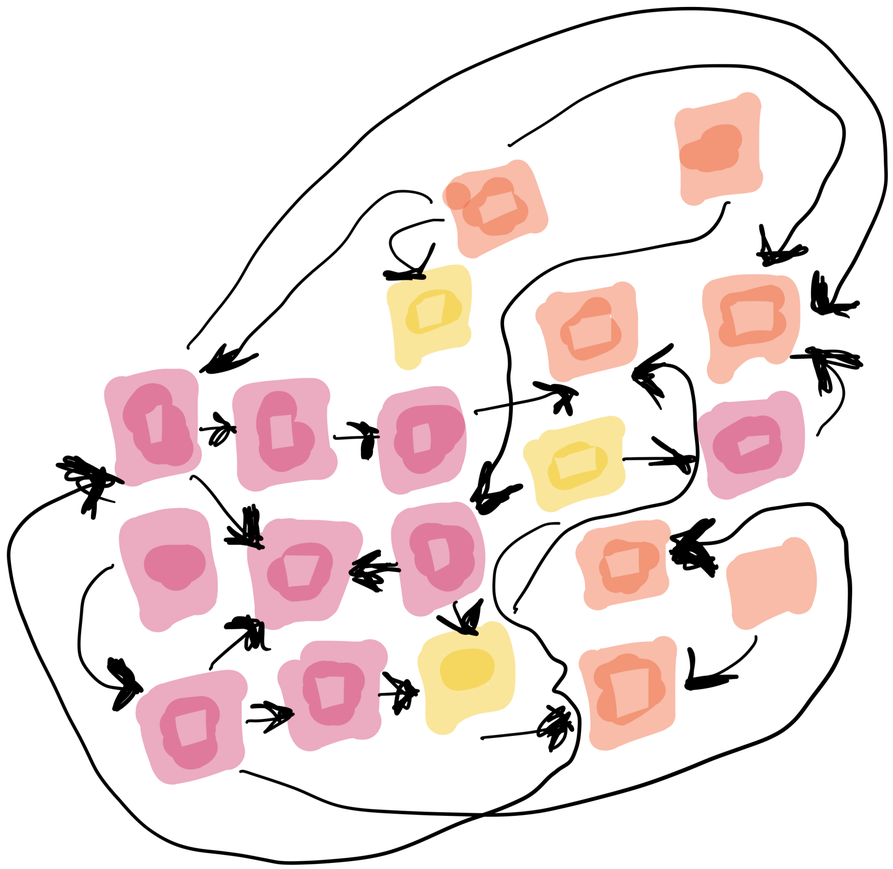
Obscurity

Complexity

Documentation

Complexity Accumulates Over Time

Over the course of a project, lots of little decisions create more dependencies and obscurity



What can we do to design research better, enhance reproducibility?

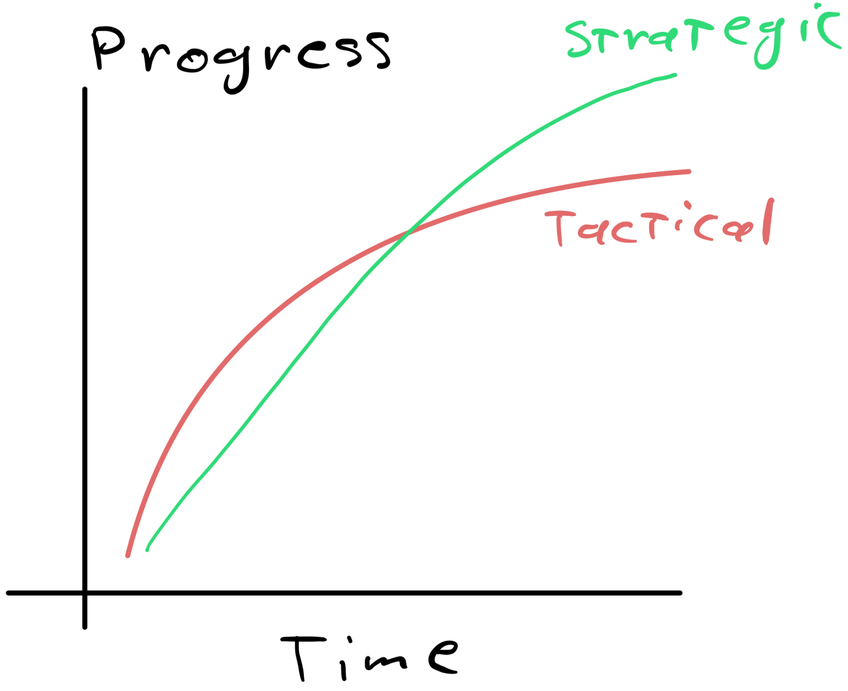
(Some) design principles

Be Strategic

Make better decisions about tactical vs. strategic development

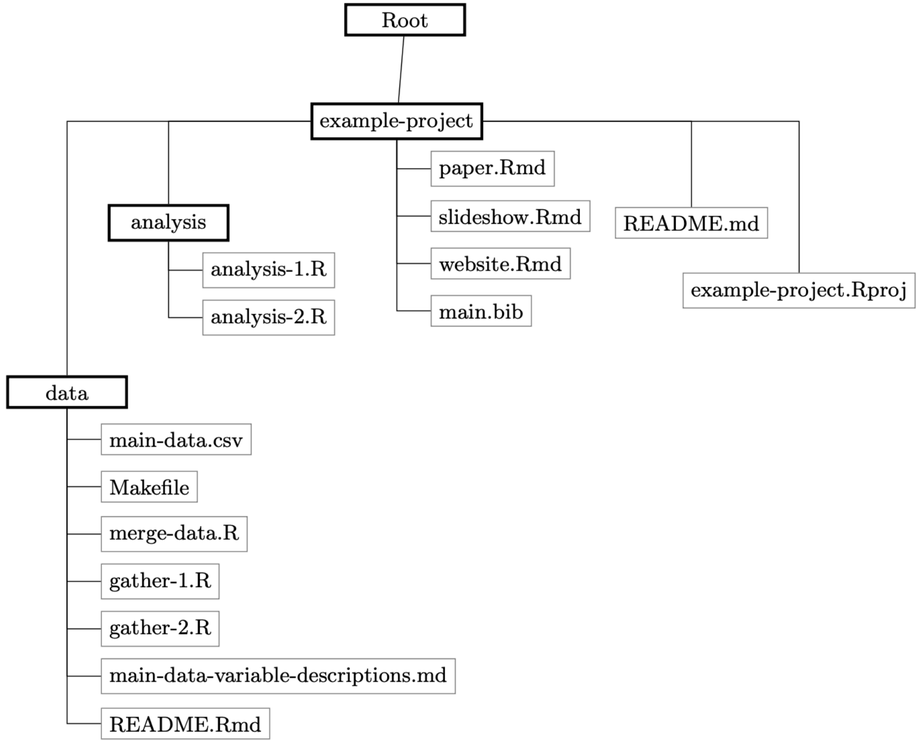
Tactical: start from code to get something working fast

Strategic: start from design to create clean (not complex) structure



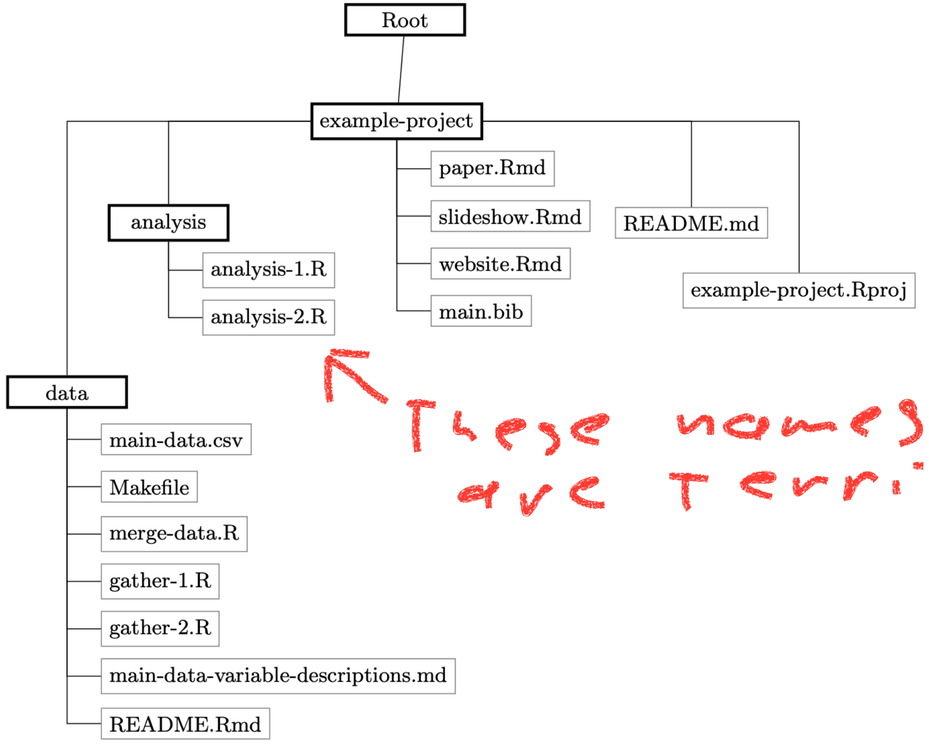
Simple File Management Example

```
# Tactical:
data <- read.csv("User/christopher/Desktop/some_data.csv")

# Strategic:
```



Simple File Management Example

```
# Tactical:
data <- read.csv("User/christopher/Desktop/some_data.csv")

# Strategic:
```



These names are terrible

Tip: Automate design

Designs for some common use cases can be automated, e.g. with ProjectTemplate, usethis.

But be careful that the automation fits the use case.

Example: need fewer docs through better design

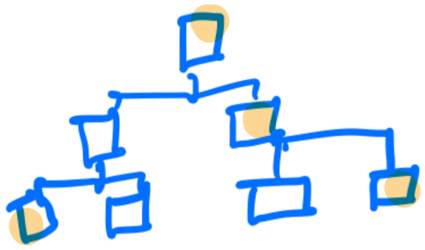
I used to try to document a project's file structure in detail in the README.md. However, I almost never kept these in sync, it was too complex.

The file structure should be obvious and not need explicit documentation.

01-author.Rmd	✏️ small copy edits 19 jan
01-stylistic-conventions.Rmd	copy edits through ch4
02-additional-resources.Rmd	(likely) final edits v3
03-introduction.Rmd	✏️ minor intro typo
04-getting-started.Rmd	✏️ small copy edits 19 jan
05-start-R.Rmd	✏️ small copy edits 19 jan
06-file-management.Rmd	✏️ small copy edits 19 jan
07-storage.Rmd	✏️ small copy edits 19 jan
08-gather.Rmd	✏️ small copy edits 19 jan
09-clean.Rmd	✏️ small copy edits 19 jan
10-modeling.Rmd	(likely) final edits v3
11-tables.Rmd	compiles in catalina with stan
12-figures.Rmd	✏️ small copy edits 19 jan
13-latex.Rmd	✏️ small copy edits 19 jan
14-web.Rmd	✏️ small copy edits 19 jan

The Danger Zone: Data Transformations

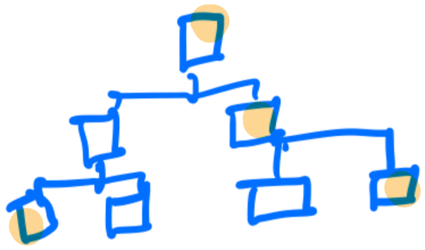
Perhaps the most tempting task for tactical programming in research software is data transformation.



Tactical data Transform

The Danger Zone: Data Transformations

Will be hard to understand and modify. Modifications will introduce bugs



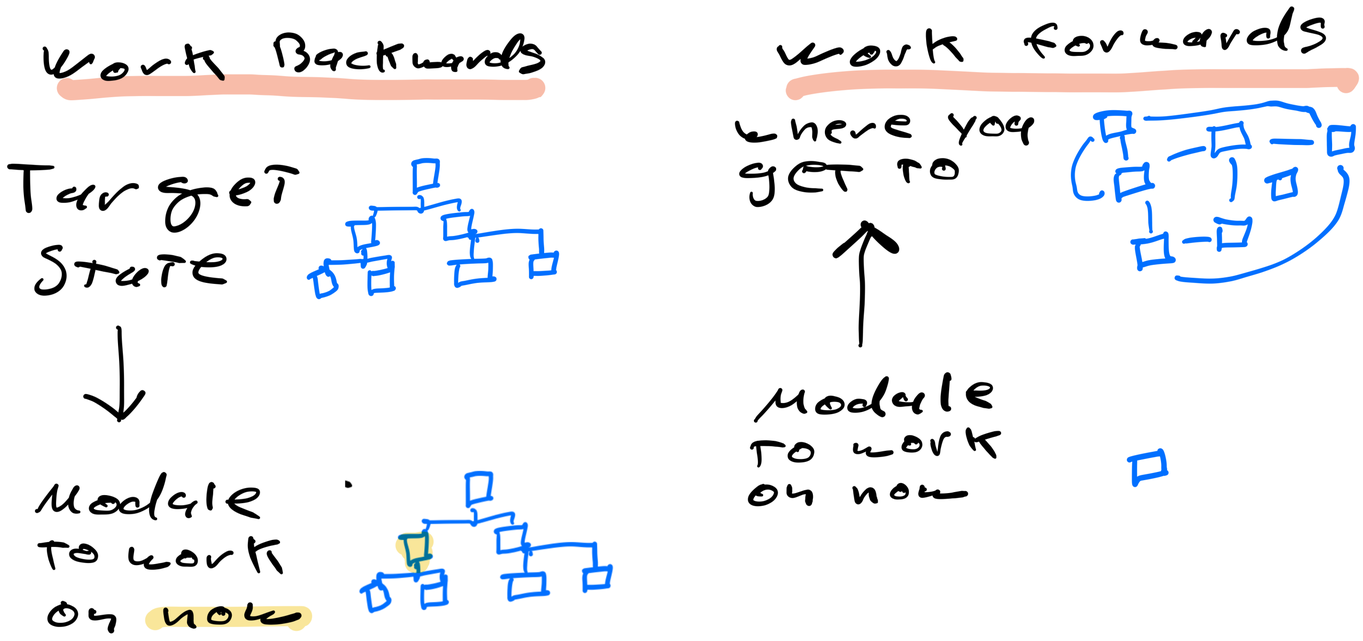
Tactical data Transform

Work Backwards

Work backwards

Start your work from an understanding of the target state to meet the use case.

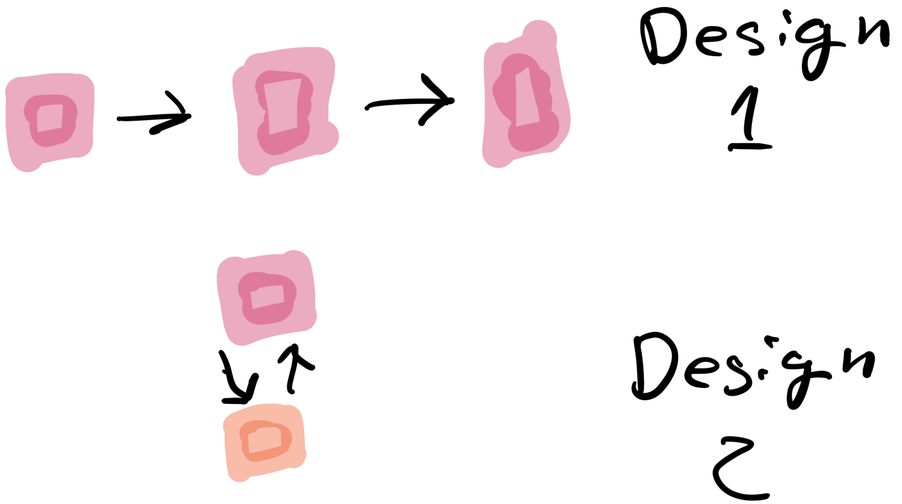
Then work backwards to the implementation.



Design it twice

Before coding, work backwards and consider alternatives.

In source code, start with comments and revise, before coding.



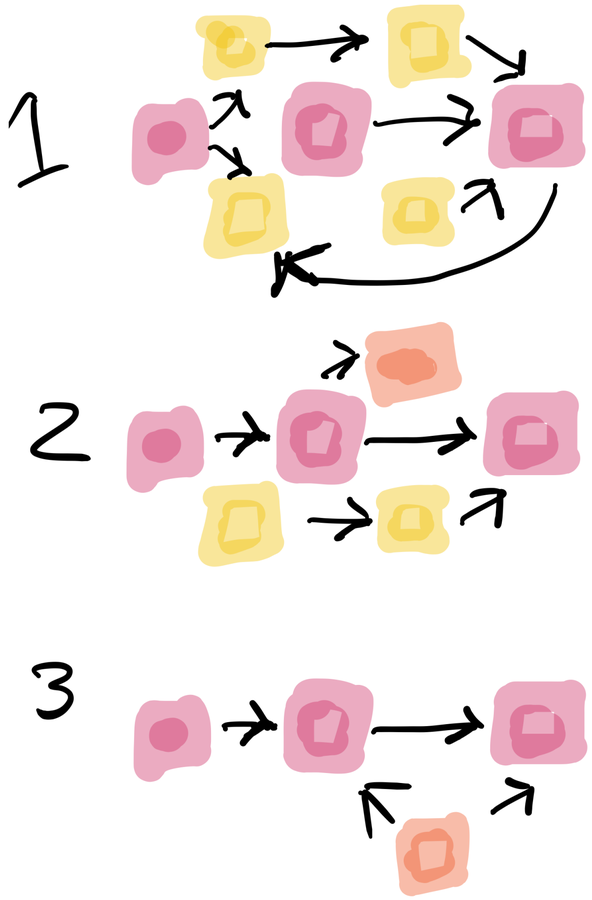
“But research is iterative. It’s hard to know the target state when starting.”

— Devil’s Advocate

Write and rewrite

Don’t just document. You won’t be able to remember it all.

Continually invest in **reducing complexity** through out a research project



Know your users

Identify the target state by understanding users and their needs

There is probably no right way to build research software for all use cases.

Understanding your users and use cases

Users of reproducible research

- The researcher’s future self
- Co-authors
- Journal reviewers
- Other researchers
- . . .

Users’ needs

What does these users need to understand and extend your research?

All benefit from reduced complexity.

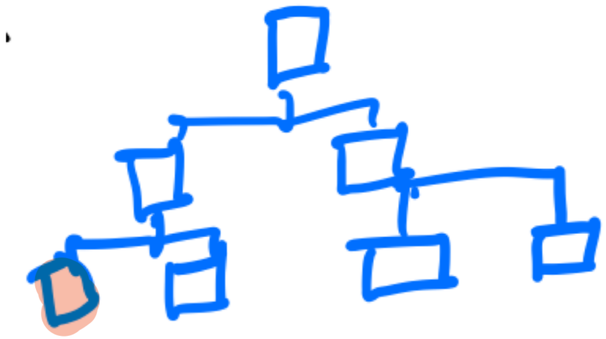
But the details matter based on what knowledge the user already has and what they are missing.

Modularise

Modularise

Do things **once** in (relatively) **independent** places, i.e. modules.

Be strategic. What is the right thing to modularise. Where in the research project?



Functions as Modules

Create functions to modularise operations.

If you are repeating an operation with minor modifications, create a function.

```
# Example, use functions to consistently transform data
transform <- function(x) {
   # Do something to x
}
```

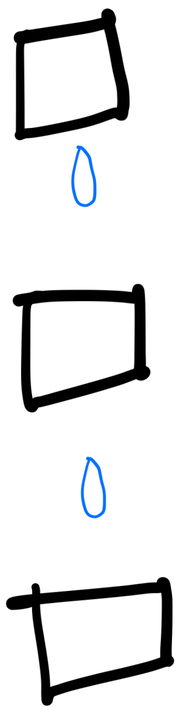
Design abstractions, not operations/code.

Focus more of your design attention on the core attributes of the module and how it interacts with the system, not the particular implementation in code.

Don’t Leak

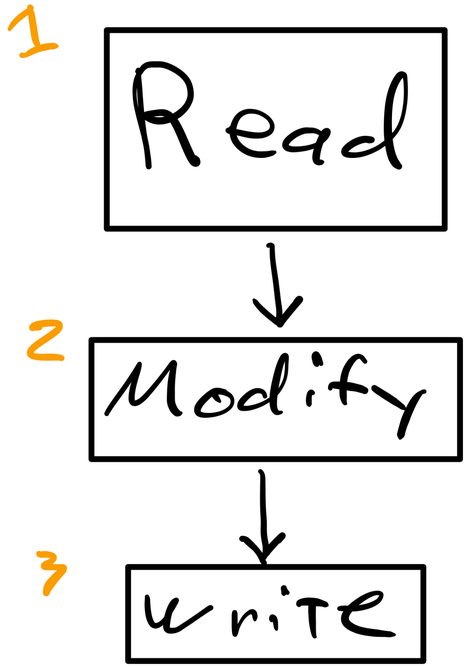
Don’t let information leak

When a design decision is reflected in many modules, requiring a change in one module to be explicitly **changed in all dependent modules**.



Tip: Avoid the temporal dependence trap

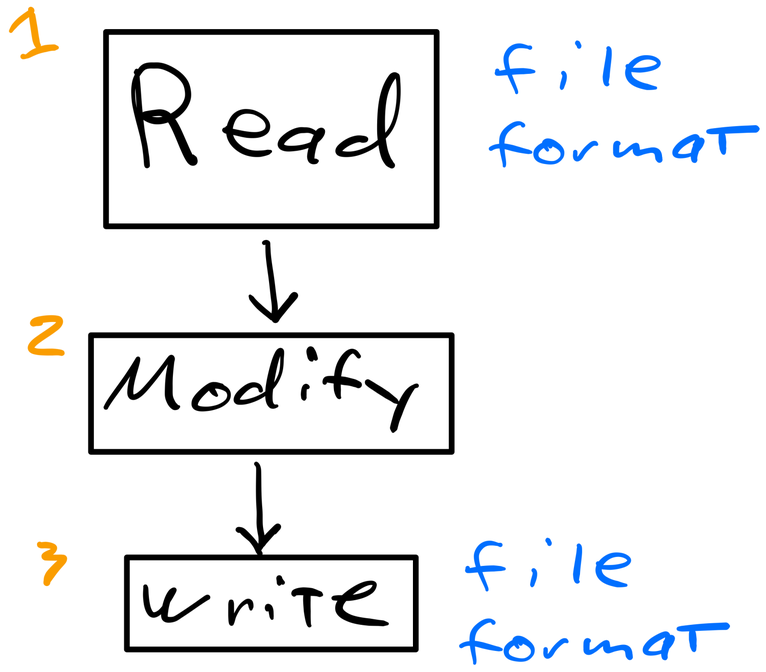
We often create functions based on the time order in which operations are performed.



Tip: Avoid the temporal dependence trap

However, this might cause information to leak.

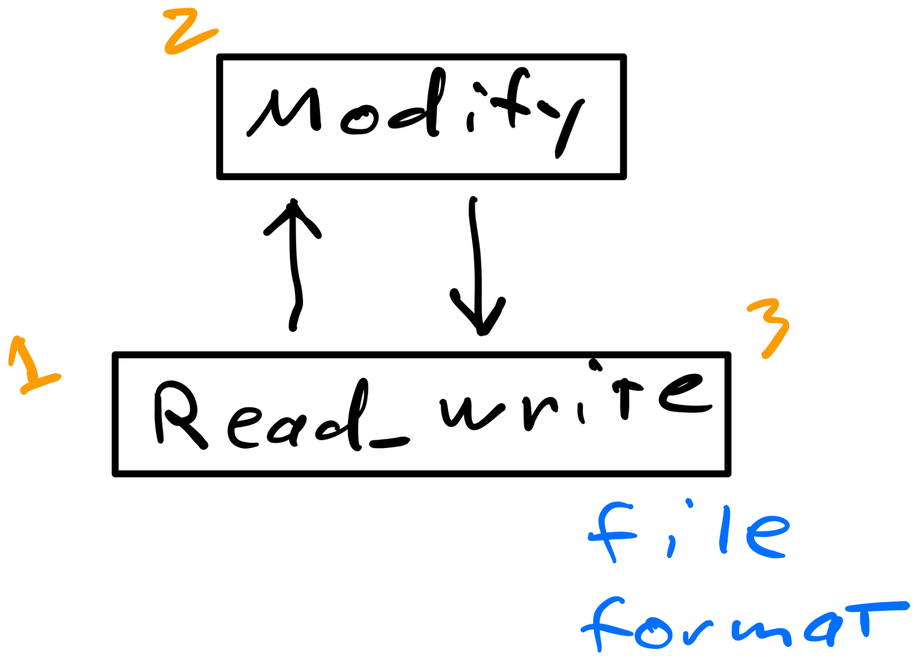
For example, if we want to read and write in the same file format, but then need to change the format, we have to remember to change in 2 places.



Tip: Avoid the temporal dependence trap

Design modules to contain information/avoid leakage.

-> Changes only need to be made in one place.

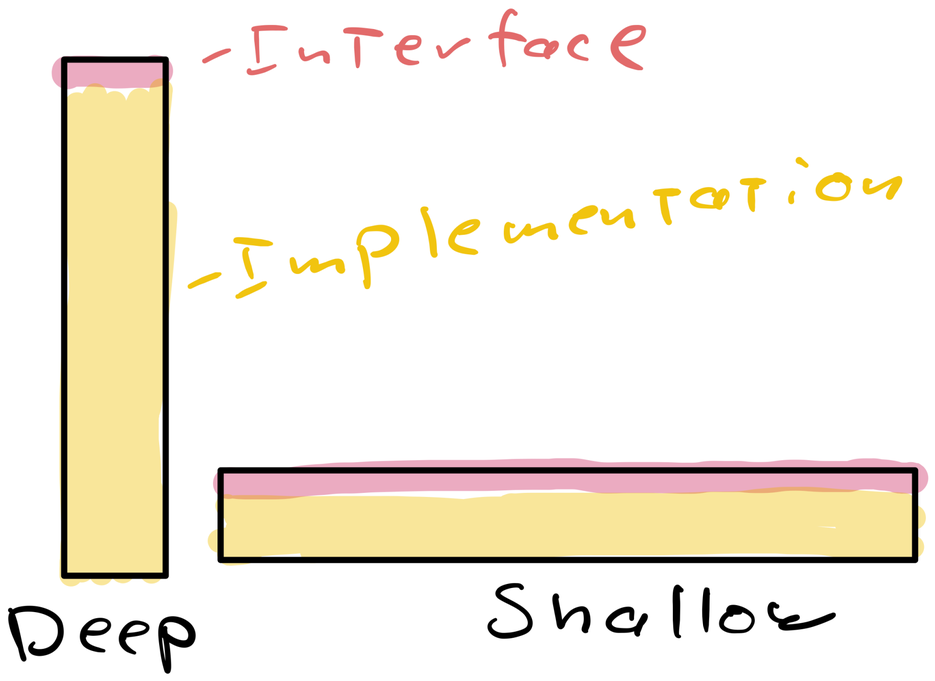


Create Deep Modules

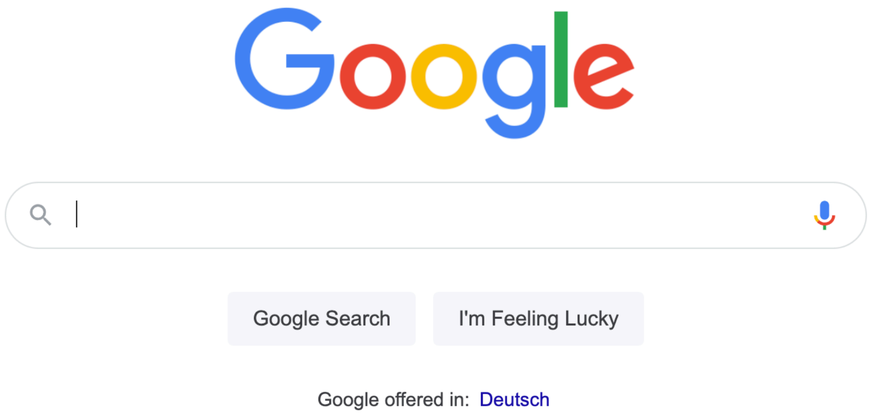
Create deep rather than shallow modules

Shallow module: Interface requires a lot of knowledge to use. Implementation doesn't do much

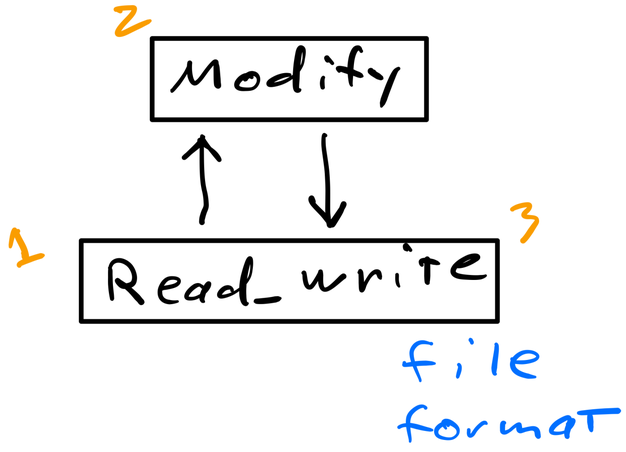
Deep module: Interface requires little knowledge, but implementation does a lot.



Example of deep module



(Toy) deep module example




```
read_write <- function(x) {
    if (inherits(x, "character"))
        read.csv(x)
    else
        write.csv(x)
}
```

simpler interface

Thin implementation

Wrap up: How will this aid reproducibility?

Understanding

Research needs to be understood to be reproduced

Reduce complexity

Software design principles help reduce complexity . . .

. . . increasing understandability

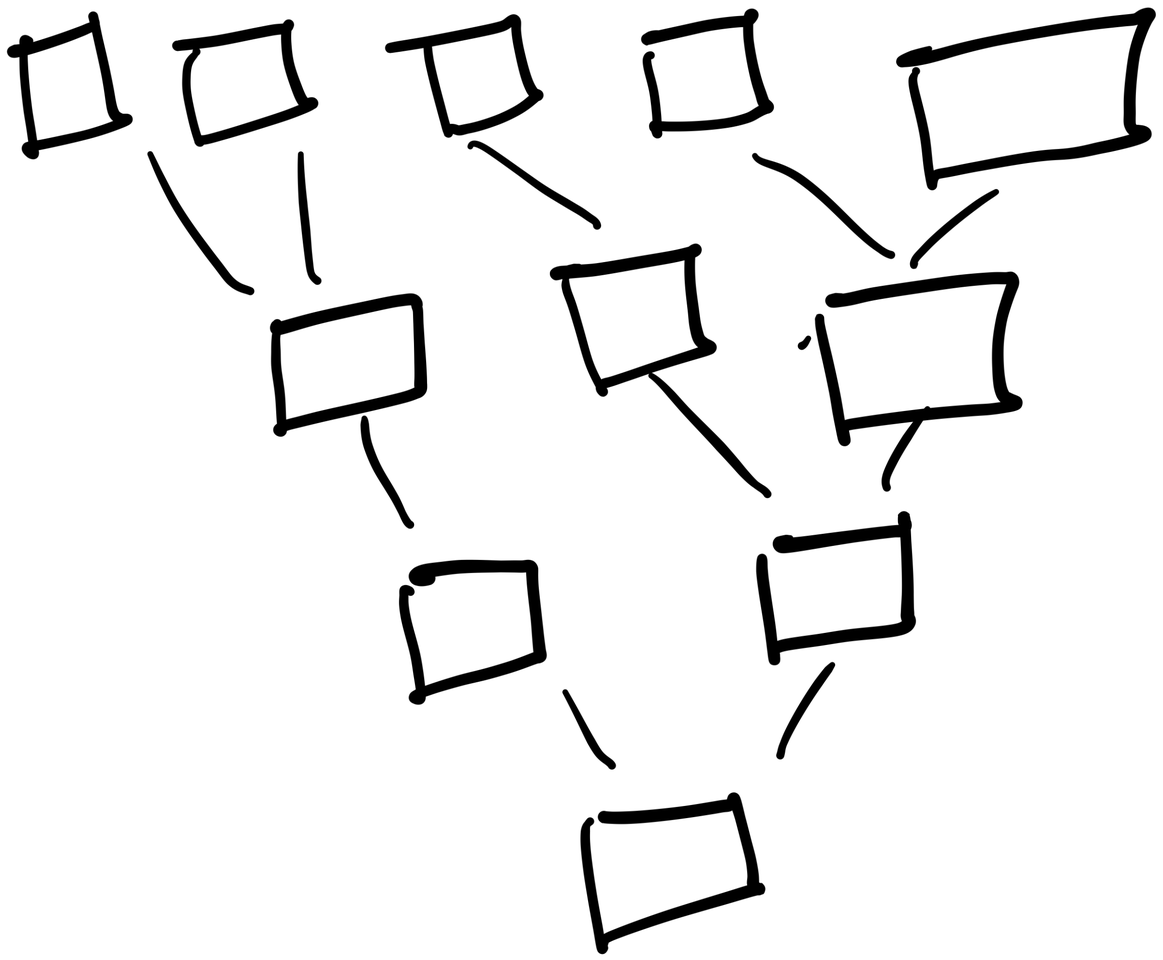
- Be strategic
- work Backwards
- Know your users
- Modularise
- Don't Leak
- Deep modules

Extensibility

Science progresses by building on each others' work.

Less complex research implementations are more extensible . . .

. . . easier for others to build on, driving science further.



Q&A