Clean coding and computational reproducibility

Russ Poldrack
Stanford University

To err is human

Professional software developers make 1 - 50 errors per 1000 lines of code

How many errors do we make as amateur developers?

How likely are they to impact our scientific conclusions?

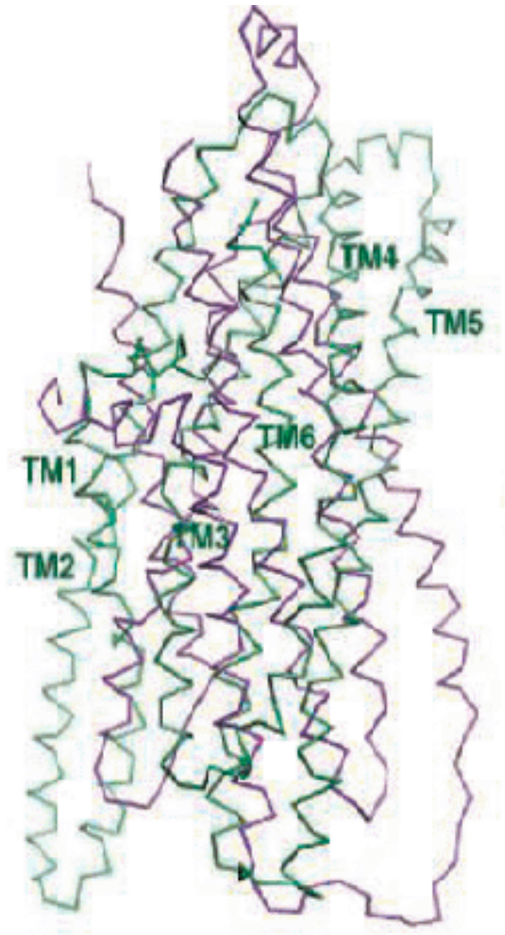
The impact of software errors

Structure of MsbA from *E. coli*: A Homolog of the Multidrug Resistance ATP Binding Cassette (ABC) Transporters

Geoffrey Chang* and Christopher B. Roth

Multidrug resistance (MDR) is a serious medical problem and presents a major challenge to the treatment of disease and the development of novel therapeutics. ABC transporters that are associated with multidrug resistance (MDR-ABC transporters) translocate hydrophobic drugs and lipids from the inner to the outer leaflet of the cell membrane. To better elucidate the structural basis for the "flip-flop" mechanism of substrate movement across the lipid bilayer, we have determined the structure of the lipid flippase MsbA from *Escherichia coli* by x-ray crystallography to a resolution of 4.5 angstroms. MsbA is organized as a homodimer with each subunit containing six transmembrane α-helices and a nucleotide-binding domain. The asymmetric distribution of charged residues lining a central chamber suggests a general mechanism for the translocation of substrate by MsbA and other MDR-ABC transporters. The structure of MsbA can serve as a model for the MDR-ABC transporters that confer multidrug resistance to cancer cells and infectious microorganisms.

www.sciencemag.org SCIENCE VOL 293 7 SEPTEMBER 2001



Geoffrey Chang

Structure of the ABC Transporter MsbA in Complex with ADP·Vanadate and Lipopolysaccharide

Christopher L. Reyes and Geoffrey Chang*

Select members of the adenosine triphosphate (ATP)–binding cassette (ABC) transporter family couple ATP binding and hydrolysis to substrate efflux and confer multidrug resistance. We have determined the x-ray structure of MsbA in complex with magnesium, adenosine diphosphate, and inorganic vanadate ($Mg{\cdot}ADP{\cdot}V_i$) and the rough-chemotype lipopolysaccharide, Ra LPS. The structure supports a model involving a rigid-body torque of the two transmembrane domains during ATP hydrolysis and suggests a mechanism by which the nucleotide-binding domain communicates with the transmembrane domain. We propose a lipid "flip-flop" mechanism in which the sugar groups are sequestered in the chamber while the hydrophobic tails are dragged through the lipid bilayer.

13 MAY 2005 VOL 308 SCIENCE www.sciencemag.org

X-ray Structure of the EmrE Multidrug Transporter in Complex with a Substrate

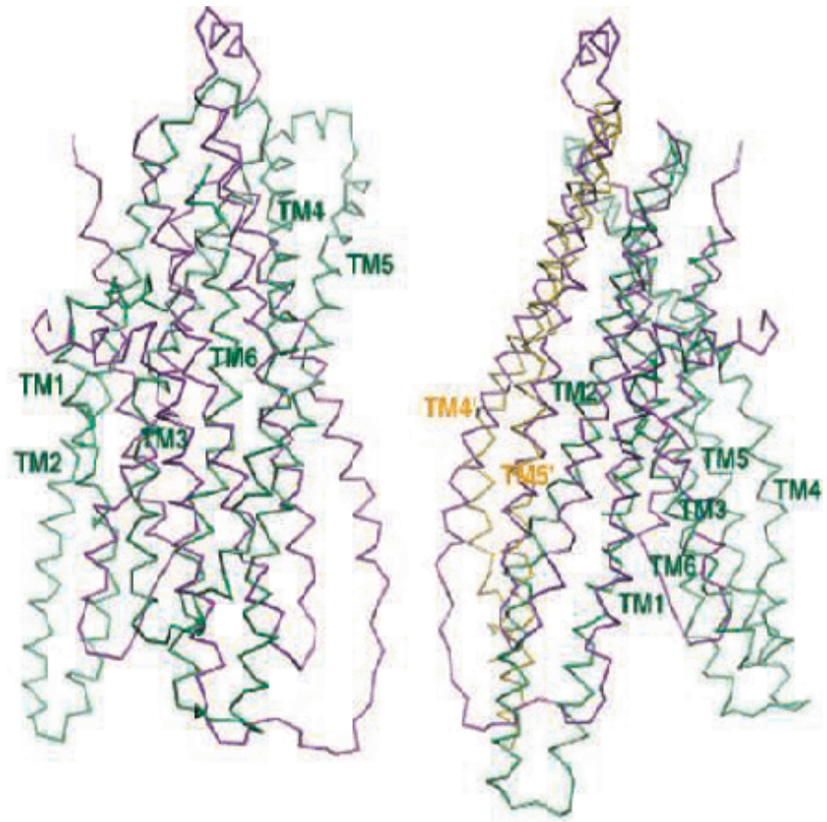
Owen Pornillos, Yen-Ju Chen, Andy P. Chen, Geoffrey Chang*

EmrE is a prototype of the Small Multidrug Resistance family of efflux transporters and actively expels positively charged hydrophobic drugs across the inner membrane of *Escherichia coli*. Here, we report the x-ray crystal structure, at 3.7 angstrom resolution, of one conformational state of the EmrE transporter in complex with a translocation substrate, tetraphenylphosphonium. Two EmrE polypeptides form a homodimeric transporter that binds substrate at the dimerization interface. The two subunits have opposite orientations in the membrane and adopt slightly different folds, forming an asymmetric antiparallel dimer. This unusual architecture likely confers unidirectionality to transport by creating an asymmetric substrate translocation pathway. On the basis of available structural data, we propose a model for the proton-dependent drug efflux mechanism of EmrE.

23 DECEMBER 2005 VOL 310 SCIENCE www.sciencemag.org

The impact of software errors

Retraction



WE WISH TO RETRACT OUR RESEARCH ARTICLE "STRUCTURE OF MsbA from *E. coli*: A homolog of the multidrug resistance ATP binding cassette (ABC) transporters" and both of our Reports "Structure of the ABC transporter MsbA in complex with ADP•vanadate and lipopolysaccharide" and "X-ray structure of the EmrE multidrug transporter in complex with a substrate" (*1*–*3*).

The recently reported structure of Sav1866 (*4*) indicated that our MsbA structures (*1*, *2*, *5*) were incorrect in both the hand of the structure and the topology. Thus, our biological interpretations based on these inverted models for MsbA are invalid.

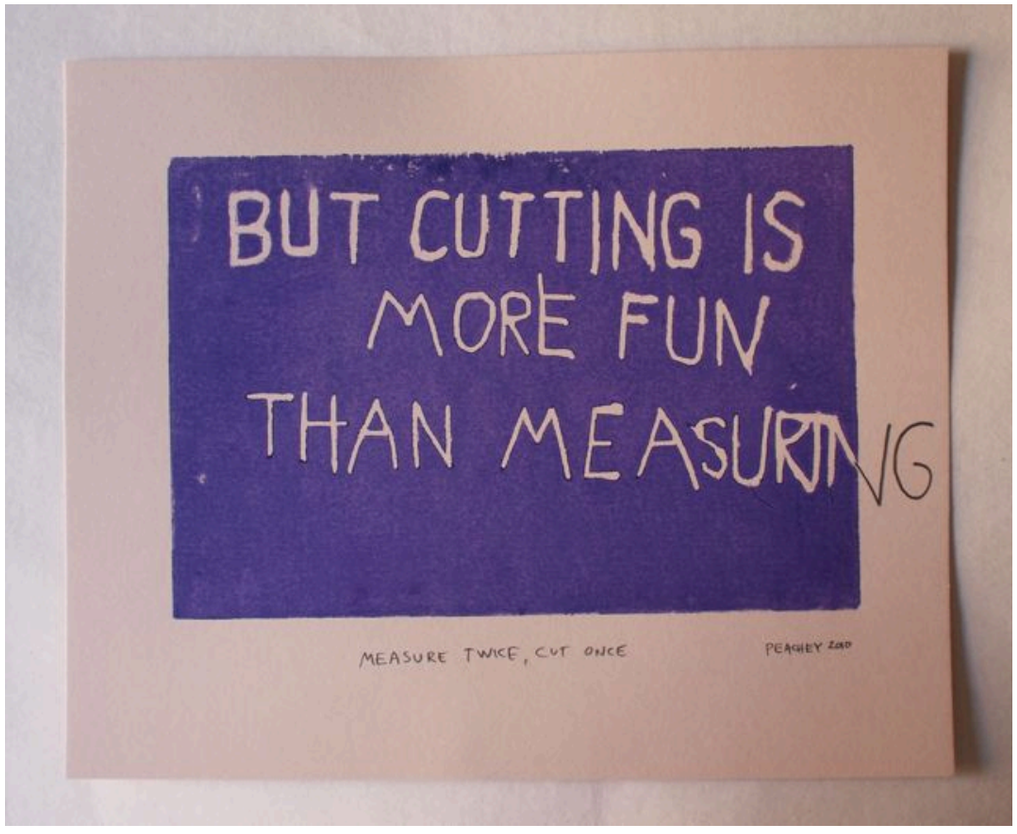
An in-house data reduction program introduced a change in sign for anomalous differences. This program, which was not part of a conventional data processing package, converted the anomalous pairs (I+ and I–) to (F– and F+), thereby introducing a sign change. As the diffraction data collected for each set of MsbA crystals and for the EmrE crystals were processed with the same program, the structures reported in (*1*–*3*, *5*, *6*) had the wrong hand.

Coding for reproducibility

- Learning clean and reproducible coding practices increases efficiency in the long run
  - “As in real carpentry — the kind done with lumber — the time saved by measuring carefully before cutting a piece of wood is much greater than the time that measuring takes.” - https://swcarpentry.github.io/python-novice-inflammation/10-defensive/index.html

Coding for reproducibility

- Learning clean and reproducible coding practices increases efficiency in the long run
  - "As in real carpentry — the kind done with lumber — the time saved by measuring carefully before cutting a piece of wood is much greater than the time that measuring takes." - https://swcarpentry.github.io/python-novice-inflammation/10-defensive/index.html

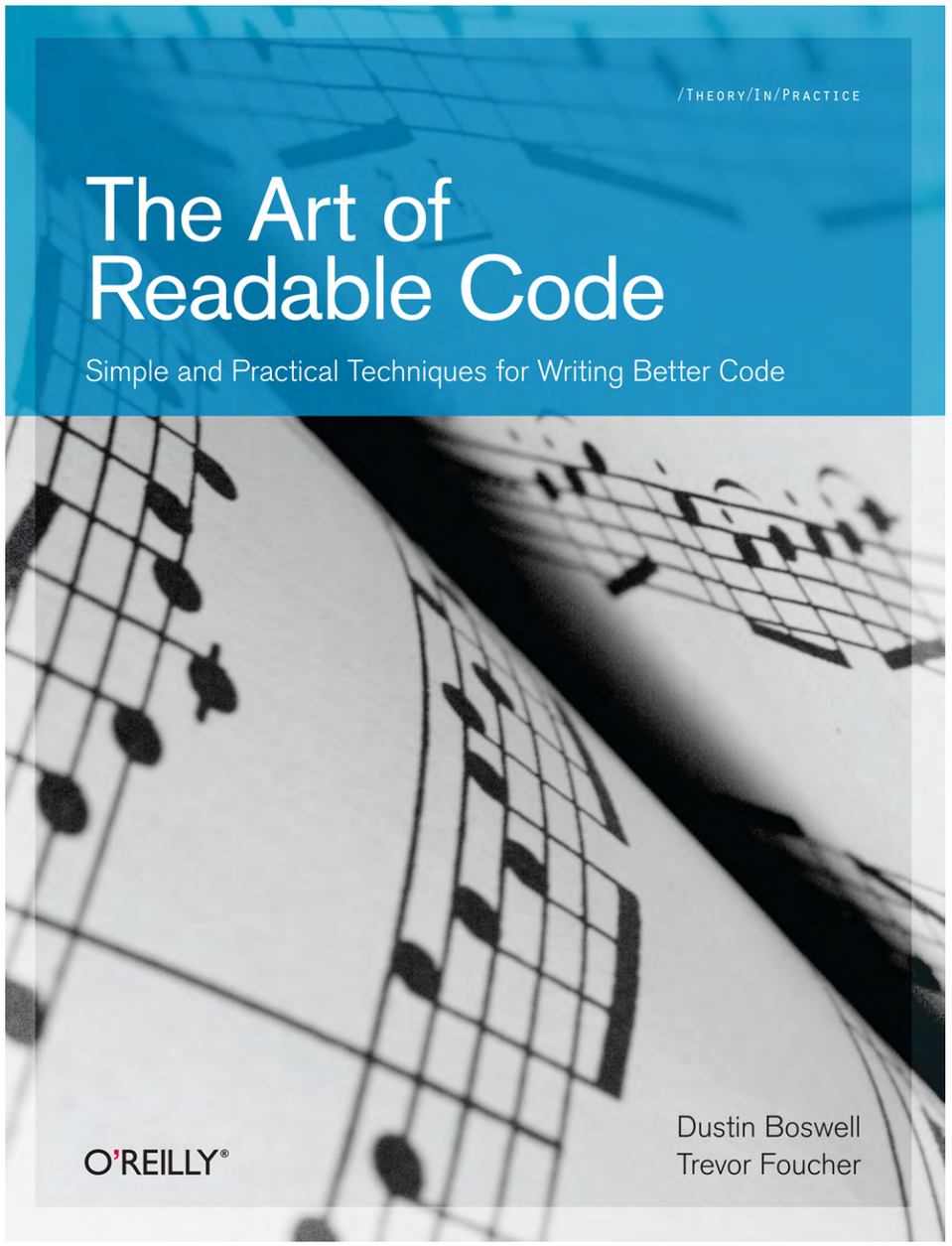


Good Coding Practices

- Code understandably
- Code defensively
- Code portably
- Automate as much as possible

Coding understandably

- "Code is read much more often than it is written" - G. van Rossum
- You should try to minimize the time it would take for someone else to understand your code
- Where "someone else" also includes your future self!



Naming

- Names should be as informative as possible
  - “A variable’s name is like a tiny comment” (Boswell & Foucher)

```
m = {'Jets': ['Riff', 'Tony', ‘Diesel’], 'Sharks': ['Bernardo', 'Chino', 'Pepe']}
c = ['Doc', 'Shrank', 'Krupke', 'Riff', 'Tony', 'Diesel', 'Bernardo', 'Chino', 'Pepe']

for k in range(len(c)):
    for i in m:
        for j in range(len(m[i])):
            if m[i] == c[j]:
                print('%s is in %s' % (m[i], c[j])
```

Naming

- Names should be as informative as possible
  - “A variable’s name is like a tiny comment” (Boswell & Foucher)

```
m = {'Jets': ['Riff', 'Tony', ‘Diesel’], 'Sharks': ['Bernardo', 'Chino', 'Pepe']}
c = ['Doc', 'Shrank', 'Krupke', 'Riff', 'Tony', 'Diesel', 'Bernardo', 'Chino', 'Pepe']

for k in range(len(c)):
    for i in m:
        for j in range(len(m[i])):
            if m[i] == c[j]:
print('%s is in %s' % (m[i], c[j])
```

incorrect index

Naming

- Names should be as informative as possible
  - “A variable’s name is like a tiny comment” (Boswell & Foucher)

```
m = {'Jets': ['Riff', 'Tony', ‘Diesel’], 'Sharks': ['Bernardo', 'Chino', 'Pepe']}
c = ['Doc', 'Shrank', 'Krupke', 'Riff', 'Tony', 'Diesel', 'Bernardo', 'Chino', 'Pepe']

for k in range(len(c)):
    for i in m:
        for j in range(len(m[i])):
            if m[i] == c[j]:
                print('%s is in %s' % (m[i], c[j])
```

incorrect index

```
club_members = {
    'Jets': ['Riff', 'Tony', 'Diesel'],
    'Sharks': ['Bernardo', 'Chino', 'Pepe']
}
cast_members = ['Doc', 'Shrank', 'Krupke', 'Riff', 'Tony', 'Diesel', 'Bernardo', 'Chino', 'Pepe']

for cast_member in cast_members:
    for club in club_members:
        if cast_member in club_members[club]:
            print(f'{cast_member} is in {club}')
```

Following style guidelines

- Python:
  - PEP8: https://www.python.org/dev/peps/pep-0008/
- R
  - Tidyverse style guide: https://style.tidyverse.org/
  - Google R style guide: https://google.github.io/styleguide/Rguide.html
- "A Foolish Consistency is the Hobgoblin of Little Minds"
  - If following the guideline is going to make your code worse, then don't do it
- Most important:
  - Be consistent within your own code

Standard naming conventions

- Python PEP8 naming conventions
  - Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.
    - In some fonts, these characters are indistinguishable from the numerals one and zero. When tempted to use 'l', use 'L' instead.
  - Variable and function names should be lowercase, with words separated by underscores as necessary to improve readability (“snake case”)

Be as specific as possible

- Don’t use generic words for naming
  - `size = 3`
    - size of what? what units?
    - better:
      - `num_nodes`, `length_cm`
  - `get_page(loc)`
    - get from where? how?
    - better:
      - `download_page_from_url(url_to_download)`

Avoid “magic numbers”

- Any numeric values should be assigned to a clearly named variable

```
c = 0
for i in range(1000):
    if my_number_generator() > 1.96:
        c += 1
print(c/1000)
```

Avoid “magic numbers”

- Any numeric values should be assigned to a clearly named variable

```
c = 0
for i in range(1000):
    if my_number_generator() > 1.96:
        c += 1
print(c/1000)
```

what if this line occurred much later in the code? would you remember to change it?

Avoid “magic numbers”

- Any numeric values should be assigned to a clearly named variable

```
c = 0
for i in range(1000):
    if my_number_generator() > 1.96:
        c += 1
print(c/1000)
```

what if this line occurred much later in the code? would you remember to change it?

```
exceedence_counter = 0
num_simulation_runs = 1000
threshold = 1.96

for run in range(num_simulation_runs):
    if my_number_generator() > threshold:
        exceedence_counter += 1
print(exceedence_counter/num_simulation_runs)
```

Using horizontal white space

- Python explicitly uses white space as a syntactic element
- In R, it's important to use white space to call out structure

```
for (trial_number in 1:nrow(trial_data)){
if (trial_data[trial_number, 'stop_signal_presented']){
if (!is.na(trial_data[trial_number, 'response'])){
if (trial_data[trial_number, 'accurate_response']){
trial_data[trial_number, 'trial_type'] <- 'StopFail'
} else{trial_data[trial_number, 'trial_type'] <- 'StopFailError'
}} else{trial_data[trial_number, 'trial_type'] <- 'StopSuccess'
}} else {if (trial_data[trial_number, 'accurate_response']){
trial_data[trial_number, 'trial_type'] <- 'Go'
} else {trial_data[trial_number, 'trial_type'] <- 'GoError'}}}
```

Using horizontal white space

- Python explicitly uses white space as a syntactic element
- In R, it's important to use white space to call out structure

```
for (trial_number in 1:nrow(trial_data)){
  if (trial_data[trial_number, 'stop_signal_presented']){
    if (!is.na(trial_data[trial_number, 'response'])){
      if (trial_data[trial_number, 'accurate_response']){
        trial_data[trial_number, 'trial_type'] <- 'StopFail'
      } else{
        trial_data[trial_number, 'trial_type'] <- 'StopFailError'
      }
    } else{
      trial_data[trial_number, 'trial_type'] <- 'StopSuccess'
    }
  } else {
    if (trial_data[trial_number, 'accurate_response']){
      trial_data[trial_number, 'trial_type'] <- 'Go'
    } else {
      trial_data[trial_number, 'trial_type'] <- 'GoError'
    }
  }
}
```

Using vertical white space

- Use empty lines to separate coherent pieces of code into “paragraphs”

```
results <- list()
stoptrials <- trial_data %>%
  dplyr::filter(str_detect(trial_type, 'Stop'))
results$p_stop_success <- stoptrials %>%
  dplyr::summarize(p_stop_success=mean(trial_type=='StopSuccess')) %>%
  dplyr::pull(p_stop_success)
gotrials <- trial_data %>%
  dplyr::filter(str_detect(trial_type, 'Go'))
results$go_accuracy <-gotrials %>%
  dplyr::summarize(go_accuracy=mean(trial_type=='Go')) %>%
  dplyr::pull(go_accuracy)
results$median_go_rt <- gotrials %>%
  dplyr::filter(trial_type=='Go') %>%
  dplyr::summarize(median_go_rt=median(rt)) %>%
  dplyr::pull(median_go_rt)
results$SSRT <- trial_data %>%
  dplyr::summarize(mean_SSD=mean(SSD, na.rm=TRUE),
                   SSRT=results$median_go_rt - mean_SSD) %>%
  pull(SSRT)
display_report(results)
```

Using vertical white space

```
results <- list()

# extract stop trials and compute stop success
stoptrials <- trial_data %>%
  dplyr::filter(str_detect(trial_type, 'Stop'))
results$p_stop_success <- stoptrials %>%
  dplyr::summarize(p_stop_success=mean(trial_type=='StopSuccess')) %>%
  dplyr::pull(p_stop_success)

# extract go trials and compute median RT and accuracy
gotrials <- trial_data %>%
  dplyr::filter(str_detect(trial_type, 'Go'))
results$go_accuracy <-gotrials %>%
  dplyr::summarize(go_accuracy=mean(trial_type=='Go')) %>%
  dplyr::pull(go_accuracy)
results$median_go_rt <- gotrials %>%
  dplyr::filter(trial_type=='Go') %>%
  dplyr::summarize(median_go_rt=median(rt)) %>%
  dplyr::pull(median_go_rt)

# compute SSRT
results$SSRT <- trial_data %>%
  dplyr::summarize(mean_SSD=mean(SSD, na.rm=TRUE),
                   SSRT=results$median_go_rt - mean_SSD) %>%
  pull(SSRT)

display_report(results)
```

Namespaces in R

- When you call a function, it's important to know where it is coming from
  - This becomes challenging in R, as imported libraries can overwrite existing functions

```
> library(tidyverse)
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──
✓ ggplot2 3.3.2     ✓ purrr   0.3.4
✓ tibble  3.0.2     ✓ dplyr   1.0.0
✓ tidyr   1.1.0     ✓ stringr 1.4.0
✓ readr   1.3.1     ✓ forcats 0.5.0
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

  - Whenever using a function from an imported library, you should explicitly specify the namespace
    - **`dplyr::filter(x)`** rather than **`filter(x)`**
  - For similar reasons, never use **`import *`** in python

Commenting is not a magic balm for bad code

- "Don't comment bad code — rewrite it" (Kernighan & Plaugher)
- Comments should help the reader to know as much as the writer knows about the code
- "Clear and expressive code with few comments is far superior to cluttered and complex code with lots of comments" (Robert Martin, *Clean Code*)

What *not* to comment

- Ugly or indecipherable code
  - Refactor so that the code is understandable

```
# function to create a random normal variate
# takes mean and SD as arguments
def crv(a, b):
  …
```

- instead, use understandable names:

```
def create_random_normal_variate(mean, sd):
  …
```

What *not* to comment

- The obvious

```
# looping over states
for state_index, state in enumerate(states):
  …
```

```
# function to create a random normal variate
def create_random_normal_variate(mean, sd):
  …
```

What *not* to comment

- Historical information

```
# RP changed on 8/12/2020 to include sd argument
def create_random_normal_variate(mean, sd):
  …
```

  - Should instead be tracked in version control system

```
git commit -m”added sd argument to create_random_normal_variate”
```

What to comment

- Intention (aka “Director Commentary”)

```
# using a numpy array here rather than
# a pandas DataFrame for performance reasons
```

- Known flaws or TODO items

```
# TODO: Currently this will fail if x == pi
# - should include code to address that edge case
```

- Comments on your constant values

```
# Using 500 bootstrap replicates based on
# Efron & Tibshirani 1993
```

Coding defensively

- Assume that errors will occur, and create code that is either:
  - Robust to errors
  - Detects and announces errors loudly
- Both of these are generally easier to do in Python vs. R

Assertions

- Assert that some statement must be true
  - Otherwise signal an error
- Python

```
In [5]: num_friends = -4

In [6]: assert num_friends >= 0
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-6-3e7acfeb5ed4> in <module>
----> 1 assert num_friends >= 0

AssertionError:
```

- R

```
> num_friends = -4
> assert_that(num_friends >= 0)
Error: num_friends not greater than or equal to 0
```

PLOS | ONE

CORRECTION

Correction: The Role of Conspiracist Ideation and Worldviews in Predicting Rejection of Science

Stephan Lewandowsky, Gilles E. Gignac, Klaus Oberauer

The dataset included two notable age outliers (reported ages 5 and 32757).

Specifically, the statement on page 9 “age turned out not to correlate with any of the indicator variables” is incorrect. It should read instead “age correlated significantly with 3 latent indicator variables (Vaccinations: .219, p < .0001; Conservatism: .169, p < .001; Conspiracist ideation: -.140, maximum likelihood p < .0001, bootstrapped p = .004), and straddled significance for a fourth (Free Market: .08, p%.05).”

```
In [1]: age=32757

In [2]: assert age>12 and age<120
---------------------------------------------
AssertionError                              Traceback (most recent call last)
<ipython-input-2-37de876b5fda> in <module>()
----> 1 assert age>12 and age<120

AssertionError:
```

Software testing

- Important functions should be tested
  - Determine what your code should do, and write a test to make sure that it does that thing properly
  - Determine the cases in which your code should raise exceptions, and make sure it properly raises them
- "Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead." — Martin Fowler
- Python
  - https://docs.pytest.org/en/stable/
- R
  - https://testthat.r-lib.org/

Testing using pytest

```
from rtanalysis.rtanalysis import RTAnalysis

def test_rtanalysis_smoke():
    rta = RTAnalysis()
    assert rta is not None
```

```
pytest_tutorial % python -m pytest tests/test_1_smoketest.py
============================= test session starts ==============================
platform darwin -- Python 3.8.3, pytest-5.4.1, py-1.8.1, pluggy-0.13.1
rootdir: /Users/poldrack/Dropbox/code/pytest_tutorial
plugins: cov-2.10.0
collected 1 item

rtanalysis/test_1_smoketest.py .                                         [100%]

============================== 1 passed in 0.25s ===============================
```

https://github.com/poldrack/pytest_tutorial

Code portably

- Don’t hard-code machine-dependent details into code
  - File paths, system names, credentials
  - Instead, either use environment variables or configuration files
    - Be sure not to check configuration files into github!
      - use the .gitignore file
- JSON files are very convenient in Python
  - easily save and retrieve Python dicts

Automating your workflow

- Goal: be able to re-run the entire workflow on a new machine with a single command
- Simple tool: UNIX make

Automating your workflow

- Goal: be able to re-run the entire workflow on a new machine with a single command
- Simple tool: UNIX make

`Makefile:`

```
all: setup analysis

setup:
    python do_setup.py

analysis:
    python run_analysis_step1.py
    python run_analysis_step2.py
```

Automating your workflow

- Goal: be able to re-run the entire workflow on a new machine with a single command
- Simple tool: UNIX make

Makefile:

```
all: setup analysis

setup:
    python do_setup.py

analysis:
    python run_analysis_step1.py
    python run_analysis_step2.py
```

```
(py38) % make setup
python do_setup.py
doing setup
```

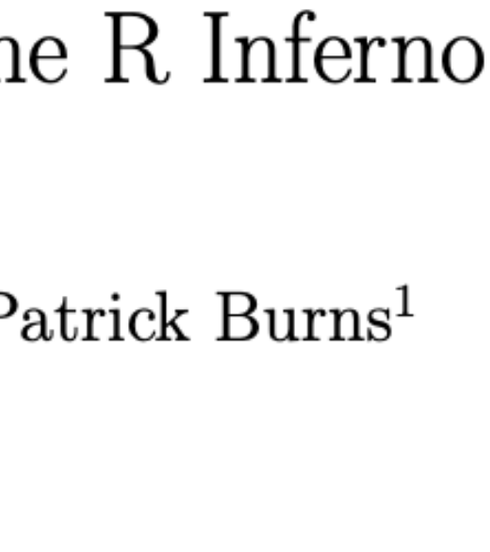
```
(py38) % make analysis
python run_analysis_step1.py
doing analysis step 1
python run_analysis_step2.py
doing analysis step 2
```

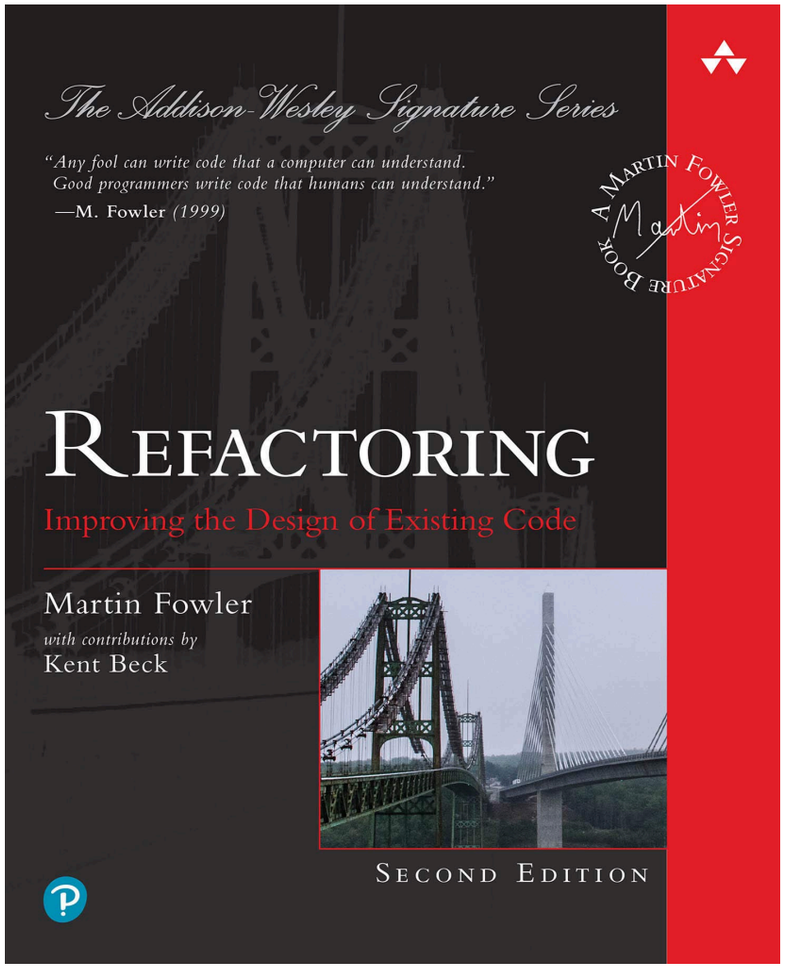
```
(py38) % make all
python do_setup.py
doing setup
python run_analysis_step1.py
doing analysis step 1
python run_analysis_step2.py
doing analysis step 2
```

Learning what not to do

https://docs.quantifiedcode.com/python-anti-patterns/

- "anti-patterns"
  - commonly used bad solutions to common problems
- "code smells"
  - Patterns in software that signal potential problems

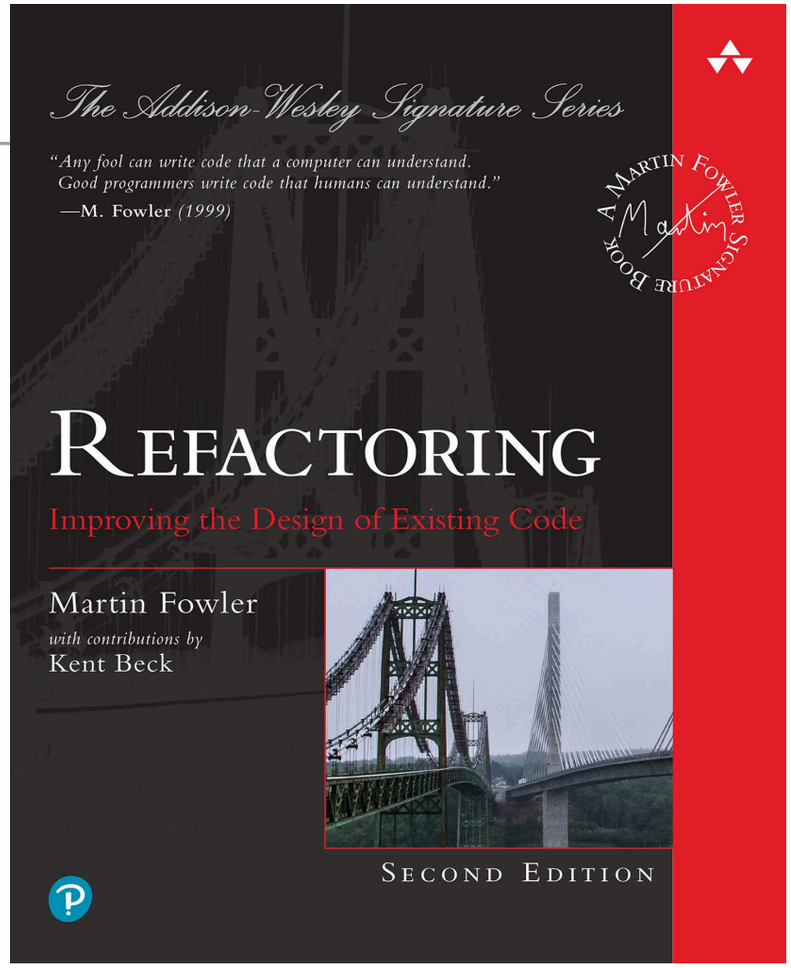




https://www.burns-stat.com/pages/Tutor/R_inferno.pdf

Common “bad smells” in code

- Mysterious name
- Duplicated code
- Long function
- Long parameter list
- Global data
- Mutable data
- Requirement for changes in multiple places
- Repeated switches
- Comments
  - “comments aren’t a bad smell; indeed they are a sweet smell. The reason we mention comments here is that comments are often used as a deodorant. It’s surprising how often you look at thickly commented code and notice that the comments are there because the code is bad.”



The DRY (Don't Repeat Yourself) principle

- Creating multiple variables with numbers or names in them is a bad practice ('anti-pattern')

```
test1 = 'stroop'
test2 = 'flanker'
test3 = 'nback'
```

```
result1 = analyze(test1)
result2 = analyze(test2)
result3 = analyze(test3)
```

The DRY (Don't Repeat Yourself) principle

- Creating multiple variables with numbers or names in them is a bad practice ('anti-pattern')

```
tes
tes
tes


res
res
res
```

"code smell"

The DRY (Don't Repeat Yourself) principle

- Creating multiple variables with numbers or names in them is a bad practice ('anti-pattern')

```
test1 = 'stroop'
test2 = 'flanker'
test3 = 'nback'
```

```
result1 = analyze(test1)
result2 = analyze(test2)
result3 = analyze(test3)
```

The DRY (Don't Repeat Yourself) principle

- Creating multiple variables with numbers or names in them is a bad practice ('anti-pattern')

```
test1 = 'stroop'
test2 = 'flanker'
test3 = 'nback'
```

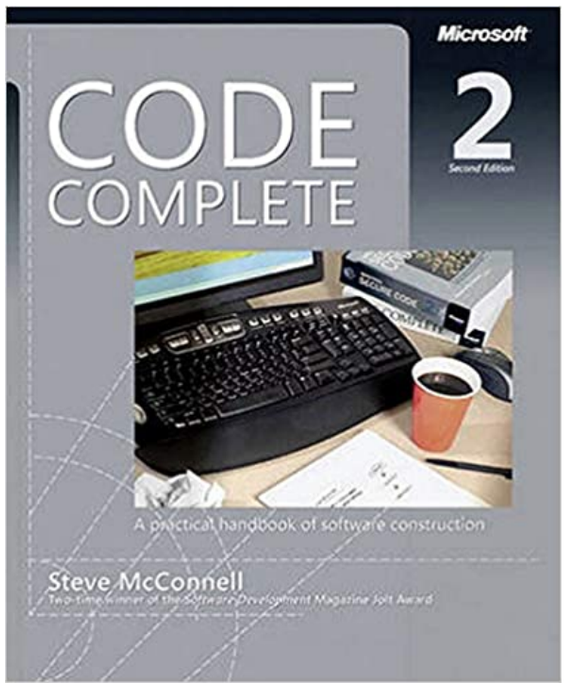
```
tests = [
    'stroop',
    'flanker',
    'nback']
```

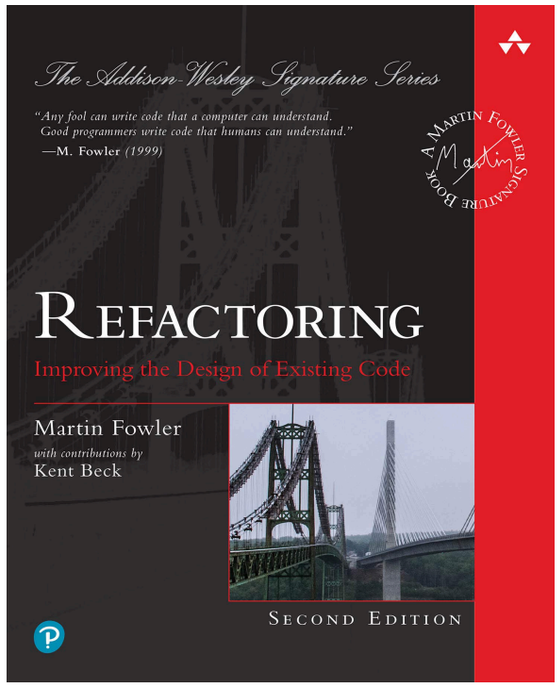
```
result1 = analyze(test1)
result2 = analyze(test2)
result3 = analyze(test3)
```

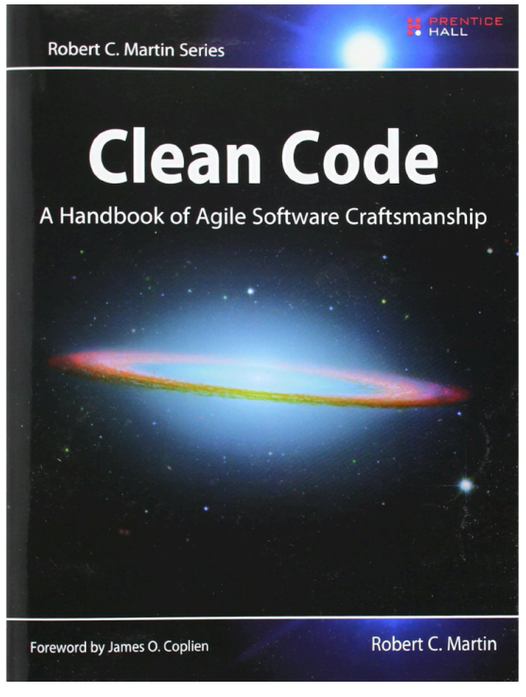
```
results = {}
for test in tests:
  results[test] = analyze(test)
```

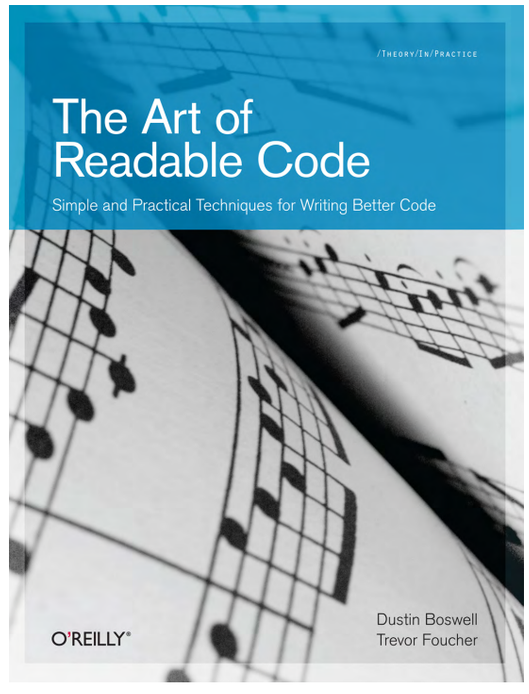
Implementing good coding practices

- Code review
  - You write code, someone else reviews it
  - Good for highlighting readability problems
- Pair programming
  - work together with someone else synchronously
  - VSCode has a Live Share mode (ala Google Docs)
- Learn about practical software engineering practices









Breakout group exercise

- Look at this code:
  - https://github.com/poldrack/clean_coding/blob/master/R_example/AntiExample.Rmd
  - (don't peek at the other versions of this code in the repo!)
- With your group, spend 10 minutes going over the code, with the following questions in mind:
  - What are the different parts of the code doing?
  - What code smells do you sense?
  - How might you refactor it to address these?
- Appoint a group spokesperson who can report back if called upon