Bad Smell – Evaluation

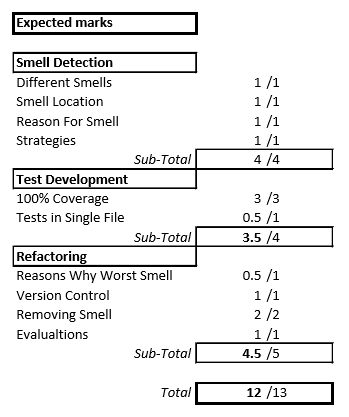
# Smell 1: Switch Statement: Controller.display()

The **Extract Method** and **Move Method** steps for refactoring this smell went as expected. However, the third step (**Replace Conditional with Polymorphism**) was interesting because to condense the switch statement I experimented with using a dictionary that contained the input flags as keys and, as values, store lambdas to call the different display\_chart() functions.

By doing this I realized I didn’t need the polymorphism at all, and instead I could just condense the duplicate display\_chart() functions into one function using the **Consolidate Conditional Expression** refactoring technique.

I ended up doing this by keeping the dictionary with the input flags as keys, but instead of storing lambdas in the values, I stored the associated pygal object for each input flag.

Now the pygal object would be selected from the dictionary (using the flag passed in by the Controller) and inserted into the (now consolidated) build\_chart() function, which returned the pygal object with the data and title inserted, and then rendering it.

I was happy with this solution, because even though I chose the wrong refactor technique in my planning, my final result was much cleaner than I expected it would be.

**Final Note:** This refactoring pass was good because it fixed the duplicate code bad smell that was in the Visualiser class.

# Smell 2: Large Class: Controller()

This bad smell was not as bad to refactor as I thought it might be. I created the new serializer.py module and moved the Controller.Serialize() method into the Serializer class.

I only had to change one line of code which was the db\_contents that would be used to Serialize. But this was not a problem.

I forgot, though, that I would have to inject the new dependency into the controller’s constructor through main.py. But this was fine once I remembered to do it.

# Smell 3: Feature Envy: Controller.Display()

Refactoring this smell was straight forward to begin with, but I quickly realized my design was worse than I originally thought! Once I had **extracted** and **moved** the two methods into the Validator class, I realized I had stored the valid flags for the Visualizer class inside the Validator class, when they should be in the Visualizer class!

When initially writing the code, I thought it would make sense for the Validator class to ‘validate’ everything. But I decided now it was better suited to move this functionality out of the Controller and Validator, and into the Visualizer, especially since the Visualizer was now storing its own list of flags. (This is also a part of the design for my whole application that needs to change, and I don’t know why I didn’t think of it earlier!!)

It also meant I could get rid of the lists (dead code) I was using to make these comparisons within the Validator class, which cleaned it up.

***Final Note:*** *I realized I could get rid of this functionality entirely (calling Boolean functions from the Controller to check for validity) by simply placing some assertions into the Visualizer class, BUT, I think that would be outside the scope of this bad smell, so I decided to leave the functionality as it is.*

# Smell 4: Speculative Generality: View Interface

All of the steps for this smell were fairly basic, and consisted of renaming things. I thought it would create some bugs or other mess, but it was easier than I thought. It was just time consuming to make sure I had changed all of the function calls etc. (which is why I did all of the renaming in separate steps/commits).