Code Smells and Feels

rstd.io/code-smells

Jennifer Bryan
RStudio

@jennybc
@JennyBryan

Slides not meant to stand alone!

See full references here:

rstd.io/code-smells

which currently points to

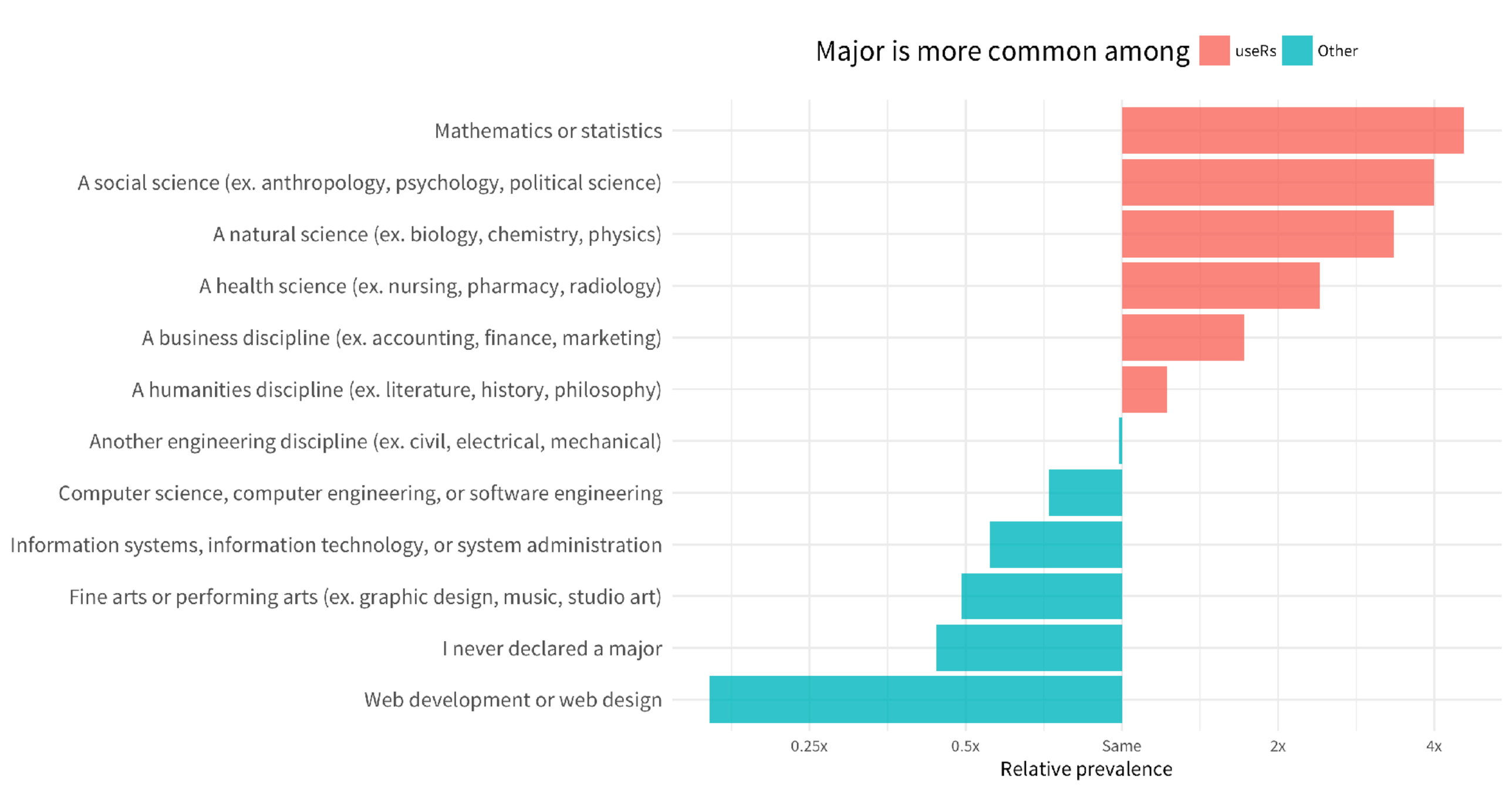
https://github.com/jennybc/code-smells-and-feels#readme

German & Econ major

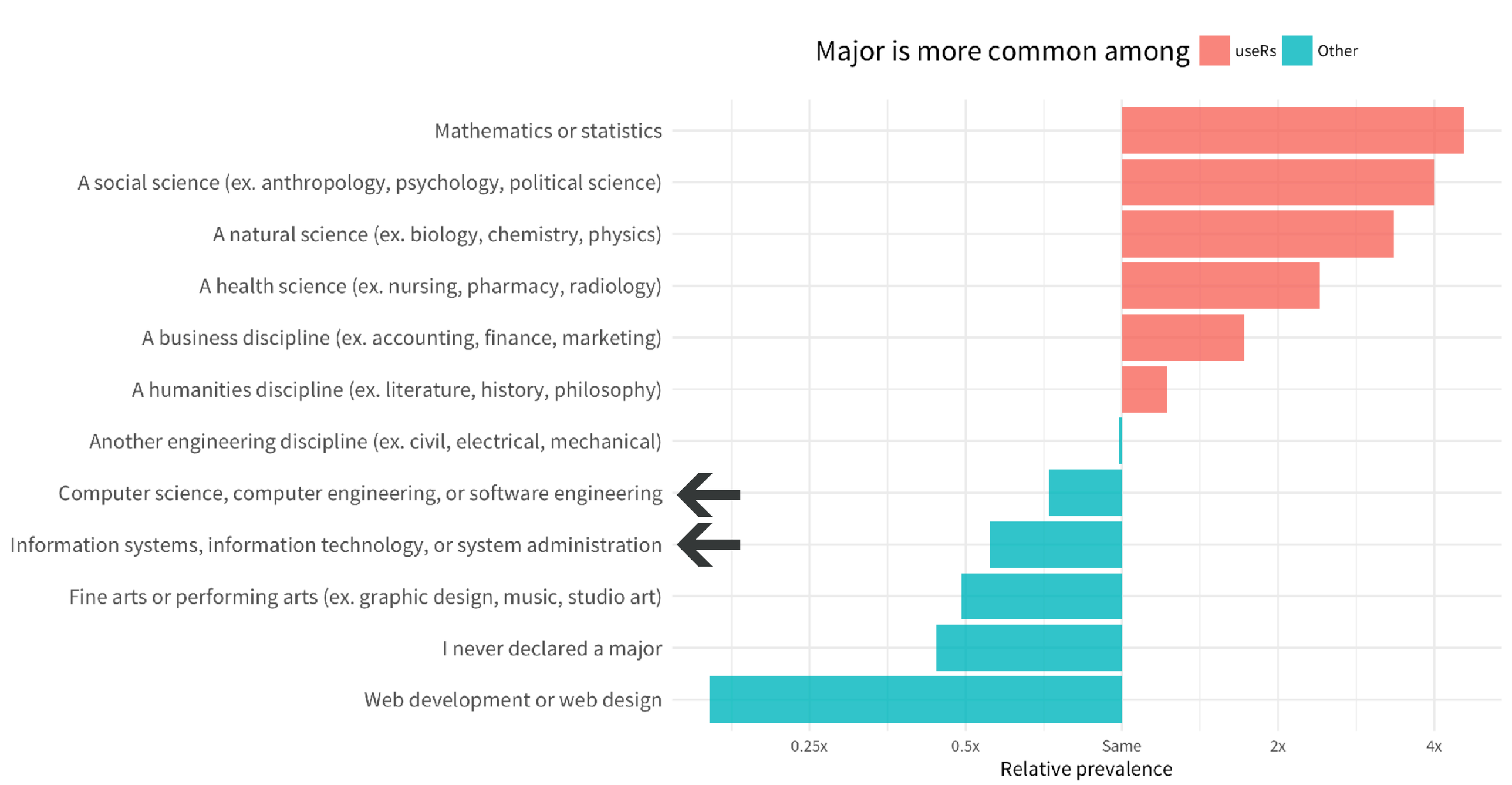
Management Consultant

(Bio)statistics PhD → Prof

Software Engineer



Julia Silge & Jenny Bryan
Source: 2018 Stack Overflow Annual Developer Survey



Julia Silge & Jenny Bryan
Source: 2018 Stack Overflow Annual Developer Survey

Users Become Developers

- Good Programming Practice, by Martin Mächler
- Language Interfaces (.Call & .External), by Peter Dalgaard
- Packaging, Documentation, Testing, by Kurt Hornik

What I find important when R Programming

and Recent Cool Features in R

Martin Maechler, ETH Zurich; R Core Team

May 15, 2018 @ eRum Budapest

useR! 2004

Users Become Developers

- Good Programming Practice, by Martin Mächler
- Language Interfaces (.Call & .External), by Peter Dalgaard
- Packaging, Documentation, Testing, by Kurt Hornik

What I find important when R Programming

and Recent Cool Features in R

Martin Maechler, ETH Zurich; R Core Team

May 15, 2018 @ eRum Budapest

EARL CONFERENCE

IASC NZSA 2017

useR!

Workflow: You should have one

Zen And The aRt Of Workflow Maintenance

Code Smell and Feels

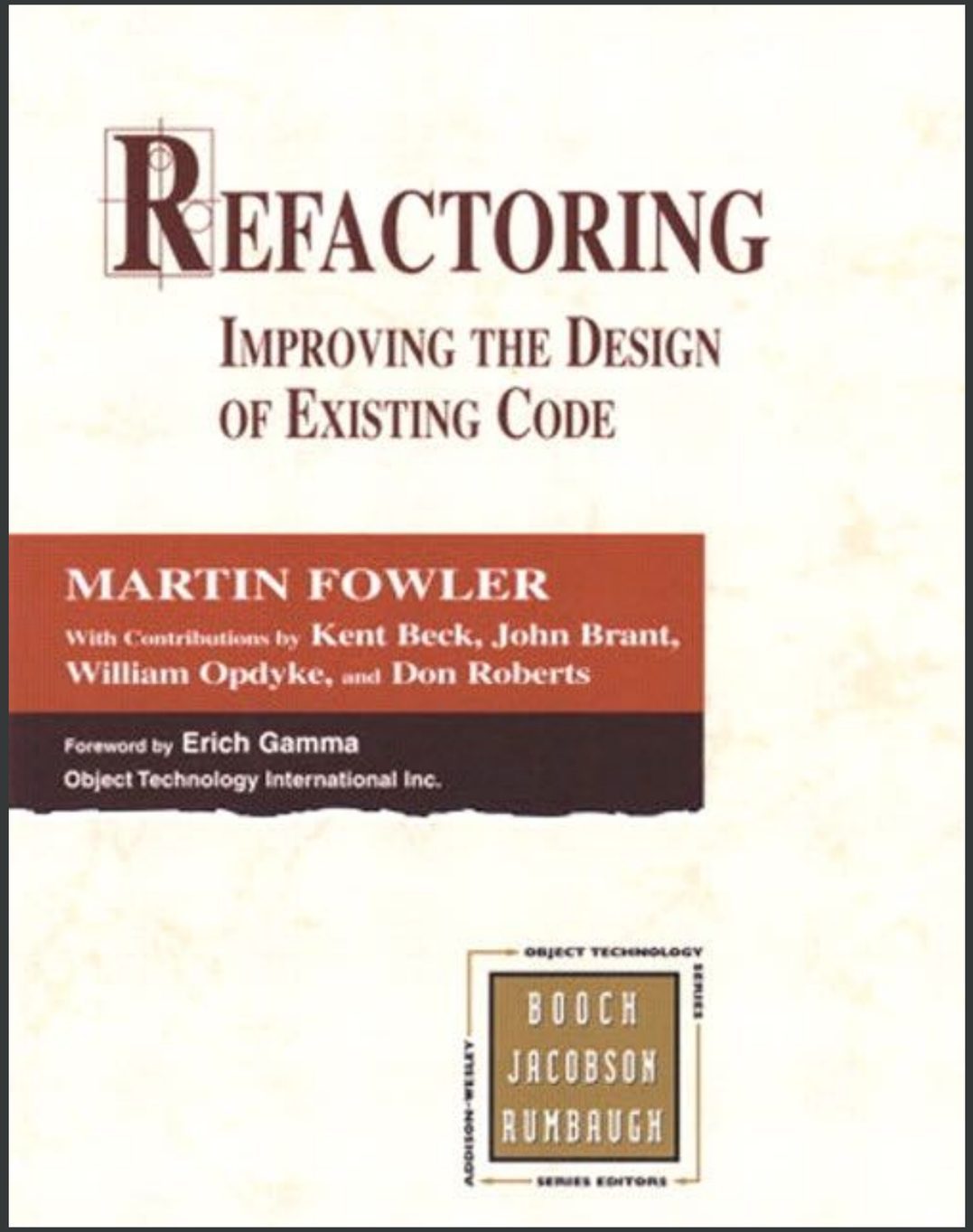
via boredpanda, bbc, reddit

Your taste develops faster than your ability.



Why does my code not smell like theirs?


code smell


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY

code smell

structures in code that suggest
(or scream for)
refactoring

refactor

make code

- easier to understand
- cheaper to modify

without changing behaviour

"Never use `attach()`."

"Always put a space before and after `=`."

"Have better taste."

"Write more elegant code."

"Never use attach()."
"Always put a space before and after =."

Detect a code smell.
Apply the prescribed refactoring.

"Have better taste."
"Write more elegant code."

code smell or ...?

romance novel

legal instrument

quilt pattern

movie franchise

Prince song

Too Hot to Handle
Primitive Obsession
Forbidden Fruit Tree
When Doves Cry
Inappropriate Intimacy
Fast and Furious
Restraining Order
Middle Man

code smell
romance novel
legal instrument
quilt pattern
Prince song
movie franchise

Too Hot to Handle	romance novel
Primitive Obsession	**code smell**
Forbidden Fruit Tree	quilt pattern
When Doves Cry	Prince song
Inappropriate Intimacy	**code smell**
Fast and Furious	movie franchise
Restraining Order	legal instrument
Middle Man	**code smell**

Too Hot to Handle	new code smell?
Forbidden Fruit Tree	new code smell?
When Doves Cry	new code smell?
Fast and Furious	new code smell?
Restraining Order	new code smell?

Duplicated Code
Long Method
Large Class
Long Parameter List
Divergent Change
Shotgun Surgery
Feature Envy
Data Clumps
Primitive Obsession
Switch Statements
Parallel Inheritance Hierarchies
Lazy Class
Speculative Generality
Temporary Field
Message Chains
Middle Man
Inappropriate Intimacy
Alt. Classes w/ Diff. Interfaces
Incomplete Library Class
Data Class
Refused Bequest
Comments

Refactoring, Chapter 9: Simplifying Conditional Expressions


bizarro

all code snippets are here:

rstd.io/code-smells

x	bizarro(x)
-77, 0, 4	77, 0, -4
TRUE, FALSE	FALSE, TRUE
"desserts", "god"	"stressed", "dog"

```r
x <- 1:5
#x <- c(TRUE, FALSE, FALSE, TRUE, FALSE)

cat(
  "The bizarro version of x is",
  -x,
  #!x,
  "\n"
)
#> The bizarro version of x is -1 -2 -3 -4 -5
```

```r
#x <- 1:5
x <- c(TRUE, FALSE, FALSE, TRUE, FALSE)

cat(
  "The bizarro version of x is",
  #-x,
  !x,
  "\n"
)
#> The bizarro version of x is FALSE TRUE TRUE FALSE TRUE
```

Tip #1:
Do not comment and uncomment sections of code to alter behaviour.

Tip #1:
Do not comment and uncomment sections of code to alter behaviour.

Tip #1a:
Especially not in multiple places that you will never, ever keep track of 😉

```r
x <- 1:5
#x <- c(TRUE, FALSE, FALSE, TRUE, FALSE)

cat(
  "The bizarro version of x is",
  if (is.numeric(x)) {
    -x
  } else {
    !x
  },
  "\n"
)
#> The bizarro version of x is -1 -2 -3 -4 -5
```

Tip #2:

Use `if () else ()` in moderation.

Tip #2:
Use `if () else ()` in moderation.

Tip #2a:
Describe in grandiose terms:
"I used a one layer neural network with identity activation and no hidden layers." *-- Federico Vaggi*

```r
bizarro <- function(x) {
  if (is.numeric(x)) {
    -x
  } else {
    !x
  }
}
```

```r
bizarro(1:5)
#> [1] -1 -2 -3 -4 -5
```

```r
bizarro(c(TRUE, FALSE, FALSE, TRUE, FALSE))
#> [1] FALSE  TRUE  TRUE FALSE  TRUE
```

Tip #3:
Use functions.

Tip #3:

Use functions.

Tip #3a:

A few little functions >> a monster function.

Tip #3:
Use functions.

Tip #3a:
A few little functions >> a monster function.

Tip #3b:
Small well-named helper >> commented code

```
bizarro <- function(x) {
  if (class(x)[[1]] == "numeric" || class(x)[[1]] == "integer") {
    -x
  } else if (class(x)[[1]] == "logical") {
    !x
  } else { stop(...) }
}

bizarro(c(TRUE, FALSE, FALSE, TRUE, FALSE))
#> [1] FALSE  TRUE  TRUE FALSE  TRUE

bizarro(c("abc", "def"))
#> Error: Don't know how to make bizzaro <character>
```

```r
bizarro <- function(x) {
  if (is.numeric(x)) {
    -x
  } else if (is.logical(x)) {
    !x
  } else { stop(...) }
}

bizarro(1:5)
#> [1] -1 -2 -3 -4 -5

bizarro(c("abc", "def"))
#> Error: Don't know how to make bizzaro <character>
```

Tip #4:
Use proper functions for handling class & type.
Use simple conditions.

Extract fussy condition logic into well-named function.

from googledrive

```
drive_cp <- function(file, ...) {
  ...
  if (is_parental(file)) {
    stop_glue("The Drive API does not copy folders or Team Drives.")
  }
  ...
}
```

```
is_parental <- function(d) {
  stopifnot(inherits(d, "dribble"))
  kind <- purrr::map_chr(d$drive_resource, "kind")
  mime_type <- purrr::map_chr(d$drive_resource, "mimeType", .default = NA)
  kind == "drive#teamDrive" | mime_type == "application/vnd.google-apps.folder"
}
```

```r
bizarro <- function(x) {
  stopifnot(is.numeric(x) || is.logical(x))

  if (is.numeric(x)) {
    -x
  } else {
    !x
  }
}

bizarro(c(TRUE, FALSE, FALSE, TRUE, FALSE))
#> [1] FALSE  TRUE  TRUE FALSE  TRUE

bizarro(1:5)
#> [1] -1 -2 -3 -4 -5

bizarro(c("abc", "def"))
#> Error: is.numeric(x) || is.logical(x) is not TRUE
```

Tip #5:

Hoist quick `stop()`s and `return()`s to the top.

I.e., use a guard clause.

Clarify and emphasize the happy path.

```
get_some_data <- function(config, outfile) {
  if (config_ok(config)) {
    if (can_write(outfile)) {
      if (can_open_network_connection(config)) {
        data <- parse_something_from_network()
        if(makes_sense(data)) {
          data <- beautify(data)
          write_it(data, outfile)
          return(TRUE)
        } else {
          return(FALSE)
        }
      } else {
        stop("Can't access network")
      }
    } else {
      ## uhm. What was this else for again?
    }
  } else {
    ## maybe, some bad news about ... the config?
  }
}
```

```
get_some_data <- function(config, outfile) {
  if (config_bad(config)) {
    stop("Bad config")
  }

  if (!can_write(outfile)) {
    stop("Can't write outfile")
  }

  if (!can_open_network_connection(config)) {
    stop("Can't access network")
  }

  data <- parse_something_from_network()
  if(!makes_sense(data)) {
    return(FALSE)
  }

  data <- beautify(data)
  write_it(data, outfile)
  TRUE
}
```

```r
get_some_data <- function(config, outfile) {
  if (config_ok(config)) {
    if (can_write(outfile)) {
      if (can_open_network_connection(config)) {
        data <- parse_something_from_network()
        if(makes_sense(data)) {
          data <- beautify(data)
          write_it(data, outfile)
          return(TRUE)
        } else {
          return(FALSE)
        }
      } else {
        stop("Can't access network")
      }
    } else {
      ## uhm. What was this else for again?
    }
  } else {
    ## maybe, some bad news about ... the config?
  }
}
```

```r
get_some_data <- function(config, outfile) {
  if (config_bad(config)) {
    stop("Bad config")
  }

  if (!can_write(outfile)) {
    stop("Can't write outfile")
  }

  if (!can_open_network_connection(config)) {
    stop("Can't access network")
  }

  data <- parse_something_from_network()
  if(!makes_sense(data)) {
    return(FALSE)
  }

  data <- beautify(data)
  write_it(data, outfile)
  TRUE
}
```

There is no else, there is only if.

```r
get_some_data <- function(config, outfile) {
  if (config_ok(config)) {
    if (can_write(outfile)) {
      if (can_open_network_connection(config)) {
        data <- parse_something_from_network()
        if(makes_sense(data)) {
          data <- beautify(data)
          write_it(data, outfile)
          return(TRUE)
        } else {
          return(FALSE)
        }
      } else {
        stop("Can't access network")
      }
    } else {
      ## uhm. What was this else for again?
    }
  } else {
    ## maybe, some bad news about ... the config?
  }
}
```

```r
get_some_data <- function(config, outfile) {
  if (config_bad(config)) {
    stop("Bad config")
  }

  if (!can_write(outfile)) {
    stop("Can't write outfile")
  }

  if (!can_open_network_connection(config)) {
    stop("Can't access network")
  }

  data <- parse_something_from_network()
  if(!makes_sense(data)) {
    return(FALSE)
  }

  data <- beautify(data)
  write_it(data, outfile)
  TRUE
}
```

TMI

too much information?

TMI

too much information?

total meat indentation!

```
get_some_data <- function(config, outfile) {
  if (config_ok(config)) {
    if (can_write(outfile)) {
      if (can_open_network_connection(config)) {
        data <- parse_something_from_network()
        if(makes_sense(data)) {
          data <- beautify(data)
          write_it(data, outfile)
          return(TRUE)
        } else {
          return(FALSE)
        }
      } else {
        stop("Can't access network")
      }
    } else {
      ## uhm. What was this else for again?
    }
  } else {
    ## maybe, some bad news about ... the config?
  }
}
```

```
get_some_data <- function(config, outfile) {
  if (config_bad(config)) {
    stop("Bad config")
  }

  if (!can_write(outfile)) {
    stop("Can't write outfile")
  }

  if (!can_open_network_connection(config)) {
    stop("Can't access network")
  }

  data <- parse_something_from_network()
  if(!makes_sense(data)) {
    return(FALSE)
  }

  data <- beautify(data)
  write_it(data, outfile)
  TRUE
}
```


get_some_data <- function(config, outfile) {
if (config_ok(config)) {
if (can_write(outfi
if (can_open_netw
→→→ data <- parse_s
if(makes_sense(
→→→→ data <- beaut
→→→→ write_it(data
→→→→ return(TRUE)
} else {
→→→→ return(FALSE)
}
} else {
stop("Can't acc
}
} else {
uhm. What was
}
} else {
maybe, some bad
}
}
get_some_data <- function(config, outfile) {
if (config_bad(config)) {
le)) {
outfile")
k_connection(config)) {
network")
ing_from_network()
) {
a)
le)
TRUE
}
nested conditionals
Jenny's made-up metric
early exits
17
TMI
1
3
MMI
0

from googledrive

```
process_response <- function(res) {
  if (httr::status_code(res) == 204) {
    return(TRUE)
  }

  if (httr::status_code(res) >= 200 && httr::status_code(res) < 300) {
    return(res %>%
             stop_for_content_type() %>%
             httr::content(as = "parsed", type = "application/json"))
  }

  ## 20+ more lines of error handling ...
}
```

Tip #5c:

An `if` block that ends with `stop()` or `return()` does not require `else`.

Recognize your early exits.

less indentation >> lots of indentation

```r
bizarro <- function(x) {
  if (is.numeric(x)) {
    -x
  } else if (is.logical(x)) {
    !x
  } else if (is.character(x)) {
    str_reverse(x)
  } else if (is.factor(x)) {
    levels(x) <- rev(levels(x))
    x
  } else {
    stop(...)
  }
}
```

```
bizarro <- function(x) {
  if (is.numeric(x)) {
    →-x
  } else if (is.logical(x)) {
    →!x
  } else if (is.character(x)) {
    →str_reverse(x)
  } else if (is.factor(x)) {
    →levels(x) <- rev(levels(x))
    →x
  } else {
    stop(...)
  }
}
```

Tip #6:

If your conditions deal with class, it's time to get object-oriented (OO).

In CS jargon, use polymorphism.

S3 OO system

```
bizarro <- function(x) {
  UseMethod("bizarro")
}

bizarro.default <- function(x) {
  stop(
    "Don't know how to make bizzaro <",
    class(x)[[1]], ">",
    call. = FALSE
  )
}
```

```
bizarro(1:5)
#> Error: Don't know how to make bizzaro <integer>

bizarro(TRUE)
#> Error: Don't know how to make bizzaro <logical>

bizarro("abc")
#> Error: Don't know how to make bizzaro <character>
```

S3 OO system

```
bizarro.numeric <- function(x) -x

bizarro.logical <- function(x) !x

bizarro.character <- function(x) str_reverse(x)

bizarro.factor <- function(x) {
  levels(x) <- rev(levels(x))
  x
}

bizarro.data.frame <- function(x) {
  names(x) <- bizarro(names(x))
  x[] <- lapply(x, bizarro)
  x
}
```

```r
str_reverse <- function(x) {
  vapply(
    strsplit(x, ""),
    FUN = function(z) paste(rev(z), collapse = ""),
    FUN.VALUE = ""
  )
}

str_reverse(c("abc", "def"))
#> [1] "cba" "fed"
```

```
bizarro(1:5)
#> [1] -1 -2 -3 -4 -5

bizarro(c(TRUE, FALSE, FALSE, TRUE, FALSE))
#> [1] FALSE  TRUE  TRUE FALSE  TRUE

bizarro(c("abc", "def"))
#> [1] "cba" "fed"

(m <- factor(month.abb[1:3], levels = month.abb[1:3]))
#> [1] Jan Feb Mar
#> Levels: Jan Feb Mar
bizarro(m)
#> [1] Mar Feb Jan
#> Levels: Mar Feb Jan

bizarro(head(iris, 3))
#>   htgneL.lapeS htdiW.lapeS htgneL.lateP htdiW.lateP   seicepS
#> 1          -5.1         -3.5          -1.4         -0.2 virginica
#> 2          -4.9         -3.0          -1.4         -0.2 virginica
#> 3          -4.7         -3.2          -1.3         -0.2 virginica
```

Tip #6 redux:

Avoid explicit conditionals by creating methods that are specific to an object's class.

```r
bizarro <- function(x) {
  cls <- class(x)[[1]] ## switching on class is 😬
  switch(
    cls,
    logical = !x,
    integer =   ,
    numeric = -x,
    character = str_reverse(x),
    stop(...)
  )
}
```

from stringr

```
str_pad <- function(string,
                    width,
                    side = c("left", "right", "both"),
                    pad = " ") {
  side <- match.arg(side)

  switch(
    side,
    left  =  stri_pad_left(string, width, pad = pad),
    right = stri_pad_right(string, width, pad = pad),
    both  =  stri_pad_both(string, width, pad = pad)
  )
}
```

Tip #7:
`switch()` is ideal if you need to dispatch different logic, based on a string.

Tip #7a: You are allowed to write a helper function to generate that string.

```r
library(tidyverse)

tibble(
  age_yrs = c(0, 4, 10, 15, 24, 55),
  age_cat = case_when(
    age_yrs < 2  ~ "baby",
    age_yrs < 13 ~ "kid",
    age_yrs < 20 ~ "teen",
    TRUE         ~ "adult"
  )
)
#> # A tibble: 6 x 2
#>   age_yrs age_cat
#>     <dbl> <chr>
#> 1       0 baby
#> 2       4 kid
#> 3      10 kid
#> 4      15 teen
#> 5      24 adult
#> 6      55 adult
```

```
library(tidyverse)

tibble(
  age_yrs = c(0, 4, 10, 15, 24, 55),
  age_cat = case_when(
    age_yrs < 2  ~ "baby",
    age_yrs < 13 ~ "kid",
    age_yrs < 20 ~ "teen",
    TRUE         ~ "adult"
  )
)
#> # A tibble: 6 x 2
#>   age_yrs age_cat
#>     <dbl> <chr>
#> 1       0 baby
#> 2       4 kid
#> 3      10 kid
#> 4      15 teen
#> 5      24 adult
#> 6      55 adult
```

alternative to:

```
ifelse(age_yrs < 2, "baby",
       ifelse(age_yrs < 13, "kid",
              ifelse(age_yrs < 20, "teen",
                     "adult"
              )
       )
)
```

Tip #8:

`dplyr::case_when()` is ideal if you need to dispatch different data, based on data (+ logic).

from rlang, purrr

```r
`%||%` <- function(x, y) {
  if (is_null(x)) y else x
}
```

from devtools

```
github_remote <- function(repo, username = NULL, ...) {
  meta <- parse_git_repo(repo)
  ...
  meta$username <- username %||%
    getOption("github.user") %||%
    stop("Unknown username.")
  ...
}
```

Write simple conditions.

Use helper functions.

Handle class properly.

Return and exit early.

polymorphism

`switch()`

`case_when()`

`%||%`

rstd.io/code-smells

@jennybc @JennyBryan