Code Metrics for the full system in R

Aarón Blanco Álvarez

28/4/2021

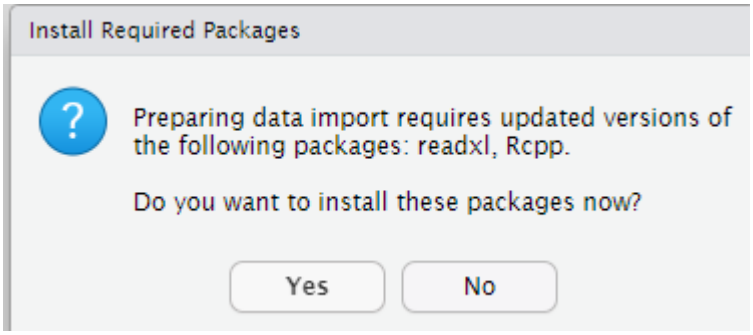
Figure 1: Logo de la UAL

Introduction

In this report we are going to discuss the evolution in the three versions of Lucene. For this goal, we need tools to make the analysis like R Studio, R, etc. With this approach, will get metrics of the system in a clear way than other tools and also, R provide us the possibility of make our own metrics.

Materials and methods

For our purpose, we need RStudio with R, the programming language itself. Also, the data in csv from the execution of Sourcemeter. Furthermore, we will install some packages to load csv among others. Here is an example:




```
# Read xls
library(readxl)
luc49_Class <- read_excel("Excel/Lucene4.9.0/luc49-Class.xlsx")
luc491_Class <- read_excel("Excel/Lucene4.9.1/luc491-Class.xlsx")
luc410_Class <- read_excel("Excel/Lucene4.10/luc410-Class.xlsx")
```

System Metrics

Let us start with the main metrics, after import the excel, we will use R to read the excels.

(Method Hiding Factor –MHF-)

This is the proportion of hiding methods. Firstly, we have to read the excels, we can do that with the **read.csv** instruction, after that, we can save the columns need it in two variables. In this case, NM and NPM. The meaning of this two variables are *number of methods* and *number of public methods*.

```
NM <- luc49_Class[["NM"]];
NPM <- luc49_Class[["NPM"]];
Privado4.9 <- (1-(sum(NPM)/sum(NM)));
print(Privado4.9);
```

```
## [1] 0.1861779
```

```
NM <- luc491_Class[["NM"]];
NPM <- luc491_Class[["NPM"]];
Privado4.9.1 <- (1-(sum(NPM)/sum(NM)));
print(Privado4.9.1);
```

```
## [1] 0.1843877
```

```
NM <- luc410_Class[["NM"]];
NPM <- luc410_Class[["NPM"]];
```

```
Privado4.1 <- (1-(sum(NPM)/sum(NM)));
print(Privado4.1);
```

```
## [1] 0.18451
```

As we can see, we can get private methods proportion if we make the operation of divide number of public methods between all methods, then, subtract to one. In theory, with each version, MHF is increasing, which means that will be less bugs and technical debt, nevertheless, is very small in this case.

(Attribute Hiding Factor –AHF-)

This is the proportion of hiding attributes. In this metric, we can make the same as before but in this case, changing the number of methods for attributes. So, we are working here with the number of attributes and NPA, public attributes. The formula is almost the same, but we are working with that two columns.

```
NAtt <- luc49_Class[["NAtt"]];
NPA <- luc49_Class[["NPA"]];
APrivado <- 1-(sum(NPA)/sum(NAtt));
print(APrivado);
```

```
## [1] 0.5661251
```

```
NAtt <- luc491_Class[["NAtt"]];
NPA <- luc491_Class[["NPA"]];
APrivado4_9 <- 1-(sum(NPA)/sum(NAtt));
print(APrivado4_9);
```

```
## [1] 0.5661843
```

```
NAtt <- luc410_Class[["NAtt"]];
NPA <- luc410_Class[["NPA"]];
APrivado4_1 <- (1-(sum(NPA)/sum(NAtt)));
print(APrivado4_1);
```

```
## [1] 0.5661031
```

Ideally, all attributes should be hidden, nevertheless, 56% it is a reasonable percentage. Anyways, we can see that in this case, there is a little decrease in the results between versions, but the idea would be increase the proportion of hidden attributes.

(Method Inheritance Factor –MIF-)

Inheritance factor means the proportion of inherits methods. So, we can calculate it subtracting the total number of methods with local public methods. After that, we have to divide it from the total number of methods.

```
NM <- luc49_Class[["NM"]];
NLM <- luc49_Class[["NLM"]];
HEREADOS <- (NM - NLM);
MIF <- (sum(HEREADOS)/sum(NM));
print(MIF);
```

```
## [1] 0.8437718
```

```
NM <- luc491_Class[["NM"]];
NLM <- luc491_Class[["NLM"]];
HEREADOS <- (NM - NLM);
```

```
MIF4.9.1 <- (sum(HEREADOS)/sum(NM));
print(MIF4.9.1);
```

```
## [1] 0.8452233
```

```
NM <- luc410_Class[["NM"]];
NLM <- luc410_Class[["NLM"]];
HEREADOS <- (NM - NLM);
MIF4.1 <- (sum(HEREADOS)/sum(NM));
print(MIF4.1);
```

```
## [1] 0.8453527
```

In this case, we have a high percentage. This range should not be low but also, not very high, this indicates either superfluous inheritance or too wide member scopes. In this example, the proportion is increasing, nevertheless, there is not a major change.

(Attribute Inheritance Factor –AIF-)

This metric is similar to MIF but in this case, we are talking about the attributes.

```
NAtt <- luc49_Class[["NAtt"]];
NLA <- luc49_Class[["NLA"]];
HEREADOS <- (NAtt - NLA);
MIF <- (sum(HEREADOS)/sum(NAtt));
print(MIF);
```

```
## [1] 0.8042234
```

```
NAtt <- luc491_Class[["NAtt"]];
NLA <- luc491_Class[["NLA"]];
HEREADOS <- (NAtt - NLA);
MIF4.9.1 <- (sum(HEREADOS)/sum(NAtt));
print(MIF4.9.1);
```

```
## [1] 0.8042521
```

```
NAtt <- luc410_Class[["NAtt"]];
NLA <- luc410_Class[["NLA"]];
HEREADOS <- (NAtt - NLA);
MIF4.1 <- (sum(HEREADOS)/sum(NAtt));
print(MIF4.1);
```

```
## [1] 0.8026854
```

The results indicate that we may have a high rate of inherits attributes. As in MIF metrics, both should be lower, this could be done with the use of Interfaces instead of Inherits.

Polymorphism Factor (PF)

This metric was impossible to get since we have not got the number of overrides methods.

```
#The formula should be something like:
# PF = overrides / sum for each class(new methods * descendants)
```

(Coupling Factor –CF-)

This metric indicates the coupling factor, which measures couplings among classes. In other words, the communication between classes. This formula use the column CBO, which means the total number of classes that use other classes, furthermore, we have to use NOA, the number of ancestors. To make the metric work, we need the number of total classes too. We have that number getting any column and measure it with length.

```
CBO <- luc49_Class[["CBO"]];
NOA <- luc49_Class[["NOA"]];
numeroClases <- length(luc49_Class[["ID"]]);
NumR <- CBO-NOA;
CF <- (numeroClases - 1)/sum(NumR);
print(CF);
```

```
## [1] 0.192708
```

```
CBO <- luc491_Class[["CBO"]];
NOA <- luc491_Class[["NOA"]];
numeroClases <- length(luc491_Class[["ID"]]);
NumR <- CBO-NOA;
CF <- (numeroClases - 1)/sum(NumR);
print(CF);
```

```
## [1] 0.1926064
```

```
CBO <- luc410_Class[["CBO"]];
NOA <- luc410_Class[["NOA"]];
numeroClases <- length(luc410_Class[["ID"]]);
NumR <- CBO-NOA;
CF <- (numeroClases - 1)/sum(NumR);
print(CF);
```

```
## [1] 0.1914854
```

As we can see, we have also no big differences, nevertheless, we can see how the coupling factor is decreasing with every version.

Personal metrics.

Proportion Number of Methods for total lines of code (NMTLC).

We can create a new metric using the proportion of number of methods divide by lines of code. This could leads us to measure the complexity.

```
LOC1 <- luc49_Class[["LOC"]];
NM1 <- luc49_Class[["NM"]];
Total1 <- (sum(LOC1))/sum(NM1);
print(Total1);
```

```
## [1] 3.043147
```

```
LOC2 <- luc491_Class[["LOC"]];
NM2 <- luc491_Class[["NM"]];
Total2 <- (sum(LOC2))/sum(NM2);
print(Total2);
```

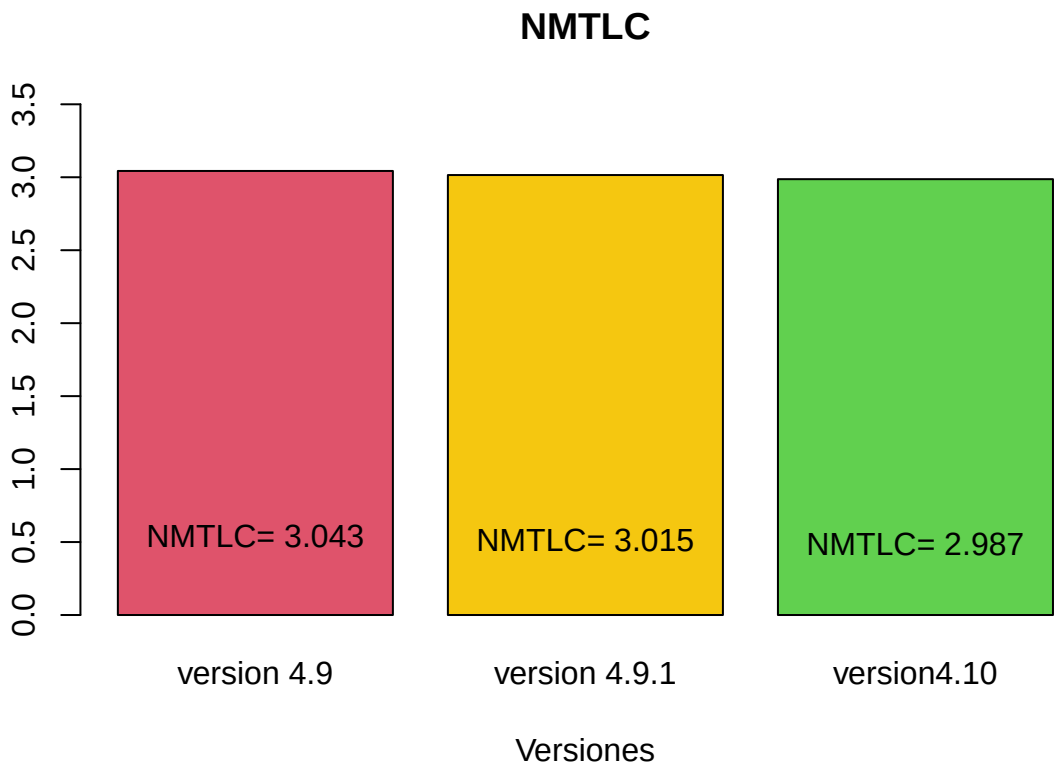
```
## [1] 3.015236
```

```
LOC3 <- luc410_Class[["LOC"]];
NM3 <- luc410_Class[["NM"]];
Total3 <- (sum(LOC3))/sum(NM3);
print(Total3);
```

```
## [1] 2.986877
```

```
media <- c(Total1, Total2, Total3)

barMedia <- barplot(media, names = c("version 4.9", "version 4.9.1", "version4.10"), ylim = c(0,3.5), x
text(barMedia, media -2.5, paste("NMTLC= ", format(round(media, 3), nsmall = 3), sep=""), cex=0)
```



In this case, the proportion is low, so we could think that in this case, there is not a large complexity, nevertheless, this may not be the best metric calculate the complexity, although is clearly that with every version, the complexity is lower.

With this metric we can think that in some cases like more than 4,5 lines of code per method means that we will have high complexity. let us review that.

```
filter(LOC1[2]/NM1[2] >= 4,5 );
```

```
## Time Series:
## Start = 1
## End = 1
## Frequency = 1
## [1] 5
```

```
print(LOC1[2]);
```

```
## [1] 9
```

```
print(NM1[2]);
```

```
## [1] 2
```

```
print(LOC1[2]/NM1[2]);
```

```
## [1] 4.5
```

```
filter(LOC1[6]/NM1[6] <= 4,5 );
```

```
## Time Series:
## Start = 1
## End = 1
## Frequency = 1
## [1] 5
```

```
print(LOC1[6]);
```

```
## [1] 53
```

```
print(NM1[6]);
```

```
## [1] 106
```

```
print(LOC1[6]/NM1[6]);
```

```
## [1] 0.5
```

In the example above, we can see two cases, for example, in the first one, the proportion is low, this is because in the first row, we have many methods, in the second case, we don't have methods, so the proportion is lower than 4,5.

Attribute mean

This metric provide us with information about our classes, more attributes could indicate low encapsulation, with leads us to a low quality code. For example, if you are defining a class Person, and this one have attribute like name, surname, email, street, city, number, etc, one attribute like address could encapsulate all in other class instead of have every attribute in class Person.

So, in this case, we have a way of check this.

```
 library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
 luc49_Class <- filter(luc49_Class, NAtt > 0)
 NAtt1NotEmpty <- luc49_Class[["NAtt"]];
 MediaNatt1 <- mean(NAtt1NotEmpty)
```

```
print(MediaNatt1);
```

```
## [1] 15.26778
```

```
luc491_Class <- filter(luc491_Class, NAtt > 0)
NAtt2NotEmpty <- luc491_Class[["NAtt"]];
MediaNatt2 <- mean(NAtt2NotEmpty)

print(MediaNatt2);
```

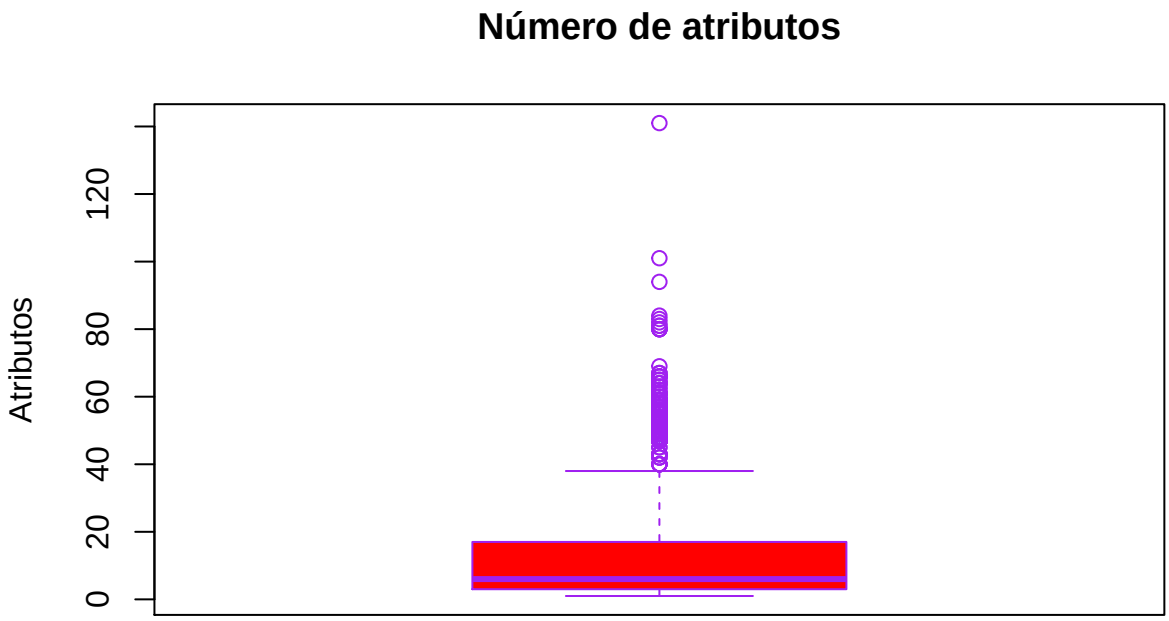
```
## [1] 15.2671
```

```
luc410_Class <- filter(luc410_Class, NAtt > 0)
NAtt3NotEmpty <- luc410_Class[["NAtt"]];
MediaNatt3 <- mean(NAtt3NotEmpty)
print(MediaNatt3);
```

```
## [1] 15.35326
```

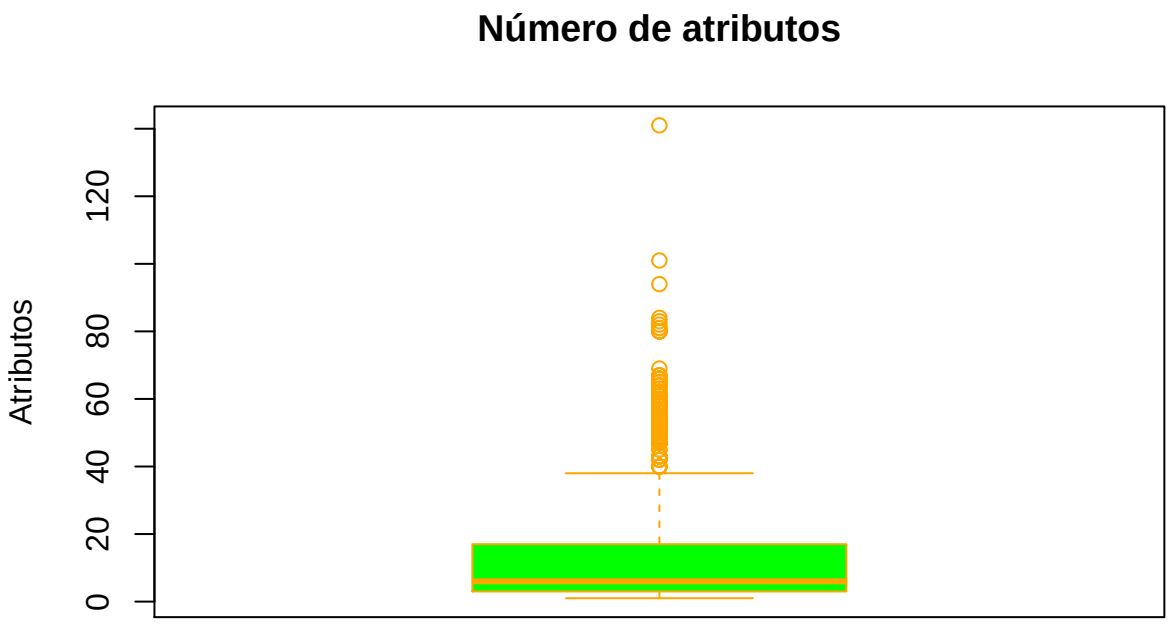
```
boxplot(NAtt1NotEmpty,
        main = "Número de atributos",
        ylab = "Atributos",
        col = "red",
        border = "purple",
        names = c("Version 9.0")

        )
```



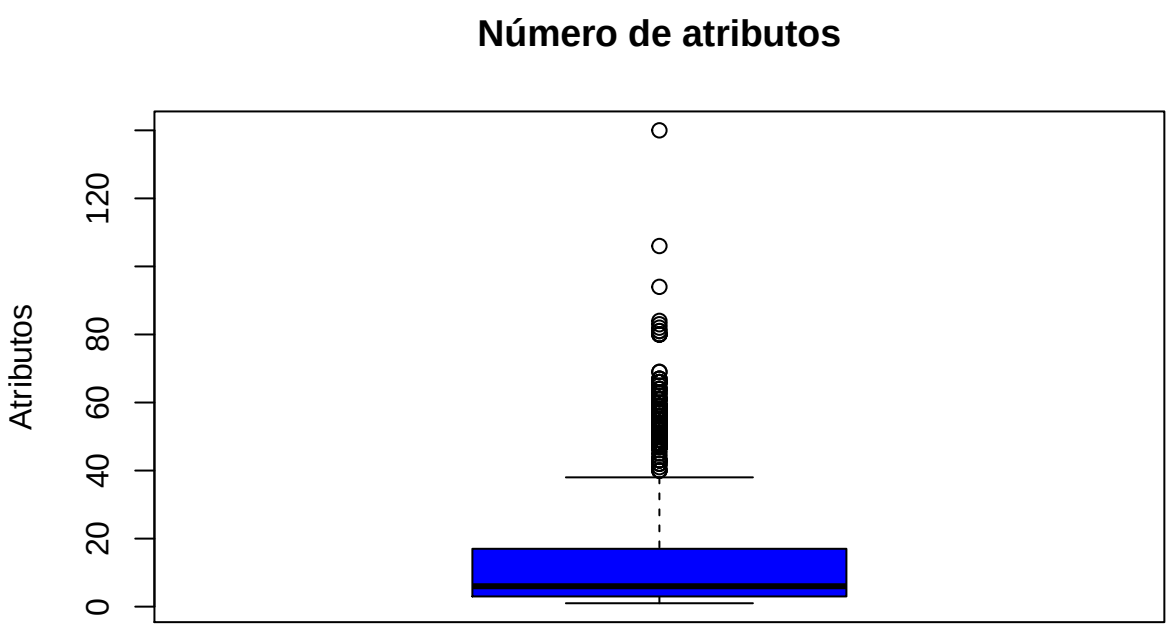
```r
boxplot(NAtt2NotEmpty,
        main = "Número de atributos",
        ylab = "Atributos",
        col = "green",
        border = "orange",
        names = c("Version 9.1")

        )
```




```r
boxplot(NAtt3NotEmpty,
        main = "Número de atributos",
        ylab = "Atributos",
        col = "blue",
        border = "black",
        names = c("Version 9.10")

        )
```



What we have done it is a way of exclude classes with zero attribute and then, calculate the mean. This provide us a way of take into account if we are watching a complex class that may be encapsulate better. For example, now that we know that the mean is of 15, we can see in the graphs that some classes have up to 120, so, verify that should be urgent. If we can verify classes, we only need to use a filter now that we know the average.

Conclusions

To sum-up, we have observe the difference with every version, it is seems clearly, that the previous version was better in most of the cases, meaning that with the new releases, a few things could change, leaving the code with less quality, nevertheless, this differences are also minimal. Some of the indicators, show us the necessity of do refactoring in the code, which can be a complex task in a big project like Lucene, although, some things like the number of inherit attributes should change in the future.