A Computational Analysis of the Dynamics of R Style Based on 94 Million Lines of Code from All CRAN Packages in the Past 20 Years

Chia-Yi Yen, Mia Huai-Wen Chang, Chung-hong Chan

04-07-2019

Please cite this as: Yen, C.Y., Chang, M.H.W., Chan, C.H. (2019) *A Computational Analysis of the Dynamics of R Style Based on 94 Million Lines of Code from All CRAN Packages in the Past 20 Years.* Paper presented at the useR! 2019 conference, Toulouse, France.

The full paper is pending.

Five-sentence Summary

- There are so many programming style variations. R is particularly problematic.
- We have analyzed 94 Million Lines of R code and quantified the evolution in popularities of many style-elements.
- We attribute 3 main factors that drive changes in programming style: effect of style-guides, effect of introducing new features and effect of editors.
- We have identified communitity-specific programming style variations. For example, there are communities which do not use snake_case at all!
- A consensus in programming style is forming. We have summarised it into a **Consensus-based Style**.

Introduction

R is flexible. For example, one can use <- or = as the assignment operator. The following two functions can both be correctly evaluated by R.

```
sum_of_square <- function(x) {
    return(sum(x^2))
}
```

```
sum_of_square = function(x) {
    return(sum(x^2))
}
```

One area that can highlight this flexible is the naming conventions. According to the previous research by Bååth (2012), there are at least 6 styles and none of the 6 has dominated the scene. There are still some other style-elements that R programmers have freedom to adopt, e.g. whether or not to add spaces around infix operators, use double quotation marks or single quotation marks to denote strings, etc. On one hand, these variations provide programmers with freedom. On the other hand, these variations can confuse new programmers and can have dire effects on programe comprehension. Also, incompatibility between programming styles might also affect reuse, maintainability (Elish & Offutt, 2002) and open source collabration (Wang & Hahn, 2017).

Various efforts to standardize the programming style (e.g. Google's R Style Guide, The Tidyverse Style Guide, Bioconductor Coding Style, etc.) are available. These style guides are based on normative assessment of code quality, e.g. style-elements that improve programe comprehension (Oman & Cook, 1990). However,

we argue that we should first study the current situation, and preferably, the historical development, of programming style variations (PSV) to supplement these standardization efforts. We have undertaken such a task, so that the larger R community can have a baseline to evaluate the effectiveness of those standardization efforts. Also, we can have better understanding on the factors driving increase and decrease in PSV historically, such that more effective efforts can be formulated.

Analysis

Data Source

In January 2019, we cloned a local mirror of CRAN using the rsync method suggested by the CRAN Mirror HOWTO. In our local mirror, it contains all package as tarball files (.tar.gz). By all package, we mean all packages actively listed online on the CRAN websites and all packages delisted for whatever reasons e.g. no maintainer, etc. In this analysis, we include all packages, including those delisted.

In order to faciliate the analysis, we have developed the baaugwo package to extract all R sourcecode and metadata from these tarballs. In this study, only the source code from the */R* directory of each tarball file is included. We have also archived the meta data from the DESCRIPTION and NAMESPACE files from these tarballs.

In order to cancel out the effect of multiple submissions in a year by one particular package, we have applied the "one-submission-per-year" rule to randomly selected only one submission from a year for each package. Unless explicitly notice, we present below the analysis of this "one-submission-per-year" sample. Similarly, unless explicitly notice, the unit of the analysis is **exported function**. The study period of this study is from 1998 to 2018.

Quantification of PSV

Every function in our sample are parsed into a parse tree (or expression) using the parser from the lintr package.

These parse trees were then filtered for function definitions and then linted using the linters from the lintr package to detect for specific style-elements. Style-elements considered in this study are:

- **fx_assign**: Use = as assignment operators

```
softplusFunc = function(value, leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L, value, value * 0.01))
    }
    return(log(1L + exp(value)))
}
```

- **fx_opencurly**: An open curly is on its own line

```
softplusFunc <- function(value, leaky = FALSE)
    {
    if (leaky)
        {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L, value, value * 0.01))
    }
```

```
    return(log(1L + exp(value)))
}
```

- **fx__infix**: No spaces are added around infix operators.

```
softplusFunc<-function(value, leaky=FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value>0L, value, value*0.01))
    }
    return(log(1L+exp(value)))
}
```

- **fx__integer**: Not explicitly type integers

```
softplusFunc <- function(value, leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0, value, value * 0.01))
    }
    return(log(1 + exp(value)))
}
```

- **fx__singleq**: Use single quotation marks for strings

```
softplusFunc <- function(value, leaky = FALSE) {
    if (leaky) {
        warnings('using leaky RELU!')
        return(ifelse(value > 0L, value, value * 0.01))
    }
    return(log(1L + exp(value)))
}
```

- **fx__commas**: No spaces are added around commas

```
softplusFunc <- function(value,leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L,value,value * 0.01))
    }
    return(log(1L + exp(value)))
}
```

- **fx__semi**: Use semicolons to terminate lines

```
softplusFunc <- function(value, leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!");
        return(ifelse(value > 0L, value, value * 0.01));
    }
    return(log(1L + exp(value)));
}
```

- **fx__t__f**: Use T/F instead of TRUE / FALSE

```
softplusFunc <- function(value, leaky = F) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L, value, value * 0.01))
    }
    return(log(1L + exp(value)))
}
```

- **fx__closecurly**: An close curly is not on its own line.

```
softplusFunc <- function(value, leaky = F) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L, value, value * 0.01)) }
    return(log(1L + exp(value))) }
```

- **fx__tab**: Use tab to indent

```
softplusFunc <- function(value, leaky = FALSE) {
    if (leaky) {
        warnings("using leaky RELU!")
        return(ifelse(value > 0L, value, value * 0.01))
    }
    return(log(1L + exp(value)))
}
```

We have studied also the naming conventions of all included functions. Using the similar technique of Bååth (2012), we classified function names into the following 7 categories:

- **alllowercase**: softplusfunc
- **ALLUPPERCASE**: SOFTPLUSFUNC
- **UpperCamelCase**: SoftPlusFunc
- **lowerCamelCase**: softPlusFunc
- **snake__case**: soft__plus__func
- **dotted.case**: soft.plus.func
- **other**: sOfTPluSfunc

The last style-element is the line-length. For each R file, we counted the distribution of line-length. In this analysis, the unit of it is line.

By not considering line-length, we have studied 10 binary style-elements and one multinomial style-element with 7 categories. Therefore, the possible number of combinations based on these 11 style-elements is: 7 * 2^10 = 7168.

Community-specific variations

On top of the overall patterns based on the analysis all functions, the community-specific variations are also studied. In this part of the study, we ask the question: do local patterns of PSV exist in programming communities? To this end, we constructed a dependency graph of CRAN packages by defining a package

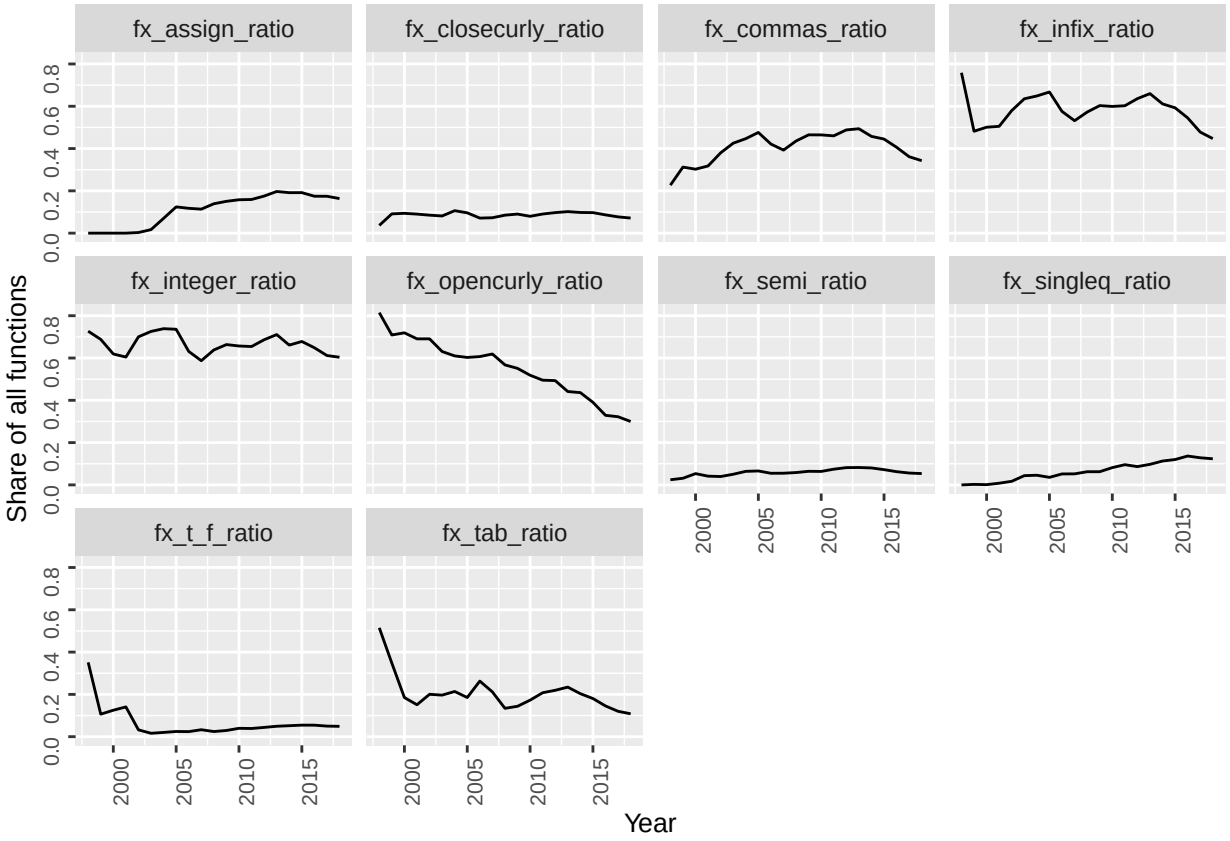

Figure 1: Evolution in popularities of 10 binary style-elements from 1998 to 2018.

as a node and a import/suggest relationship as a directed edge. Communities in this dependency graph were extracted using the Walktrap Community Detection Algorithm (Pons & Latapy, 2005) provided by the igraph package.

The 18 largest communities were extracted to study local patterns in PSV.

Results

We studied more than 94 Million lines of code from 15530 unique packages. In total, 1898142 exported functions were studied. Figure 1 displays the evolution of the 10 binary style-elements from 1998 to 2008. Some style-elements have a very clear trends towards an majority-vs-minority pattern, e.g. fx_closecurly, fx_semi, fx_t_f and fx_tab. Some styles-elements are instead trending towards a divergence from a previously majority-vs-minority pattern, e.g. fx_assign, fx_commas, fx_infix, fx_integer, fx_opencurly and fx_singleq. There are two style-elements that deserve specical scrutiny. Firstly, the variation in fx_assign is a clear example illustrating the effect of introducing a new language element by the R Development Core Team. The introduction of the language feature (= as assignment operator) in R 1.4 (Chambers, 2001) conicides with the taking off in popularity of such style-element since 2001. Up to now, around 20% of exported functions use such style.

Secondly, the popularity of fx_opencurly shows how a previously established majority style (~80% in late 90s) slowly reduced into a minority, but still very prominant, style (~30% in late 10s).

Similarly, the evolution of different naming conventions is shown in figure 2. This analysis can best be used to illustrate the effect of style-guides. According to Bååth(2012), dotted style is a unique naming convention

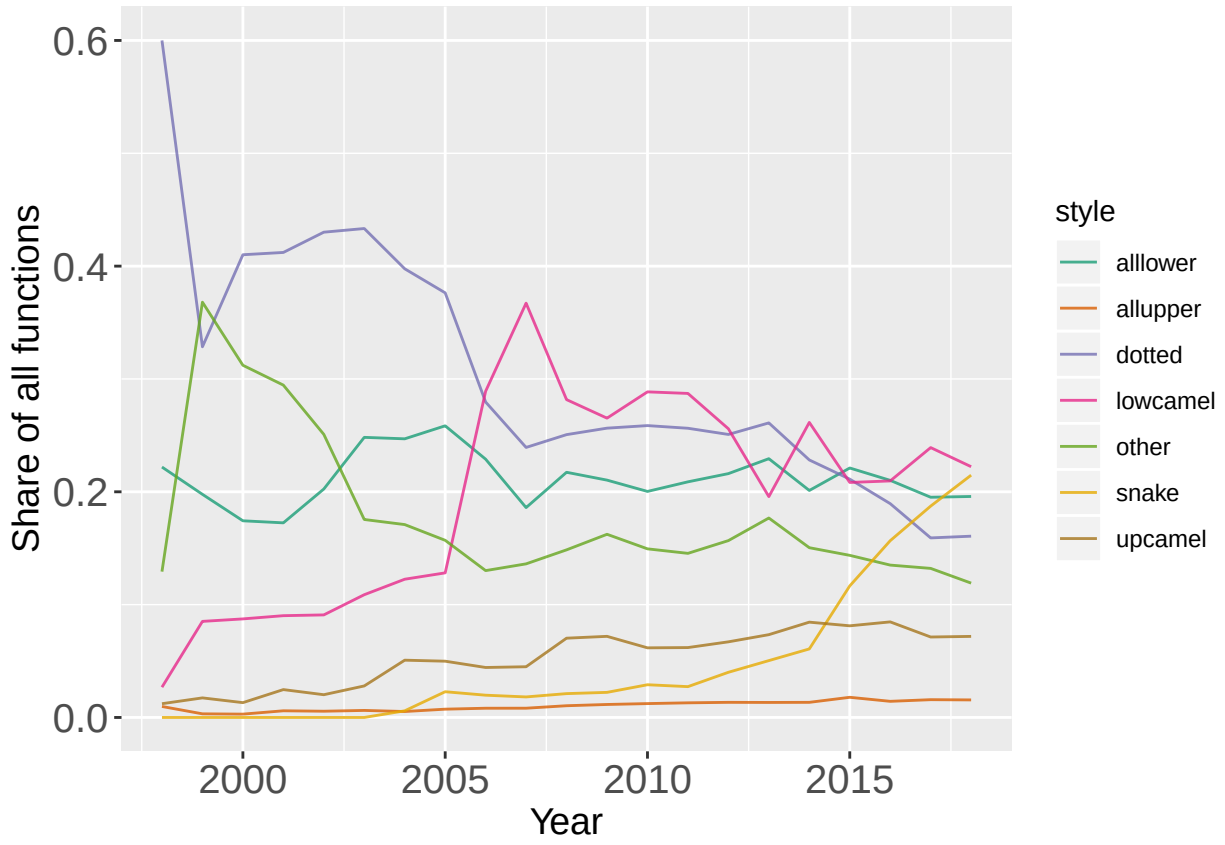

Figure 2: Evolution in popularities of 7 naming conventions from 1998 to 2018.

that is very specific to R programming. This style is the dominant style in the early days of CRAN. However, multiple style guides advise universially against the use of dotted style and thus, a significant decline trend is observed. snake_case and UpperCamelCase are the styles endorsed by the Tidyverse Style Guide and the Google's R Style Guide respectively. These two styles see an increasing trend since 10s, although the growth of snake_case is relatively more explosive. To our surprise, lowerCamelCase, a style not endorsed by any style-guide, is currently the most popular naming convention (22.5% in 2018). However, its reign might soon be dethroned by snake_case (21.5% in 2018) in the near future.

The evolution of line lengths is tricky to be visualised in 2-D space. We have prepared an animation to visualise the change in line distribution over the span of 20 years. In this paper, figure 3 shows the snapshot of the change in line distribution from 40 characters to 100 characters. In general, programmers writes with lesser number of characters per line. Similar to previous analyses with Python programs (VanderPlas, 2017), artificial peaks corresponding to recommendations from either style-guides, linters and editor settings are observed in our analysis. In the most 2018, the artificial peak of 80 characters (recommended by most of the style-guides and linters such as lintr) is more pronounced for lines with comments but not those with actual code.

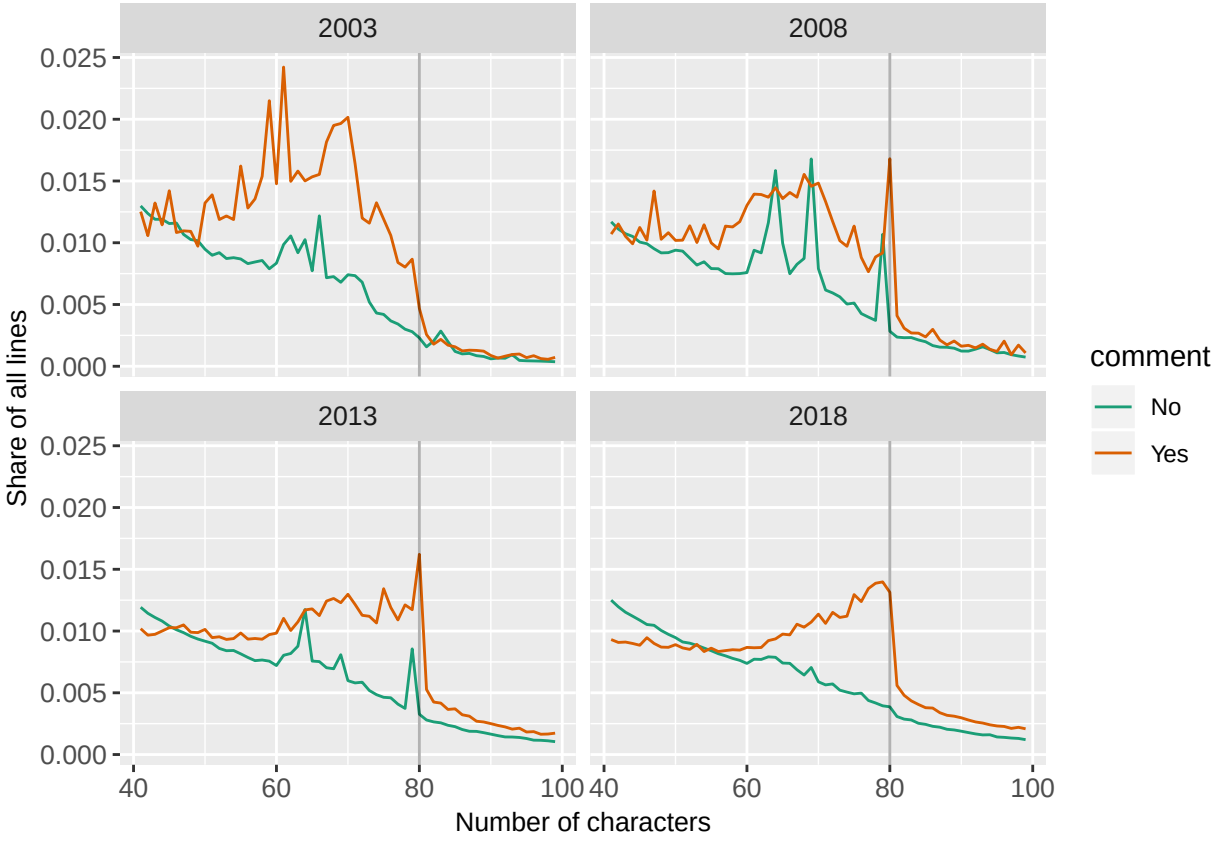

Figure 3: Change in line distribution: 2003, 2008, 2013 and 2018.

Communitiy-based variations

Discussion

About the authors

References

Bååth, R. (2012). The state of naming conventions in R. The R journal, 4(2), 74-75.

Chambers, J. (2001). Assignments with the = Operator. http://developer.r-project.org/equalAssign.html

Elish, M. O., & Offutt, J. (2002). The adherence of open source java programmers to standard coding practices.

Oman, P. W., & Cook, C. R. (1990). A taxonomy for programming style. Proceedings of the 1990 ACM Annual Conference on Cooperation - CSC '90. doi:10.1145/100348.100385

Pons, P., & Latapy, M. (2005, October). Computing communities in large networks using random walks. In International symposium on computer and information sciences (pp. 284-293). Springer, Berlin, Heidelberg.

VanderPlas, J. (2017). Exploring Line Lengths in Python Packages. https://jakevdp.github.io/blog/2017/11/09/exploring-line-lengths-in-python-packages/

Wang, Z., & Hahn, J. (2017). The Effects of Programming Style on Open Source Collaboration.