Lecture 10 – Refactoring Code

Learning Objectives:

3. **Learn the basic principles of software design.**

    3.4. Understand how, why, and when to refactor code.

Territorial Acknowledgement: Chapman University exists on the occupied land of the **Tongva people**

What is ‘Refactoring’?

- Process by which code is *restructured* without changing what the code *produces*

- Improves readability and reduces complexity

- Cleaner code that is easier to maintain

- Can help fix vulnerabilities and bugs

- Often makes code more compact and versatile

What are the goals of refactoring?

Good goals 👍🏻

- ✅ Improve readability & understandability
- ✅ Breaking up code into logical pieces
- ✅ Reducing repetition & duplication
- ✅ Improving naming & location of code
- ✅ Improving documentation

NOT good goals 🙅🏼‍♀️

- ❌ “Rewriting” code
- ❌ Code golfing
- ❌ Optimization (different step!)
- ❌ Improving looks (interface)

When should you refactor code?

- ***After*** your code works and does the thing you want it to do.

- ***After*** you've written a good test (to make sure it keeps doing that thing correctly).

- ***"Rule of 3"*** or ***"Three strikes and refactor"*** third time you have to use/copy the code, you refactor.

- ***Around*** the time you optimize the code.

- ***Around*** the time you have a code review.

- ***Before*** you give it to undergrads (or other users/developers).

- ***Before*** doing production runs of data analysis.

- ***Before*** publishing/leaving a lab/putting it down for several months.

Code Smells – Areas where refactoring can help

Smelly code 	**Nicer code** 
Shotgun surgery	Consider loading a parameter list
Long Parameter Lists	
Too long or too short variable names	Rename during refactor
Commenting/uncommenting to alter behavior	Use conditionals for flow control
Using `attach()`	Try `with()` instead
Using `setwd()`	Use R Projects, standard organization
Excessive use of `if` and `else`	Try switch, early exits, or time to move to object-oriented programming
Lots and lots of indentation	

Example of Refactored Code

generate_grid2d.R is a file for producing .vertex files and .csv files for use with IBAMR (C/C++ library). It gives the location of points that create a boundary of the hairs in a 2D hair array (circles). Code is looped to produce one vertex and one csv file per simulation in a set of simulations with different parameter values.

Why was it written this way?

- Original code in MATLAB, this is how it was written.

- Easiest way to explain what was happening to undergrads.

- Trying to put something together quickly.

What were my goals for refactoring?

- Reduce duplications

- Clean up unused variables

- Make the code more modular

Example of Refactored Code

Pre-refactored code

```
#### Produces points within defined hairs ####

# Antennule
ant<-circle(c(0,0),0.5*adia,L,dx);  # Produces points that define antennule
aN<-size(ant$X,2)                   # Records number of points inside antennule
plot(Y~X,data=ant,xlim=c(-0.5,0.5),ylim=c(-0.5,0.5),pch=19) #Plots antennule

# Hair 1
h1<-circle(c(hair1Centerx,hair1Centery),0.5*hdia,L,dx)
h1N<-size(h1$X,2)
disp(h1N)
points(Y~X,data=h1,pch=19)

# Hair 2
h2<-circle(c(hair2Centerx,hair2Centery),0.5*hdia,L,dx)
h2N<-size(h2$X,2)
points(Y~X,data=h2,pch=19)

# Hair 3
h3<-circle(c(hair3Centerx,hair3Centery),0.5*hdia,L,dx)
h3N<-size(h3$X,2)
points(Y~X,data=h3,pch=19)
```

x 8

Post-refactored code

```
#### Produces points within defined hairs ####
# Antennule
ant <- circle(c(0, 0), 0.5 * adia, dx);  # Produces points that define antennule
aN <- size(ant$X, 2)                     # Records number of points inside antennule
if(plotit == 1){
  plot(0, 0, xlim = c(-0.5, 0.5), ylim = c(-0.5,0.5), pch = 19, cex = 4) #Plots antennule
  text(0, 0, labels = "Ant", col = "red")
}
# Each hair
for (i in 1:nohairs){
  hairx <- eval(as.name(paste("hair", i, "Centerx", sep = "")))
  hairy <- eval(as.name(paste("hair", i, "Centery", sep = "")))
  h <- plotahair(hairx, hairy, hdia, dx, i, plotit)
  assign(paste("h", i, sep = ""), h)
}
```

all 8 hairs

```
filename<-paste("hairs",number,".vertex",sep="")   # Defines file name
if(file.exists(filename)) file.remove(filename)  # Deletes file with that name if it exists
cat(as.character(totalN),sep="\n",file=filename,append=TRUE)
for (i in 1:aN){
cat(c(as.character(ant$X[i])," ",as.character(ant$Y[i]),"\n"),file=filename,sep="",append=TRUE)
}
for (i in 1:h1N){
cat(c(as.character(h1$X[i])," ",as.character(h1$Y[i]),"\n"),file=filename,sep="",append=TRUE)
}
for (i in 1:h2N){
cat(c(as.character(h2$X[i])," ",as.character(h2$Y[i]),"\n"),file=filename,sep="",append=TRUE)
}
for (i in 1:h3N){
cat(c(as.character(h3$X[i])," ",as.character(h3$Y[i]),"\n"),file=filename,sep="",append=TRUE)
}
for (i in 1:h4N){
  cat(c(as.character(h4$X[i])," ",as.character(h4$Y[i]),"\n"),file=filename,sep="",append=TRUE)
}
```

x 8

```
#### Write points to vertex file ####
totalN <- aN + nohairs * hN  # Calculates total number of points (first line of vertex file)

filename <- paste("hairs", number, ".vertex", sep = "")   # Defines file name
if(file.exists(filename)) file.remove(filename)  # Deletes file with that name if it exists
cat(as.character(totalN), sep = "\n", file = filename, append = FALSE)
# Writes antennule points
write.table(ant,file=filename, sep=" ", append=TRUE, col.names = FALSE, row.names = FALSE, qu
# Writes hair points
for (k in 1:nohairs){
  hair <- eval(as.name(paste("h", k, sep = "")))
  write.table(hair, file = filename, sep = " ", append = TRUE,
              col.names = FALSE, row.names = FALSE, quote = FALSE)
}
```

all 8 hairs

Example of Refactored Code

What were the results of refactoring?

- More modular
- Much less duplication / repetition
- More compact and understandable
- Faster (even though that wasn't a goal)

Group work: In breakout groups, look at lines 100 – 123 in the post-refactor code. This sets the center of each hair (as x and y coordinates) for each hair in the array. This is very repetitive code.

1) Should it be refactored?
2) How could you go about refactoring it?

What other areas of the code could benefit from additional refactoring?

More Information

https://github.com/jennybc/code-smells-and-feels – "Code Smells and Feels" by Jenny Bryan, a talk at the UseR conference 2018

http://silab.fon.bg.ac.rs/wp-content/uploads/2016/10/Refactoring-Improving-the-Design-of-Existing-Code-Addison-Wesley-Professional-1999.pdf – Refactoring: Improve the Design of Existing Code by Martin Fowler