Software quality analysis of fourteen hydrological models

Arnald Puy

Contents

1 Preliminary functions

```r
#   PRELIMINARY FUNCTIONS #####################################################

sensobol::load_packages(c("data.table", "tidyverse", "openxlsx", "scales",
                          "cowplot", "readxl", "ggrepel", "tidytext"))

# Create custom theme -----------------------------------------------------

theme_AP <- function() {
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          legend.background = element_rect(fill = "transparent", color = NA),
          legend.key = element_rect(fill = "transparent", color = NA),
          strip.background = element_rect(fill = "white"),
          legend.text = element_text(size = 7.3),
          axis.title = element_text(size = 10),
          legend.key.width = unit(0.4, "cm"),
          legend.key.height = unit(0.4, "cm"),
          legend.key.spacing.y = unit(0, "lines"),
          legend.box.spacing = unit(0, "pt"),
          legend.title = element_text(size = 7.3),
          axis.text.x = element_text(size = 7),
          axis.text.y = element_text(size = 7),
          axis.title.x = element_text(size = 7.3),
          axis.title.y = element_text(size = 7.3),
          plot.title = element_text(size = 8),
          strip.text.x = element_text(size = 7.4),
          strip.text.y = element_text(size = 7.4))
}

# Select color palette ----------------------------------------------------

color_languages <- c("fortran" = "steelblue", "python" = "lightgreen")
```

2 Results

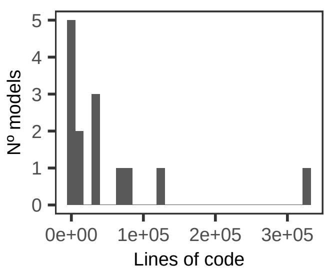
```r
# READ IN DATASET #############################################################

# Get name of sheets ------------------------------------------------------

sheets <- excel_sheets("./datasets/results_sqa.xlsx")

# Read all sheets ---------------------------------------------------------
```

```r
dt <- lapply(sheets, function(x) data.table(read_excel("./datasets/results_sqa.xlsx",
                                                        sheet = x)))

# Name the slots ---------------------------------------------------------------

names(dt) <- sheets
```

2.1 Descriptive statistics

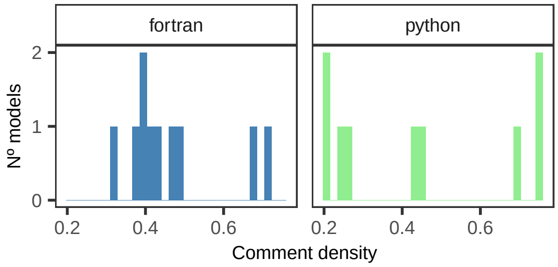
```r
# PLOT LINES OF CODE ###########################################################

plot_lines_code <- dt$descriptive_stats[, .(total_lines_code = sum(lines_code)), model] %>%
  ggplot(., aes(total_lines_code)) +
  geom_histogram() +
  labs(x = "Lines of code", y = "Nº models") +
  theme_AP()

plot_lines_code
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```r
# PLOT COMMENT DENSITY #########################################################

plot_comment_density <- dt$descriptive_stats[, .(total_lines_code = sum(lines_code),
       total_lines_comments = sum(lines_comments)), .(model, language)] %>%
  .[, comment_density:= total_lines_comments / total_lines_code] %>%
  ggplot(., aes(comment_density, fill = language)) +
  geom_histogram() +
  facet_wrap(~language) +
  scale_y_continuous(breaks = breaks_pretty(n = 3)) +
  scale_fill_manual(values = color_languages) +
  labs(x = "Comment density", y = "Nº models") +
  theme_AP() +
  theme(legend.position = "none")

plot_comment_density
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



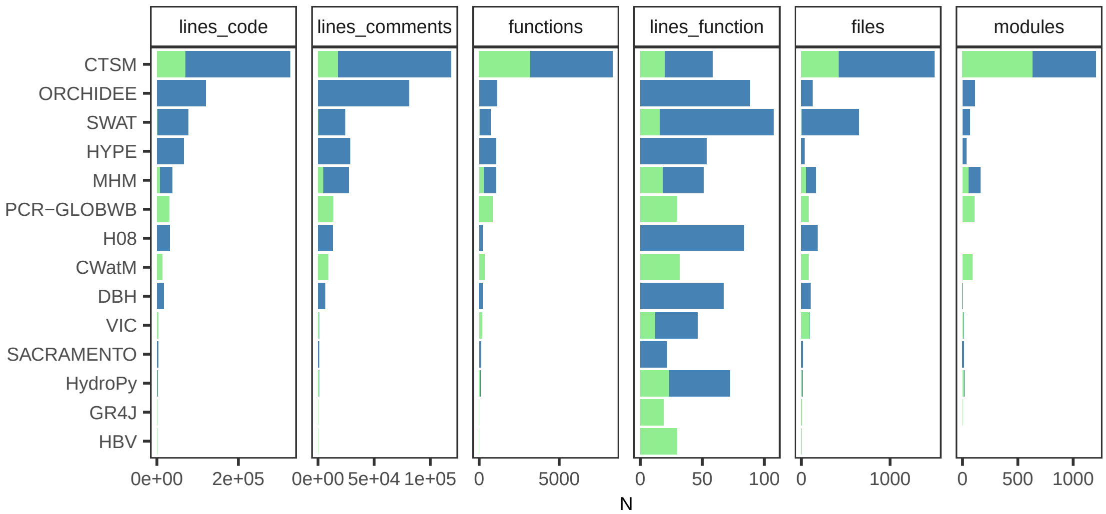

```
# PLOT PER MODEL ################################################################

# Sort by model ----------------------------------------------------------------

model_ordered <- dt$descriptive_stats[, sum(lines), model] %>%
  .[order(V1)]

# Print ------------------------------------------------------------------------

model_ordered
```

```
##          model     V1
##         <char>  <num>
##  1:        HBV    180
##  2:       GR4J    423
##  3:    HydroPy   3739
##  4: SACRAMENTO   5294
##  5:        VIC   5952
##  6:        DBH  24334
##  7:      CWatM  27745
##  8:        H08  42917
##  9: PCR-GLOBWB  52686
## 10:        MHM  76286
## 11:       HYPE  89137
## 12:       SWAT  99976
## 13:   ORCHIDEE 211871
## 14:       CTSM 491592
```

```
# Extract column names ---------------------------------------------------------

col_names <- colnames(dt$descriptive_stats)

# Order facets -----------------------------------------------------------------

facet_order <- c("lines", "lines_code", "lines_comments", "functions",
                 "lines_function", "files", "modules")



# Plot -------------------------------------------------------------------------
```

```r
plot_per_model <- melt(dt$descriptive_stats, measure.vars = col_names[-c(1, length(col_names))]
  .[, variable:= factor(variable, levels = facet_order)] %>%
  .[, model:= factor(model, levels = model_ordered[, model])] %>%
  .[!variable == "lines"] %>%
  ggplot(., aes(model, value, fill = language)) +
  geom_col() +
  coord_flip() +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  scale_fill_manual(values = color_languages) +
  facet_wrap(~ variable, ncol = 7, scales = "free_x") +
  labs(x = "", y = "N") +
  theme_AP() +
  theme(legend.position = "none")

plot_per_model
```

```
## Warning: Removed 3 rows containing missing values or values outside the scale range
## (`geom_col()`).
```



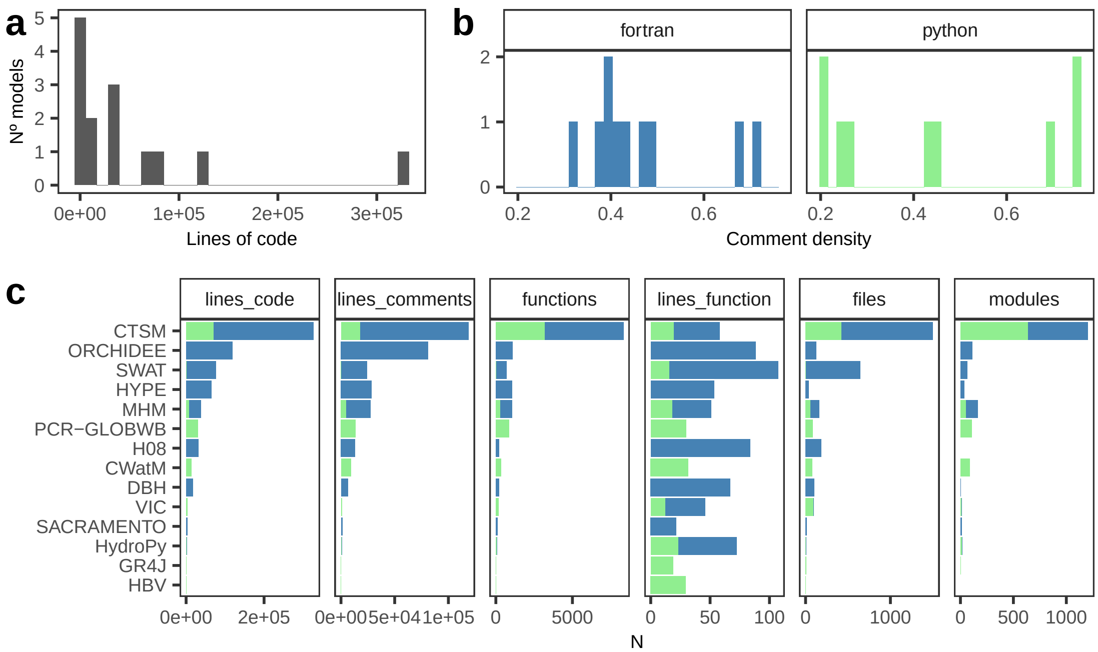

```r
# MERGE PLOTS ##################################################################

top <- plot_grid(plot_lines_code, plot_comment_density + labs(x = "Comment density", y  = ""),
          labels = "auto", rel_widths = c(0.4, 0.6))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```r
p1 <- plot_grid(top, plot_per_model, ncol = 1, labels = c("", "c"), rel_heights = c(0.4, 0.6))
```

```
## Warning: Removed 3 rows containing missing values or values outside the scale range
## (`geom_col()`).
```

```r
p1
```



2.2 Maintainability index

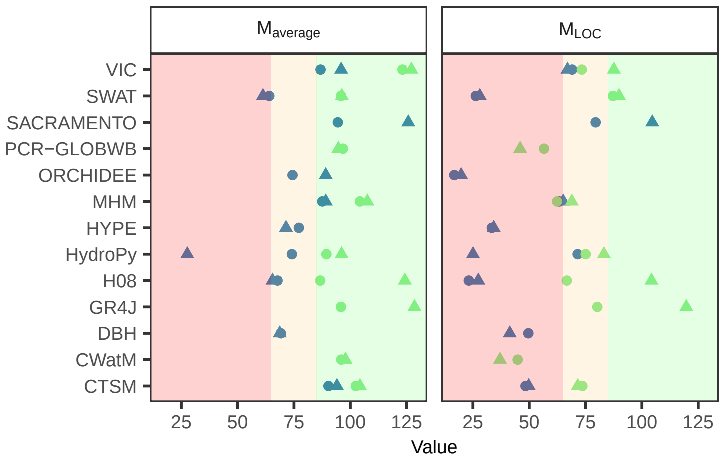
```r
# CALCULATE INTERPRETATIBILITY OF MAINTAINABILITY INDEX 3#######################

# Define vector of interpretation ---------------------------------------------

vec_interpretation <- c("low", "moderate", "high")

# Calculate -------------------------------------------------------------------

dt$maintainability_index %>%
  melt(., measure.vars = c("M_loc", "M_average")) %>%
  .[, interpretativity:= ifelse(value > 85, vec_interpretation[3],
                                ifelse(value <=85 & value >= 65, vec_interpretation[2],
                                       vec_interpretation[1]))] %>%
  .[, .N, .(language, interpretativity, variable)] %>%
  dcast(., variable + language ~ interpretativity, value.var = "N") %>%
  .[, total:= rowSums(.SD, na.rm = TRUE), .SDcols = vec_interpretation] %>%
  .[, paste(vec_interpretation, "prop", sep = "_"):= lapply(.SD, function(x)
    x / total), .SDcols = vec_interpretation] %>%
  print()
```

```
## Key: <variable, language>
##     variable language  high   low moderate total  low_prop moderate_prop
##       <fctr>   <char> <int> <int>    <int> <num>     <num>         <num>
## 1:     M_loc  fortran     1    14        5    20 0.7000000     0.2500000
## 2:     M_loc   python     5     5        8    18 0.2777778     0.4444444
## 3: M_average  fortran     9     3        8    20 0.1500000     0.4000000
## 4: M_average   python    18    NA       NA    18        NA            NA
```

```
##    high_prop
##        <num>
## 1: 0.0500000
## 2: 0.2777778
## 3: 0.4500000
## 4: 1.0000000
```

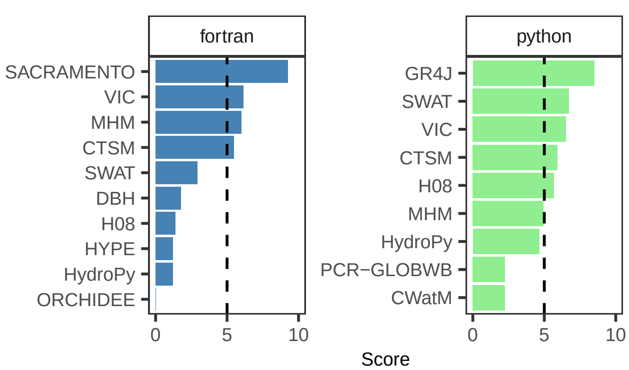
By combining the classic and extended versions of the maintainability index, our analysis reveals differences between Fortran and Python implementations. Using the weighted measure (M_{LOC}), 70% of Fortran code falls into the "low" maintainability category, compared with only 15% when using the unweighted average (M_{average}). This discrepancy indicates that a few complex, poorly maintainable routines dominate the overall profile of the Fortran codebase. In contrast, Python routines present a more favorable profile: 27% achieve high maintainability under M_{LOC}, and all are classified as "highly maintainable" under M_{average}.

```r
# PLOT MAINTAINABILITY INDEX ###################################################

plot_maintainability_index <- dt$maintainability_index %>%
  melt(., measure.vars = c("M_loc", "M_average")) %>%
  .[, variable:= factor(variable, levels = c("M_average", "M_loc"))] %>%
  ggplot(., aes(model, value, color = language, shape = type)) +
  geom_point() +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = 65,
           fill = "red", alpha = 0.18) +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = 65, ymax = 85,
           fill = "orange", alpha = 0.1) +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = 85, ymax = Inf,
           fill = "green", alpha = 0.1) +
  facet_wrap(~variable, labeller = as_labeller(c(M_loc = "M[LOC]",
                                                  M_average = "M[average]"),
                                                default = label_parsed)) +
  labs(x = "", y = "Value") +
  scale_color_manual(values = color_languages, guide = "none") +
  theme_AP() +
  theme(legend.position = "none") +
  coord_flip()

plot_maintainability_index
```

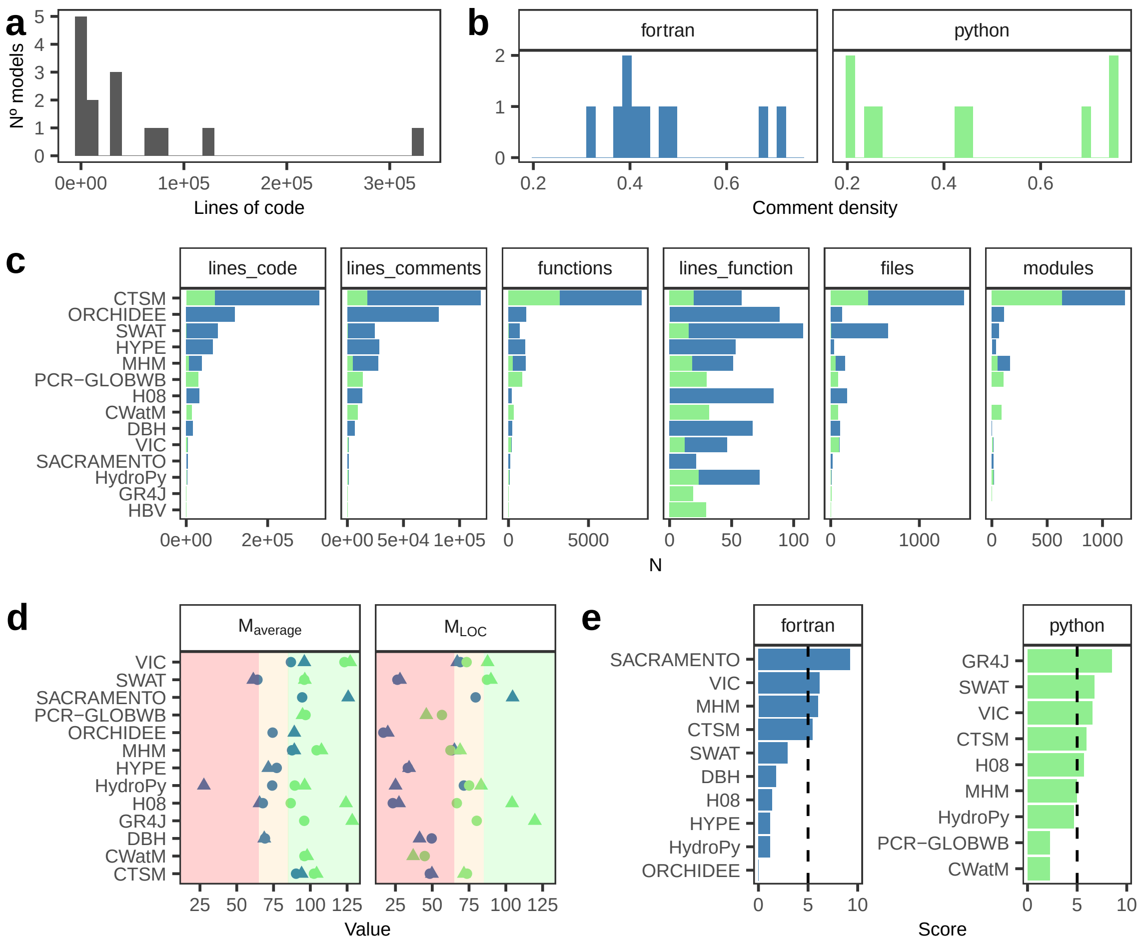


2.3 Score

```r
# PLOT SCORE ###################################################################

plot_score <- dt$score %>%
  ggplot(aes(x = reorder_within(model, score, language), y = score, fill = language)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ language, scales = "free_y") +
  labs(x = "", y = "Score") +
  scale_fill_manual(values = color_languages) +
  geom_hline(yintercept = 5, lty = 2) +
  scale_x_reordered() +
  scale_y_continuous(limits = c(0, 10), breaks = c(0, 5, 10)) +
  coord_flip() +
  theme_AP() +
  theme(legend.position = "none")

plot_score
```




```r
# MERGE PLOTS ##################################################################

bottom <- plot_grid(plot_maintainability_index, plot_score, ncol = 2, labels = c("d", "e"))

plot_grid(p1, bottom, ncol = 1, rel_heights = c(0.62, 0.38))
```



2.4 Metrics at the function level

```r
# METRICS AT THE FILE AND FUNCTION LEVEL #########################################

folder <- "./datasets/results_function"

# Get names of files ------------------------------------------------------------

csv_files <- list.files(path = folder, pattern = "\\.csv$", full.names = TRUE)

# Split into file_metrics and func_metrics -----------------------------------

file_metric_files <- grep("file_metrics", csv_files, value = TRUE)
func_metric_files <- grep("func_metrics", csv_files, value = TRUE)

# Build one named list --------------------------------------------------------

list_metrics <- list(file_metrics = setNames(lapply(file_metric_files, fread),
                                             basename(file_metric_files)),
                     func_metrics = setNames(lapply(func_metric_files, fread),
```

```r
                                          basename(func_metric_files)))

# Create function to combine files ------------------------------------------

make_combined <- function(subset_list, pattern) {
  rbindlist(subset_list[grep(pattern, names(subset_list))], idcol = "source_file")
}

# Combine files ------------------------------------------------------------

metrics_combined <- list(file_fortran = make_combined(list_metrics$file_metrics,  "fortran"),
                         file_python = make_combined(list_metrics$file_metrics,  "python"),
                         func_fortran = make_combined(list_metrics$func_metrics, "fortran"),
                         func_python = make_combined(list_metrics$func_metrics, "python"))

# Functions to extract name of model and language from file --------------------

extract_model <- function(x)
  sub("^(file|func)_metrics_\\d+_([A-Za-z0-9-]+)_(fortran|python).*", "\\2", x)

extract_lang  <- function(x)
  sub("^(file|func)_metrics_\\d+_([A-Za-z0-9-]+)_(fortran|python).*", "\\3", x)

# Extract name of model and language -------------------------------------------

metrics_combined <- lapply(metrics_combined, function(dt) {
  dt[, source_file:= sub("\\.csv$", "", basename(source_file))]
  dt[, model:= extract_model(source_file)]
  dt[, language:= extract_lang(source_file)]
  dt
})

# Add column of complexity category ---------------------------------------------

metrics_combined <- lapply(names(metrics_combined), function(nm) {
  dt <- as.data.table(metrics_combined[[nm]])
  if (grepl("^func_", nm) && "cyclomatic_complexity" %in% names(dt)) {
    dt[, complexity_category := cut(
      cyclomatic_complexity,
      breaks = c(-Inf, 10, 20, 50, Inf),
      labels = c("b1","b2","b3","b4")
    )]
  }
  dt
}) |> setNames(names(metrics_combined))

# Define labels ------------------------------------------------------------
```

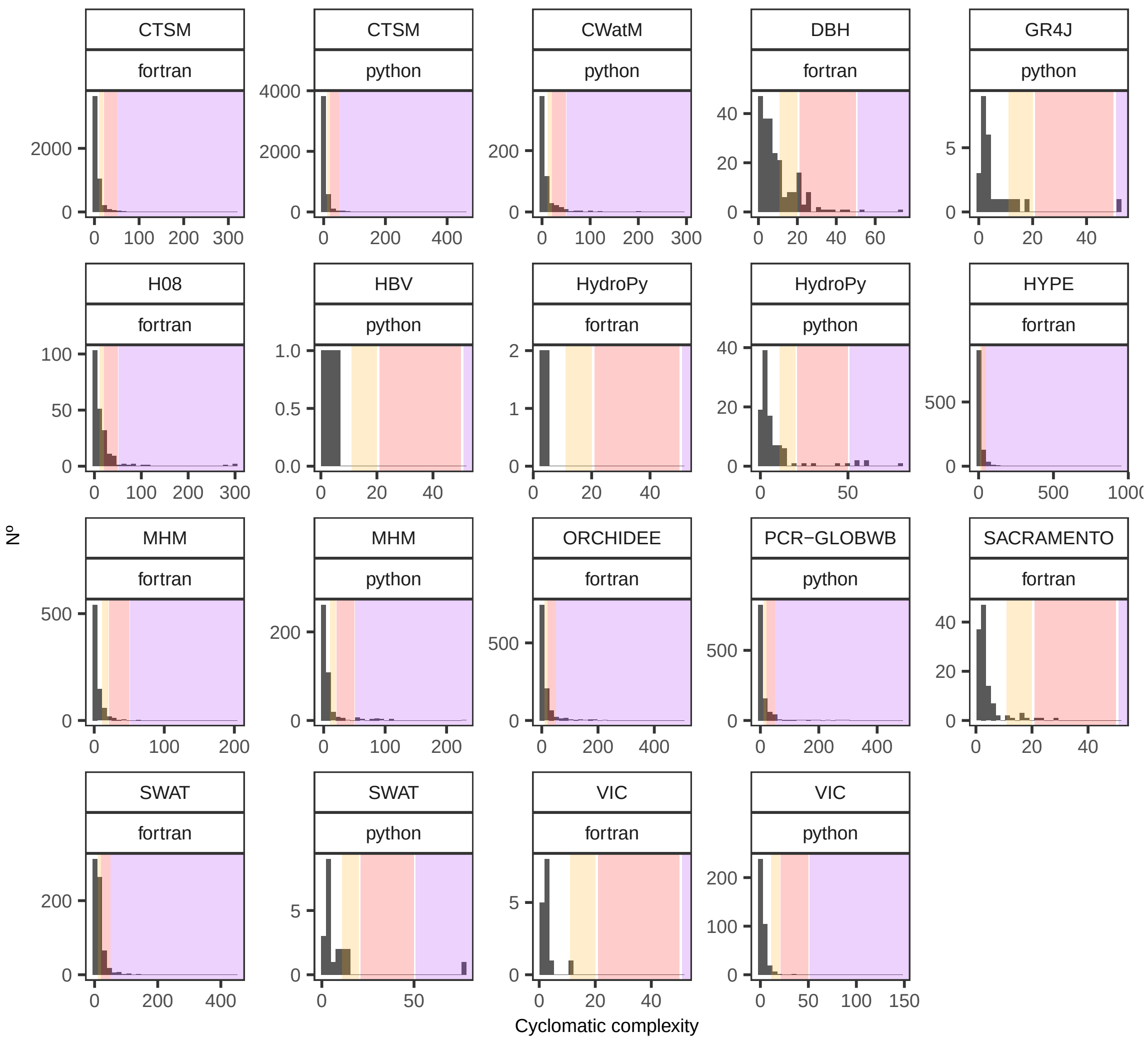
```r
lab_expr <- c(
  b1 = expression(C %in% "(" * 0 * ", 10" * "]"),
  b2 = expression(C %in% "(" * 10 * ", 20" * "]"),
  b3 = expression(C %in% "(" * 20 * ", 50" * "]"),
  b4 = expression(C %in% "(" * 50 * ", " * infinity * ")")
)

# EXPORT DATA TO .CSV ##########################################################

# set output folder inside "datasets" ------------------------------------------

outdir <- file.path("datasets", "merged_results")

# write each slot to its own CSV -----------------------------------------------

lapply(names(metrics_combined), function(nm) {
  out_file <- file.path(outdir, paste0(nm, ".csv"))
  fwrite(metrics_combined[[nm]], out_file)
})
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
```

```r
# PLOT #########################################################################

# Cyclomatic complexity at the model level -------------------------------------

metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(cyclomatic_complexity, model, language)]) %>%
  rbindlist() %>%
  ggplot(., aes(cyclomatic_complexity)) +
  geom_histogram() +
  annotate("rect",
           xmin = 11, xmax = 20,
           ymin = -Inf, ymax = Inf,
           fill = "orange", alpha = 0.2) +
  annotate("rect",
           xmin = 21, xmax = 50,
```

```r
           ymin = -Inf, ymax = Inf,
           fill = "red", alpha = 0.2) +
  annotate("rect",
           xmin = 51, xmax = Inf,
           ymin = -Inf, ymax = Inf,
           fill = "purple", alpha = 0.2) +
  facet_wrap(model ~ language, scales = "free") +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  scale_y_continuous(breaks = breaks_pretty(n = 2)) +
  labs(x = "Cyclomatic complexity", y = "Nº") +
  theme_AP()
```

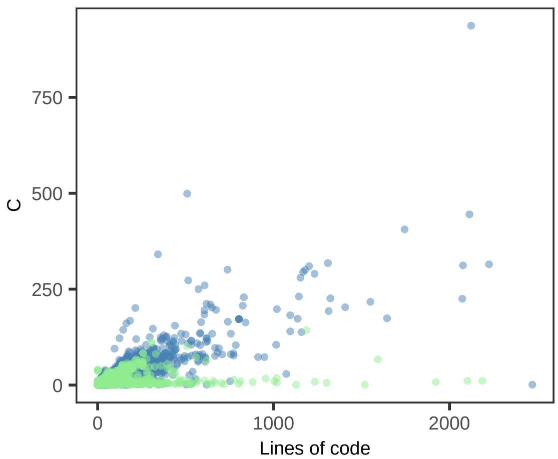
```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```



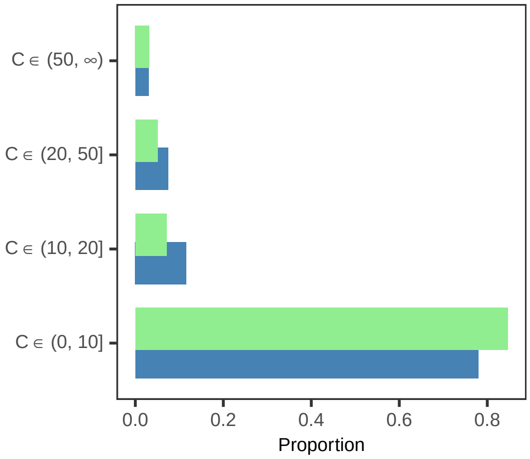

```r
# Scatterplot cyclomatic vs lines of code ---------------------------------------

plot_c_vs_loc <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(loc, cyclomatic_complexity, language)]) %>%
  rbindlist() %>%
```

```r
  ggplot(., aes(loc, cyclomatic_complexity, color = language)) +
  geom_point(alpha = 0.5, size = 0.7) +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = "Lines of code", y = "C") +
  scale_color_manual(values = color_languages) +
  theme_AP() +
  theme(legend.position = "none")

plot_c_vs_loc
```

```
## Warning: Removed 1195 rows containing missing values or values outside the scale range
## (`geom_point()`).
```




```r
# Count & proportion ----------------------------------------------------------

plot_bar_cyclomatic <- metrics_combined[grep("^func_", names(metrics_combined))] %>%
  lapply(., function(x) x[, .(complexity_category, language)]) %>%
  rbindlist() %>%
  .[, .N, .(complexity_category, language)] %>%
  .[, proportion := N / sum(N), language] %>%
  ggplot(., aes(complexity_category, proportion, fill = language)) +
  geom_bar(stat = "identity", position = position_dodge(0.6)) +
  scale_fill_manual(values = color_languages) +
  scale_y_continuous(breaks = scales::breaks_pretty(n = 4)) +
  scale_x_discrete(labels = lab_expr) +
  labs(x = "", y = "Proportion") +
  coord_flip() +
  theme_AP() +
  theme(legend.position = "none")

plot_bar_cyclomatic
```




```
# MERGE ########################################################################

plot_grid(plot_c_vs_loc, plot_bar_cyclomatic, ncol = 2, labels = "auto",
          rel_widths = c(0.45, 0.55))
```

```
## Warning: Removed 1195 rows containing missing values or values outside the scale range
## (`geom_point()`).
```