# Bad Smell Detection

#### Name: Duplicated Code

Location:

FOLDER\_NAME = umlify

FILE\_NAME = database.py

CLASS\_NAME = Database

METHOD\_NAME = get\_db\_info() / drop\_table() / create\_table() / insert\_data() / remove\_data() / get\_specific() / get\_all()

BETWEEN\_LINES = 56-64, 76-87, 95-106, 114-126, 132-144, 148-175, 190-212

Reasons:

1. The 7 methods have very similar lines of code which is to connect to the database and execute a query. This breaks the DRY principle which requires changes to be made in many places.
2. This could cause issues if a bug within a method was only fixed or a change was done for one occurrence but not in the duplicated versions. Such as if the variables *self.file\_path* and *self.db\_name* were renamed, these changes would need to be made for each method with the line *sql.connect(self.file\_path, self.db\_name).*
3. This would reduce the number of lines for each method and remove duplicated variables such as *\_conn, \_cursor* as there are already *self.\_conn, self.\_cursor*

Strategy/Approach

Use the Extract Method to make a general case. Taking the similar lines of code which connects to the database and executes a query into a new method. This method takes in the parameter of a SQL command given by the existing methods. This new method will be called *query().*

#### Name: Large Class

Location:

FOLDER\_NAME = umlify

FILE\_NAME = extractor.py

CLASS\_NAME = Extractor

METHOD\_NAME = regex\_search(), extract\_class(), extract\_parents(), extract\_functions(), extract\_attributes(), extract defaults\_datatypes(), extract\_attribute\_defaults(), extract\_attribute\_datatypes()

BETWEEN\_LINES = 82-85, 87-90, 92-104, 106-108, 110-112, 114-121, 123-128, 130-150

Reasons:

1. This Extractor class has too many responsibilities. It goes against the Single Responsibility principle of SOLID. There some functions that could be better suited in their own class to separate from the main data extractor. This would be the *extract\_class(), extract\_parents(), extract\_functions(), extract\_attributes()* methods which gets all the main attributes of the Component class.
2. The Database class, Shelf class, ComponentViewer class, UmlifyComponentViewer class, depend on the Extractor class for extracting the data from a .py file.
3. Fixing this code smell would greatly reduce the lines of code within the Extractor class which will make it easier to understand for future changes. This would make the Extractor methods easier to clean or modify.

Strategy/Approach

Using the Extract Class technique, I first must determine which methods to extract from the original class. I will move the *extract\_class(), extract\_parents(), extract\_functions(), extract\_attributes()* methods to a new class which will be called *ComponentExtractor* in the module *component\_extractor.py*. I will replace the original functions within the Extractor class with a call to the new methods.

The methods *extract\_defaults\_datatypes(), extract\_attribute\_defaults(), extract\_attribute\_datatypes()* will also be placed into their own class called *DataTypeExtractor* in the module *data\_type\_extractor.*

I will then establish the relationship between the *Extractor* class and the two newly created classes by referencing them as objects and making them attributes in *\_\_init\_\_* to be accessed by the methods within *Extractor* class.

#### Name: Long Method

Location:

FOLDER\_NAME = umlify

FILE\_NAME = extractor.py

CLASS\_NAME = Extractor

METHOD\_NAME = data\_extraction()

BETWEEN\_LINES = 40-79

Reasons:

1. This method contained too many lines of code which made it more difficult to understand how the method works (as this was not the class I wrote).
2. In a way, this method breaks the Single Responsibility principle as there are three different cases being performed (indicated by the conditional statements *if class\_name, elif function\_name, elif attribute\_name*).
3. While reading through the block of code, there were variables referenced before their assignment which does not conform to the PEP8 style. This could have been ignored to prevent having more lines of code written in the method or it could have been overlooked due to the large amount of code.

Strategy/Approach

Using the Extract Method technique, I first create a new method called *add\_class\_and\_parents\_to\_dict()* which will hold the block of code from lines 48-59 and remove the original copy from the method *data\_extraction().* In place of this, I put a call for the new method and added parameters required for the new method (class\_name, line).

This was done the same for lines 62-68 and lines 70-79. To deal with the local variable *comp*, I need to set this as an attribute of the Extractor class so that it can be used by all the new methods. The other two methods will be called *add\_function\_to\_dict()* and *add\_attribute\_to\_dict()* with their needed parameters.

# Refactoring Evaluation

## Code Smell 1: Duplicated Code

### Testing

Since I hadn’t written a full test coverage in the previous assignment, I added unit tests for each method before the refactoring process. And since I added a new method *query()*, I wrote a test for this method as well which would check if exceptions has been thrown. For each method in the Database class that I refactored, I would run tests relevant to the individual behaviour to check if the refactoring process had not change the functionality of the method. But during this process, I had to modify my tests as I struggled with handling the exceptions and errors of the methods. Only in a perfect scenario do these methods work successfully. The new set of tests are added to the unittest\_database.py. Unittests were executed using test\_runner.py. Unittests were executed using test\_runner.py

**TEST COVERAGE:**





### Version Control

For every method that I refactored, I commit the changes and push it to my remote repository. There were at least 7 commits relevant to refactoring this code smell.

### Evaluation

After refactoring database.py, I had issues of error handling of invalid cases in most of my methods and I had to remove my print statements that indicated to the user of whether a method was executed properly. During the process, I created another method which wasn’t relevant to refactoring the code smell, *close\_connection()***,** which is used in all the methods after the *query()* method has been performed. The *query()* seems to deal with a lot of the exceptions and errors originally written for the methods so this over complicated the new method *query()* and made it more difficult to code for all methods.

Although I have successfully reduced the number of lines (to increase readability) in the code and made it easier in the future to modify the database connection, I’ve introduced another code smell, ***Large Class.*** The Database class seems as if it has multiple functions such as *create\_connection(), query(), create\_table(), drop\_table(),* *close\_connection()* which could be in a class of its own and separated from the functions that modify or access the data from the database.

## Code Smell 2: Large Class

This was the worst code smell as the Extractor class affects multiple classes which uses the set\_file() method and return a component dictionary holding extracted data from a given file. There were too many lines of code to be able to read and understand in the perspective of someone who didn’t write this class.

### Testing

The new set of tests are added to the extractor\_unittest.py. Unittests were executed using test\_runner.py.

**TEST COVERAGE:**









### Version Control

For every method that I refactored, I commit the changes and push it to my remote repository. There were at least 5 commits relevant to refactoring this code smell.

### Evaluation

I believe that refactoring this code smell was successful and effectively improved the quality of the code by making it easier to manage as its been split into three classes instead of one. Moving the methods to their corresponding classes were simple to do and did not require much modification to the main extractor class other than creating new attributes for the Extractor class (to inject the new dependencies). The only drawback during the refactoring process was that it created duplicate code within the two new classes *ComponentExtractor* and *DataTypeExtractor* which was the regex\_search() method. This static method was needed by both classes and it would not have been relevant to the Extractor class if it was not moved. Overall, the refactoring process still improved the readability by reducing the number of lines in a class.

## Code Smell 3: Long Method

### Testing

The new set of tests are added to the extractor\_unittest.py. Unittests were executed using test\_runner.py.

**TEST COVERAGE:**

### 







### Version Control

For every method that I refactored, I commit the changes and push it to my remote repository. There were at least 3 commits relevant to refactoring this code smell.

### Evaluation

The refactoring process for this code smell was straightforward and easy. The result of this was being able to understand *data\_extraction()* better and being able to make changes without the risk of encountering errors which could be overlooked due to the vast amount of code within the method. The refactoring method (Extract Method) did not seem as difficult to use compared to the Duplicated Code smell in the Database class. I think I have successfully removed the Large Class code smell as the number of lines has been reduced by 59 lines of code within the *data\_extraction()* method. The only issue I had was with adding a new attribute to the Extractor class (*self.comp*) which was needed for all the new methods as it was previously used as a local variable within the original method.