Software Engineering Methods: Assignment 2

Group 04a: Bogdan Iordache, Kyongsu Kim, Matyáš Kollert, Anna Lantink,
Kristóf Lévay, Javier Páez Franco

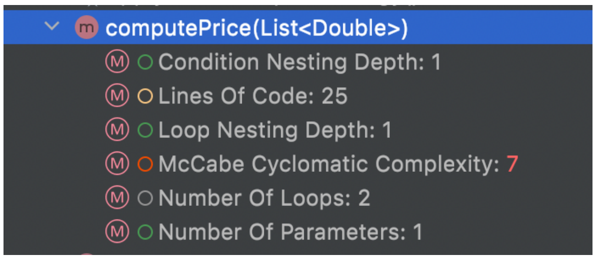
January 18th, 2022

1 Refactoring Methods

1.1 *computePrice(...)* in ApplyBOGOCouponStrategy

One of the methods identified for refactoring was the *computePrice(...)* method in the ApplyBOGO-CouponStrategy class. The reason this method was refactored was because the cyclomatic complexity was flagged by the metrics as too high, and the lines of code were also a little too high, as can be seen in Figure 1.

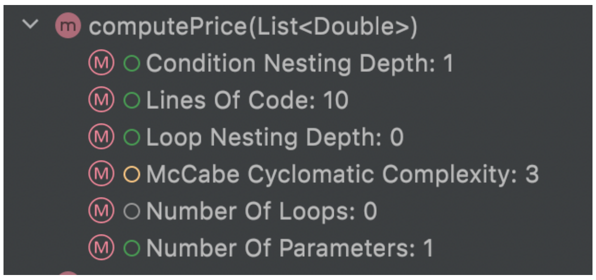
Noticing that these problems were flagged by the metrics, the method was manually looked through to check if these flags were actually valid. For example, checking the validity of the cyclomatic complexity flag was done by seeing if there were any conditionals that could be removed or replaced and still have the function work the same. Indeed, in this class there were several unnecessary conditionals and for loops that clogged the class, and could be refactored.



```
computePrice(List<Double>)
    Condition Nesting Depth: 1
    Lines Of Code: 25
    Loop Nesting Depth: 1
    McCabe Cyclomatic Complexity: 7
    Number Of Loops: 2
    Number Of Parameters: 1
```

Figure 1: The original method metrics for the computePrice method in ApplyBOGOCouponStrategy

This method needs to calculate the sum of prices, and also the cheapest price in a list. Prior to refactoring, these were calculated manually using conditionals and for loops. In the case of these simple operations, cleaner solutions exist. For summing the prices, instead of a for loop, a stream was used with the *sum()* operation. For refactoring the cheapest price in a list, the list was sorted using the *Collections* class in Java, and the first item was taken. There was another conditional that checked if a value was non-negative, which was simply replaced with *Math.max(0, value)*. Replacing manual calculations with more elegant solutions significantly brought down both the lines of code and the cyclomatic complexity of the method, as can be seen in Figure 2. The remaining cyclomatic complexity consisted of necessary case checking, that could not be brought down further.

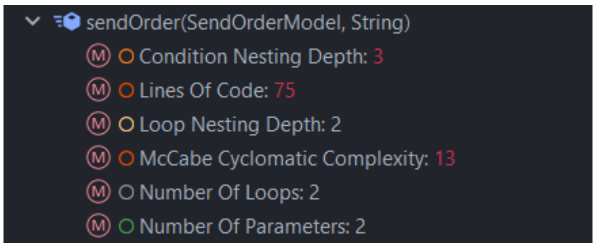


```
computePrice(List<Double>)
    Condition Nesting Depth: 1
    Lines Of Code: 10
    Loop Nesting Depth: 0
    McCabe Cyclomatic Complexity: 3
    Number Of Loops: 0
    Number Of Parameters: 1
```

Figure 2: The refactored method metrics for the computePrice method in ApplyBOGOCouponStrategy

1.2 *sendOrder(...)* in OrderController

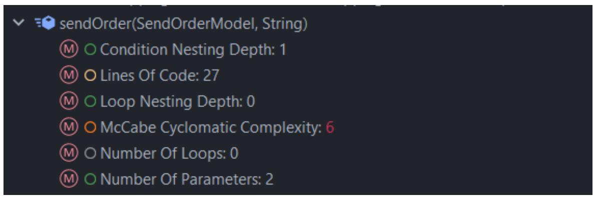
The *sendOrder* method is a crucial component of our system as it manages the creation of new orders, incorporating the initial products and their associated toppings, and subsequently storing them in the database. As a result, it is a method of substantial complexity, as depicted in Figure 3.



```
sendOrder(SendOrderModel, String)
    Condition Nesting Depth: 3
    Lines Of Code: 75
    Loop Nesting Depth: 2
    McCabe Cyclomatic Complexity: 13
    Number Of Loops: 2
    Number Of Parameters: 2
```

Figure 3: The original method metrics for the *sendOrder* method in Order Controller

The functionality of the *sendOrder* method can be significantly enhanced through the utilization of the "ModifyOrderService" helper class, specifically the *addProductsToOrder* method. Currently, the *sendOrder* method is responsible for both adding products to the order and storing it in the database, but it does not utilize the helper method. By refactoring it to utilize *addProductsToOrder*, we can eliminate duplicated code. This will result in improved metrics, with a reduction of 64% in the number of lines and 54% in complexity, as illustrated in Figure 4.



```
sendOrder(SendOrderModel, String)
    Condition Nesting Depth: 1
    Lines Of Code: 27
    Loop Nesting Depth: 0
    McCabe Cyclomatic Complexity: 6
    Number Of Loops: 0
    Number Of Parameters: 2
```

Figure 4: The refactored method metrics for the *sendOrder* method in Order Controller

1.3 *addProductsToOrder* in ModifyOrderService

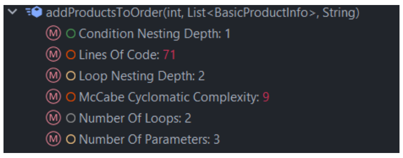
The *addProductsToOrder* helper method within the "ModifyOrderService" class can also be refactored to improve its performance. Its current metrics are presented in Figure 5. It can be observed that it has more lines and complexity than what is recommended.

Similar to the *sendOrder* method, the *addProductsToOrder* helper method also has the issue of duplicating code when adding toppings to products. There is already a method within the ”ModifyOrderService” class, *addToppingsToProduct*, that performs this task, but it is not being utilized. To improve the performance, the existing helper method was used.

Another change that was made was refactoring the logic to check if the order was paid or cancelled. It was refactored into a separate method, *checkStatus()*, as these checks are required in most methods of the class. Another extracted part of the methods is now called *handleAddingProduct()*. This change improved the code readability and also all the metrics.

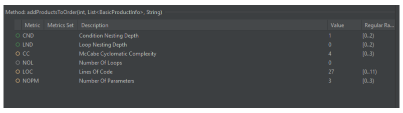
Finally, to reduce the complexity of the code, the code was reviewed for unnecessary loops, conditionals, etc.

By refactoring the method to make use of helper methods and reducing the number of loops in the code, we can achieve the metrics presented in Figure 6, with a reduction of 70% in the number of lines and over 50% in complexity.



```
addProductsToOrder(int, List<BasicProductInfo>, String)
    Condition Nesting Depth: 1
    Lines Of Code: 71
    Loop Nesting Depth: 2
    McCabe Cyclomatic Complexity: 9
    Number Of Loops: 2
    Number Of Parameters: 3
```

Figure 5: The original method metrics for the *addProductsToOrder* method in ModifyOrderService



Method: addProductsToOrder(int, List<BasicProductInfo>, String)

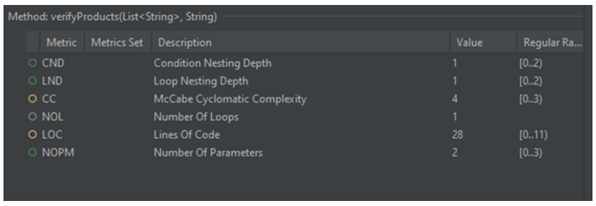
Metric	Metrics Set	Description	Value	Regular Ra...
CND		Condition Nesting Depth	1	[0..2)
LND		Loop Nesting Depth	0	[0..2)
CC		McCabe Cyclomatic Complexity	4	[0..3)
NOL		Number Of Loops	0	
LOC		Lines Of Code	27	[0..11)
NOPM		Number Of Parameters	3	[0..3)

Figure 6: The refactored method metrics for the *addProductsToOrder* method in AddToProductService

Unrelated to this change, this method was also moved to a new class called ”AddToProductService” as a part of class refactoring.

1.4 *verifyProducts(...)* in ProductService

Another method identified for refactoring was the *computePrice(...)* method in the ProductService class. We refactored this method since the cyclomatic complexity as well as the lines of code weren’t ideal, as can be seen in Figure 7. When we checked the method, there were definitely parts that could be extracted into separate methods, which would improve both metrics.

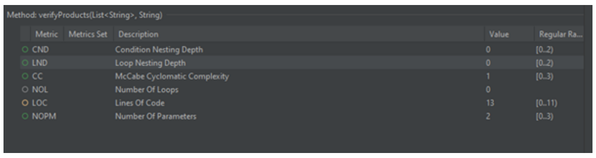


Method: verifyProducts(List<String>, String)

Metric	Metrics Set	Description	Value	Regular Ra...
CND		Condition Nesting Depth	1	[0..2)
LND		Loop Nesting Depth	1	[0..2)
CC		McCabe Cyclomatic Complexity	4	[0..3)
NOL		Number Of Loops	1	
LOC		Lines Of Code	28	[0..11)
NOPM		Number Of Parameters	2	[0..3)

Figure 7: The original method metrics for the verifyProducts method in ProductService

This method checks whether the list of products passed to it corresponds to database entries. For each product, it also checks for all the topping. This inner loop was extracted into its separate method which, along with removing one if condition, lowered both the lines of code and the cyclomatic complexity of the method, as can be seen in Figure 8. The lines of code could not be lowered any more since the tool includes the JavaDoc in it, and we also decided that the method was small enough.



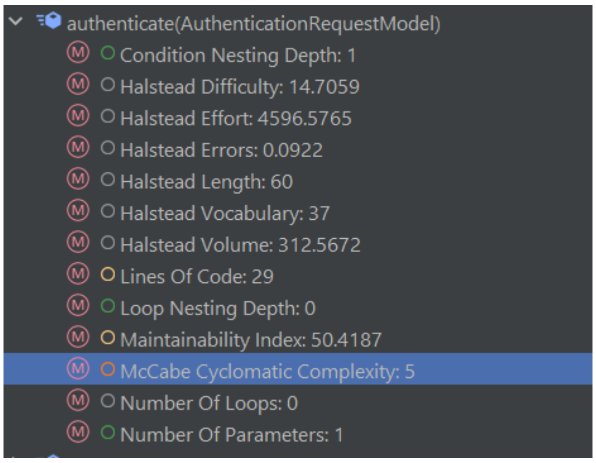
Method: verifyProducts(List<String>, String)

Metric	Metrics Set	Description	Value	Regular Ra...
CND		Condition Nesting Depth	0	[0..2)
LND		Loop Nesting Depth	0	[0..2)
CC		McCabe Cyclomatic Complexity	1	[0..3)
NOL		Number Of Loops	0	
LOC		Lines Of Code	13	[0..11)
NOPM		Number Of Parameters	2	[0..3)

Figure 8: The refactored method metrics for the verifyProducts method in ProductService

1.5 *authenticate(...)* in User-microservice

The authenticate method in the User microservice's Authentication Controller class was found to have a high cyclomatic complexity and an excessive amount of lines of code, as depicted in Figure 9. To address these issues, it was decided that the best course of action was to extract specific parts of the code into separate, distinct methods. This would result in a more readable and maintainable codebase, while preserving the original functionality of the method.



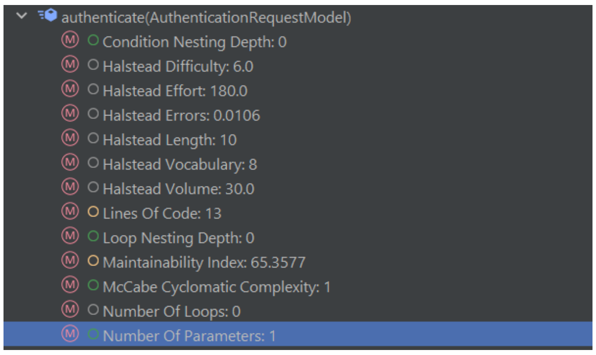
```
authenticate(AuthenticationRequestModel)
    Condition Nesting Depth: 1
    Halstead Difficulty: 14.7059
    Halstead Effort: 4596.5765
    Halstead Errors: 0.0922
    Halstead Length: 60
    Halstead Vocabulary: 37
    Halstead Volume: 312.5672
    Lines Of Code: 29
    Loop Nesting Depth: 0
    Maintainability Index: 50.4187
    McCabe Cyclomatic Complexity: 5
    Number Of Loops: 0
    Number Of Parameters: 1
```

Figure 9: The original method metrics for the authenticate Method

Authentication is done by first checking if the email and password input are provided. Then, it verifies that the email exists in the database and that the password hashes match. If successful, a JWT token is generated using the user's role and email and sent as the response. In case of failure, matching exceptions are returned.

To improve the code structure, these functionalities are broken down into three smaller methods to reduce complexity and code length, as well as make it more modular. The 3 methods that are thus extracted are *validateInput()*, *authenticateUser()* and *generateToken()*. As we can observe in the screenshots, after the extraction of the methods, the cyclomatic complexity of the method we wanted to refactor went down to 1, and the code was obviously reduced.

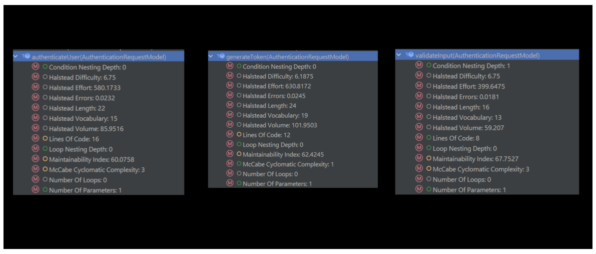
The metrics for the extracted methods can be observed below, notice how after the refactoring, all



```
authenticate(AuthenticationRequestModel)
    Condition Nesting Depth: 0
    Halstead Difficulty: 6.0
    Halstead Effort: 180.0
    Halstead Errors: 0.0106
    Halstead Length: 10
    Halstead Vocabulary: 8
    Halstead Volume: 30.0
    Lines Of Code: 13
    Loop Nesting Depth: 0
    Maintainability Index: 65.3577
    McCabe Cyclomatic Complexity: 1
    Number Of Loops: 0
    Number Of Parameters: 1
```

Figure 10: The refactored method authenticate(...)

3 classes now have no metric too high, making this refactoring worth implementing.



```
authenticateUser(AuthenticationRequestModel)
    Condition Nesting Depth: 0
    Halstead Difficulty: 6.75
    Halstead Effort: 580.1733
    Halstead Errors: 0.0232
    Halstead Length: 22
    Halstead Vocabulary: 15
    Halstead Volume: 85.9516
    Lines Of Code: 16
    Loop Nesting Depth: 0
    Maintainability Index: 60.0758
    McCabe Cyclomatic Complexity: 3
    Number Of Loops: 0
    Number Of Parameters: 1
```

```
generateToken(AuthenticationRequestModel)
    Condition Nesting Depth: 0
    Halstead Difficulty: 6.1875
    Halstead Effort: 630.8172
    Halstead Errors: 0.0245
    Halstead Length: 24
    Halstead Vocabulary: 19
    Halstead Volume: 101.9503
    Lines Of Code: 12
    Loop Nesting Depth: 0
    Maintainability Index: 62.4245
    McCabe Cyclomatic Complexity: 1
    Number Of Loops: 0
    Number Of Parameters: 1
```

```
validateInput(AuthenticationRequestModel)
    Condition Nesting Depth: 1
    Halstead Difficulty: 6.75
    Halstead Effort: 399.6475
    Halstead Errors: 0.0181
    Halstead Length: 16
    Halstead Vocabulary: 13
    Halstead Volume: 59.207
    Lines Of Code: 8
    Loop Nesting Depth: 0
    Maintainability Index: 67.7527
    McCabe Cyclomatic Complexity: 3
    Number Of Loops: 0
    Number Of Parameters: 1
```

Figure 11: The new methods extracted from authenticate(...))

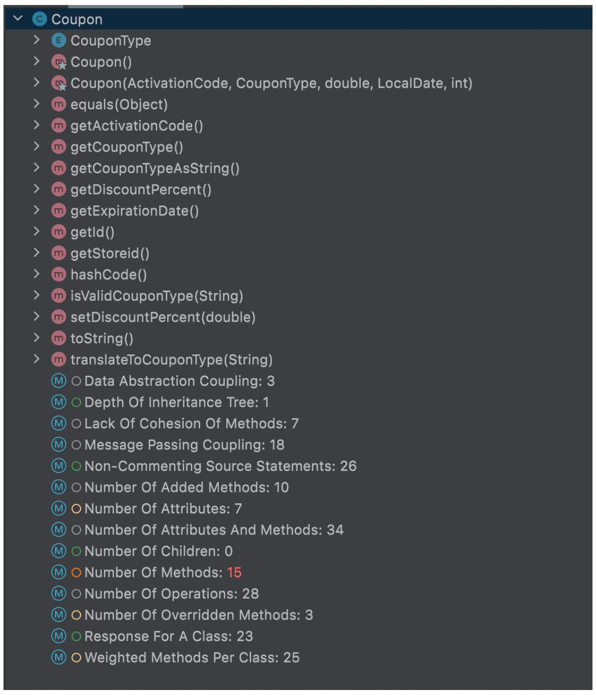
2 Refactoring Classes

2.1 Coupon Class

The class metrics for the Coupon class, prior to refactoring, flagged the number of methods as a problem and, related to that, the weighted methods per class, as can be seen in Figure 12. This indicated that there could be a problem within the class, so the methods in the class were manually analyzed to see if there were any that were unnecessary. As the class metrics indicated, there were several methods that could be removed or combined, so the Coupon class was chosen to be refactored.

The main point of refactoring for the Coupon class was altering or removing methods. There were several methods that were either never used, only used by tests, or had very similar functionality to another method. The reason for this was because as the working prototype was coded, methods would be added to the Coupon class in case they were needed, but were never actually used and also never removed.

The easiest point of refactoring was removing the never used methods. Methods that were used only in tests were also removed, and their tests were either modified or removed. For example, the *equals(...)* method was only ever used to check equality in tests, never in the actual functionality of the program were coupons compared (since only the price saved was compared instead). This made



```
Coupon
  CouponType
  Coupon()
  Coupon(ActivationCode, CouponType, double, LocalDate, int)
  equals(Object)
  getActivationCode()
  getCouponType()
  getCouponTypeAsString()
  getDiscountPercent()
  getExpirationDate()
  getId()
  getStoreid()
  hashCode()
  isValidCouponType(String)
  setDiscountPercent(double)
  toString()
  translateToCouponType(String)
  Data Abstraction Coupling: 3
  Depth Of Inheritance Tree: 1
  Lack Of Cohesion Of Methods: 7
  Message Passing Coupling: 18
  Non-Commenting Source Statements: 26
  Number Of Added Methods: 10
  Number Of Attributes: 7
  Number Of Attributes And Methods: 34
  Number Of Children: 0
  Number Of Methods: 15
  Number Of Operations: 28
  Number Of Overridden Methods: 3
  Response For A Class: 23
  Weighted Methods Per Class: 25
```

Figure 12: The original class metrics for the Coupon class

it seem as though the *equals(...)* method was in use, when in actuality it never was. So, the methods that were only used in tests were removed.

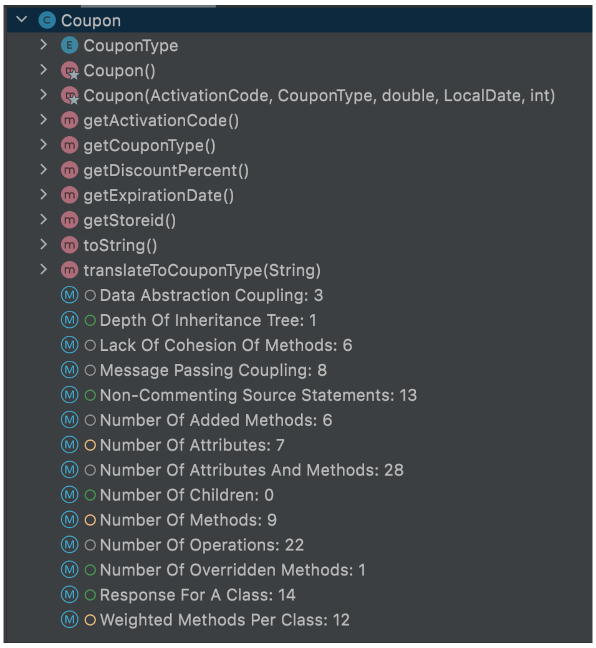
Finally, the biggest point of refactoring was in removing the *isValidCouponType(...)* method by merging its functionality with the *translateToCouponType(...)* method. These two methods were essentially duplicates of each other, but with different return types - one being a boolean and one being a *CouponType*. These two methods were merged by changing translateToCouponType(...) such that it returned an *Optional¡CouponType¿*, where an invalid *CouponType* would be an empty *Optional*. This way, the *isValidCouponType(...)* method could be removed.

As can be seen in Figure 13, the above-mentioned refactoring resulted in the significant reduction of the number of methods, and also the significant reduction of several other metrics such as Message Passing Coupling, Weighted Methods Per Class (which involves Cyclomatic Complexity and Lines of Code, among other things), and Response for a Class.

It's important to note that some of the metrics that were flagged as yellow, such as number of attributes, could not be brought down. For the attributes, this was because upon analysis, all the attributes were used frequently and had unique purposes. So in this case the class metric indicates that there may be a problem, when there actually isn't one.

2.2 OrderController Class

The class that requires the most refactoring within the Order domain is "OrderController", as illustrated in Figure 14. It exhibits high levels of coupling and has an excessive number of parameters.



```
Coupon
  CouponType
  Coupon()
  Coupon(ActivationCode, CouponType, double, LocalDate, int)
  getActivationCode()
  getCouponType()
  getDiscountPercent()
  getExpirationDate()
  getStoreid()
  toString()
  translateToCouponType(String)
    Data Abstraction Coupling: 3
    Depth Of Inheritance Tree: 1
    Lack Of Cohesion Of Methods: 6
    Message Passing Coupling: 8
    Non-Commenting Source Statements: 13
    Number Of Added Methods: 6
    Number Of Attributes: 7
    Number Of Attributes And Methods: 28
    Number Of Children: 0
    Number Of Methods: 9
    Number Of Operations: 22
    Number Of Overridden Methods: 1
    Response For A Class: 14
    Weighted Methods Per Class: 12
```

Figure 13: The refactored class metrics for the Coupon class

It is important to note that as it is a controller class, a high RFC is expected, as it depends on various helper classes, services, repositories, etc.

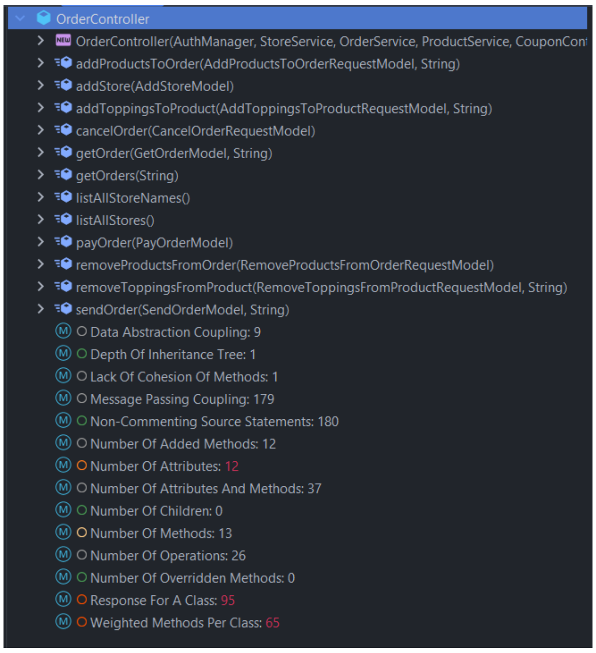
There are several ways to improve the current design of the ”OrderController” class.

Firstly, by refactoring the connection between the ”ModifyOrderService” and ”OrderController” classes, a significant amount of duplicated code can be removed by calling methods from the other class. An example of this is the *sendOrder* method, which can greatly reduce its code by calling the *addProductsToOrder* method. This change will greatly reduce the RFC.

Secondly, the store methods were moved to their own controller. Initially, when there were only a few methods, it was planned to keep all controller methods in the same controller. However, as more methods were added, it was determined that one controller was not sufficient, and a ”CouponController” was created. However, as the store methods were already made, it was forgotten to also create a ”StoreController”. This has now been added.

Finally, some unused and unnecessary parameters have also been removed, as the class had an excessive number of them. An example is the Coupon Controller parameter, which was never used after the constructor.

The results of the refactoring are illustrated in Figure 15. As shown, the metrics of ”OrderController” are much better now. We reduced by 36% the RFC and by nearly 20% the WMC metric.



```
OrderController
  OrderController(AuthManager, StoreService, OrderService, ProductService, CouponCon
  addProductsToOrder(AddProductsToOrderRequestModel, String)
  addStore(AddStoreModel)
  addToppingsToProduct(AddToppingsToProductRequestModel, String)
  cancelOrder(CancelOrderRequestModel)
  getOrder(GetOrderModel, String)
  getOrders(String)
  listAllStoreNames()
  listAllStores()
  payOrder(PayOrderModel)
  removeProductsFromOrder(RemoveProductsFromOrderRequestModel)
  removeToppingsFromProduct(RemoveToppingsFromProductRequestModel, String)
  sendOrder(SendOrderModel, String)
  Data Abstraction Coupling: 9
  Depth Of Inheritance Tree: 1
  Lack Of Cohesion Of Methods: 1
  Message Passing Coupling: 179
  Non-Commenting Source Statements: 180
  Number Of Added Methods: 12
  Number Of Attributes: 12
  Number Of Attributes And Methods: 37
  Number Of Children: 0
  Number Of Methods: 13
  Number Of Operations: 26
  Number Of Overridden Methods: 0
  Response For A Class: 95
  Weighted Methods Per Class: 65
```

Figure 14: The Order Controller class without refactoring

2.3 ModifyOrderService class

Another class that we can improve in the Order domain is ”ModifyOrderService”. The values for WMC, RFC, and NOM were flagged by the tool as really high, with which we agreed as the number were very far from the threshold given to us and the class itself contained too many lines of code and methods. This is shown in Figure 16.

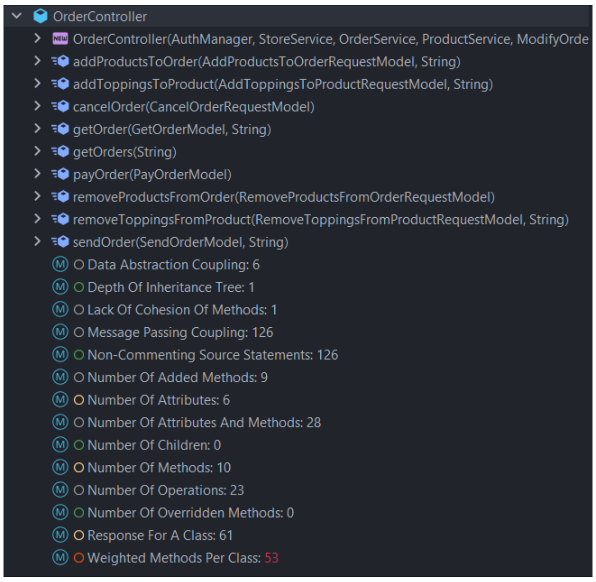
The main problem of this class was the sheer size of it. Therefore, we found a way to split the functionality into three separate classes. We extracted the methods for adding products and toppings to an order into AddToOrderService and other methods that didn't really belong into either one into OrderHelperService which is being used by both AddToOrderService and ModifyOrderService.

By doing this split and also refactoring a few methods with poor metrics, we reached a much improved result as shown in Figures 17, 18,19. All three metrics identified earlier were reduced into much better numbers.

2.4 Order class

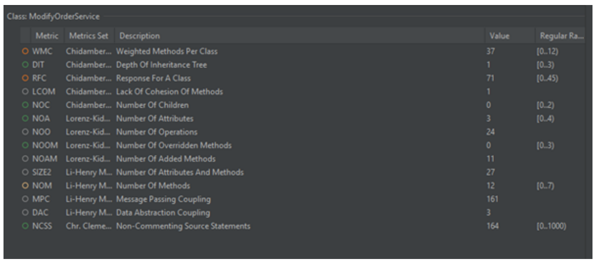
Prior to refactoring, the class metrics for the Order class highlighted the number of methods as well as the number of attributes and weighted methods per class metric as being very high, as we can see in figure 20.

The class describes a database table, thus the number of attributes represents the actual number of columns of the ”ORDERS” table. All these metrics combined indicated that the complexity and sheer size of the class was too large, requiring to refactor the database table and interaction of this table with



```
OrderController
  OrderController(AuthManager, StoreService, OrderService, ProductService, ModifyOrde
  addProductsToOrder(AddProductsToOrderRequestModel, String)
  addToppingsToProduct(AddToppingsToProductRequestModel, String)
  cancelOrder(CancelOrderRequestModel)
  getOrder(GetOrderModel, String)
  getOrders(String)
  payOrder(PayOrderModel)
  removeProductsFromOrder(RemoveProductsFromOrderRequestModel)
  removeToppingsFromProduct(RemoveToppingsFromProductRequestModel, String)
  sendOrder(SendOrderModel, String)
  Data Abstraction Coupling: 6
  Depth Of Inheritance Tree: 1
  Lack Of Cohesion Of Methods: 1
  Message Passing Coupling: 126
  Non-Commenting Source Statements: 126
  Number Of Added Methods: 9
  Number Of Attributes: 6
  Number Of Attributes And Methods: 28
  Number Of Children: 0
  Number Of Methods: 10
  Number Of Operations: 23
  Number Of Overridden Methods: 0
  Response For A Class: 61
  Weighted Methods Per Class: 53
```

Figure 15: The refactored Order Controller class



Class: ModifyOrderService

Metric	Metrics Set	Description	Value	Regular Ra...
WMC	Chidamber...	Weighted Methods Per Class	37	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	71	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	0	[0..2)
NOA	Lorenz-Kid...	Number Of Attributes	3	[0..4)
NOO	Lorenz-Kid...	Number Of Operations	24	
NOOM	Lorenz-Kid...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Kid...	Number Of Added Methods	11	
SIZE2	Li-Henry M...	Number Of Attributes And Methods	27	
NOM	Li-Henry M...	Number Of Methods	12	[0..7)
MPC	Li-Henry M...	Message Passing Coupling	161	
DAC	Li-Henry M...	Data Abstraction Coupling	3	
NCSS	Chr. Cleme...	Non-Commenting Source Statements	164	[0..1000)

Figure 16: The Modify Order Service class without refactoring

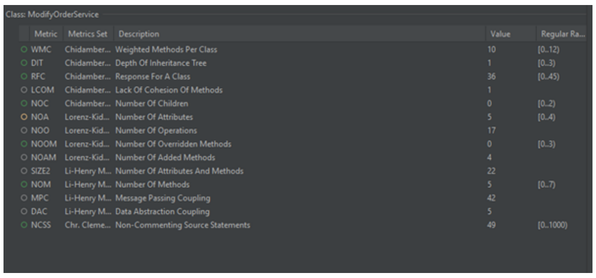
the rest of the system.

Besides attributes and plenty of getters and setters, the class also contained functional methods that interacted with the database.The class was analyzed manually, and we observed that these methods were unnecessarily contained inside of it as well as the fact that there were too many columns and attributes, which led to higher database complexity.

Thus, the main points of refactoring the Order class become evident, refactoring the database table to use less columns by changing the data stored within it, extracting methods that operate on the database to a separate database utilities class and removing redundant or never used methods. These arose from the coding of the initial prototype, when methods and columns were added and removed constantly during development.

Methods that were redundant and not used, like the 2 create(...) methods were deleted, as they were only used in testing and the constructors act in the same way.

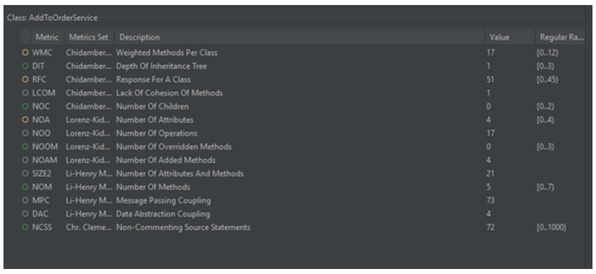
The utility methods, for example pay(...), computePrice(...), add/deleteProduct and Products(...) that operate on an order were all refactored by being extracted to a new class called OrderUtils, and made



Class: ModifyOrderService

	Metric	Metrics Set	Description	Value	Regular Ra...
○	WMC	Chidamber...	Weighted Methods Per Class	10	[0..12)
○	DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
○	RFC	Chidamber...	Response For A Class	36	[0..45)
○	LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
○	NOC	Chidamber...	Number Of Children	0	[0..2)
○	NOA	Lorenz-Kid...	Number Of Attributes	5	[0..4)
○	NOO	Lorenz-Kid...	Number Of Operations	17	
○	NOOM	Lorenz-Kid...	Number Of Overridden Methods	0	[0..3)
○	NOAM	Lorenz-Kid...	Number Of Added Methods	4	
○	SIZE2	Li-Henry M...	Number Of Attributes And Methods	22	
○	NOM	Li-Henry M...	Number Of Methods	5	[0..7)
○	MPC	Li-Henry M...	Message Passing Coupling	42	
○	DAC	Li-Henry M...	Data Abstraction Coupling	5	
○	NCSS	Chr. Cleme...	Non-Commenting Source Statements	49	[0..1000)

Figure 17: The refactored Modify Order Service class



Class: AddToOrderService

	Metric	Metrics Set	Description	Value	Regular Ra...
○	WMC	Chidamber...	Weighted Methods Per Class	17	[0..12)
○	DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
○	RFC	Chidamber...	Response For A Class	51	[0..45)
○	LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
○	NOC	Chidamber...	Number Of Children	0	[0..2)
○	NOA	Lorenz-Kid...	Number Of Attributes	4	[0..4)
○	NOO	Lorenz-Kid...	Number Of Operations	17	
○	NOOM	Lorenz-Kid...	Number Of Overridden Methods	0	[0..3)
○	NOAM	Lorenz-Kid...	Number Of Added Methods	4	
○	SIZE2	Li-Henry M...	Number Of Attributes And Methods	21	
○	NOM	Li-Henry M...	Number Of Methods	5	[0..7)
○	MPC	Li-Henry M...	Message Passing Coupling	73	
○	DAC	Li-Henry M...	Data Abstraction Coupling	4	
○	NCSS	Chr. Cleme...	Non-Commenting Source Statements	72	[0..1000)

Figure 18: The new AddToOrderService class

static, so that they can more easily be used when needed. This change not only makes the code more modular and maintainable, but it makes more sense, since they are not logically methods that should be contained in a class describing a database table. Also, it lowers the metric of number of methods.

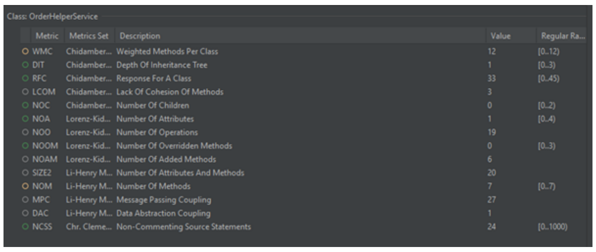
To solve the issue of the number of attributes, a new class was created, named OrderData. This class combines database columns / attributes of customerId, storeId, status, deliveryTime, price and moneySaved into one data type. Now all of these have been merged into one column in the new Database table. Furthermore, this class contains all getters and setters needed in the project for these parameters. Moving all these fields and their subsequent accessor methods to a new class greatly reduced the number of methods and attributes of the Order class.

One issue that arises from the refactoring of the database table of orders was the need for refactoring the tests and the project to accommodate the change. Another class was required to be created, the OrderDataConverter, that is required for storing the object and retrieving it from the database.

The refactoring process led to 16 files being changed to accommodate it, but greatly optimizing the metrics in the process. As we can observe from the 21 figure, the number of methods was reduced by 62%, from 29 to 11, the attributes were reduced by 60% from 10 to 4, and the weighted methods per class metric was reduced from 44 to 15, an improvement of 65%.

The metrics of the added classes can be observed in 23, 22, and noticeably no metric is out of order.

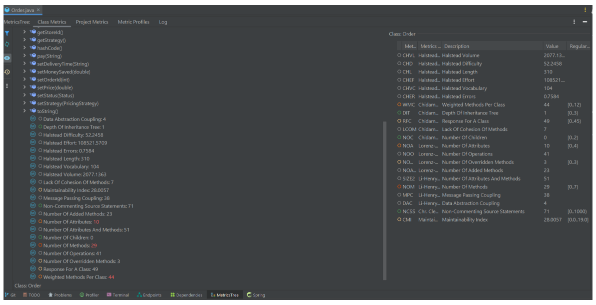
Due to the improvement of the metrics and overall efficiency of the project, the refactoring of Order was very beneficial



Class: OrderHelperService

Metric	Metrics Set	Description	Value	Regular Ra...
WMC	Chidamber...	Weighted Methods Per Class	12	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	33	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	3	
NOC	Chidamber...	Number Of Children	0	[0..2)
NOA	Lorenz-Kid...	Number Of Attributes	1	[0..4)
NOO	Lorenz-Kid...	Number Of Operations	19	
NOOM	Lorenz-Kid...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Kid...	Number Of Added Methods	6	
SIZE2	Li-Henry M...	Number Of Attributes And Methods	20	
NOM	Li-Henry M...	Number Of Methods	7	[0..7)
MPC	Li-Henry M...	Message Passing Coupling	27	
DAC	Li-Henry M...	Data Abstraction Coupling	1	
NCSS	Chr. Cleme...	Non-Commenting Source Statements	24	[0..1000)

Figure 19: The new OrderHelperService class



Order.java

MetricsTree: Class Metrics Project Metrics Metric Profiles Log

```
> getStoreId()
> getStrategy()
> hashCode()
> pay(String)
> setDeliveryTime(String)
> setMoneySaved(double)
> setOrderId(int)
> setPrice(double)
> setStatus(Status)
> setStrategy(PricingStrategy)
> toString()
  Data Abstraction Coupling: 4
  Depth Of Inheritance Tree: 1
  Halstead Difficulty: 52.2458
  Halstead Effort: 108521.5709
  Halstead Errors: 0.7584
  Halstead Length: 310
  Halstead Vocabulary: 104
  Halstead Volume: 2077.1363
  Lack Of Cohesion Of Methods: 7
  Maintainability Index: 28.0057
  Message Passing Coupling: 38
  Non-Commenting Source Statements: 71
  Number Of Added Methods: 23
  Number Of Attributes: 10
  Number Of Attributes And Methods: 51
  Number Of Children: 0
  Number Of Methods: 29
  Number Of Operations: 41
  Number Of Overridden Methods: 3
  Response For A Class: 49
  Weighted Methods Per Class: 44
Class: Order
```

Class: Order

Met...	Metrics ...	Description	Value	Regular...
CHVL	Halstead...	Halstead Volume	2077.13...	
CHD	Halstead...	Halstead Difficulty	52.2458	
CHL	Halstead...	Halstead Length	310	
CHEF	Halstead...	Halstead Effort	108521....	
CHVC	Halstead...	Halstead Vocabulary	104	
CHER	Halstead...	Halstead Errors	0.7584	
WMC	Chidam...	Weighted Methods Per Class	44	[0..12)
DIT	Chidam...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidam...	Response For A Class	49	[0..45)
LCOM	Chidam...	Lack Of Cohesion Of Methods	7	
NOC	Chidam...	Number Of Children	0	[0..2)
NOA	Lorenz-...	Number Of Attributes	10	[0..4)
NOO	Lorenz-...	Number Of Operations	41	
NO...	Lorenz-...	Number Of Overridden Methods	3	[0..3)
NOA...	Lorenz-...	Number Of Added Methods	23	
SIZE2	Li-Henry...	Number Of Attributes And Methods	51	
NOM	Li-Henry...	Number Of Methods	29	[0..7)
MPC	Li-Henry...	Message Passing Coupling	38	
DAC	Li-Henry...	Data Abstraction Coupling	4	
NCSS	Chr. Cle...	Non-Commenting Source Statements	71	[0..1000)
CMI	Maintai...	Maintainability Index	28.0057	[0.0..19.0]

Figure 20: Order class metrics prior to refactoring

2.5 AppUser Class

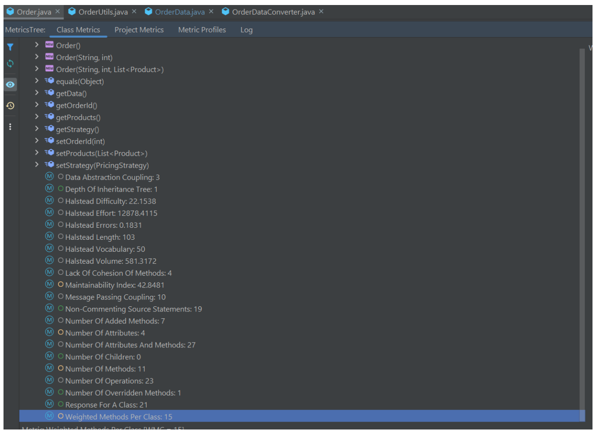
The next class we refactored was AppUser class. Using the metrics tree, we could find that the number of methods and overridden methods are too high. Additionally, the weighted method per class was high compared to the normal boundary, which can be reflected in Figure 24.

The primary goal of this refactoring was to remove unneeded methods, classes, and interfaces. We made the decision not to apply the decorator design pattern (we used two different design patterns) which was the basis for the design of the Role class in the User microservice. We chose not to use decorator because we discovered that it was a poor fit for our application. Our old implementation of the decorator design pattern still remained though and was essentially obsolete and over complicated; it contained some unnecessary parts such as the RoleType class and RoleDefaultFunctionalityInterface interface. We have removed and relocated them into functions such as the Role class and UserRole/StoreOwnerRole/AdminRole class.

Furthermore, we opted to shift some AppUser functionalities to other classes. We chose to assign the functionality based on allergens to the Allergens class and remove it from the AppUser since, in particular, functionalities related to allergens in AppUser class were duplicated in the allergens class. The AppUser class had improved statistical measures in the metric tree after the refactoring, as shown in Figure 25.

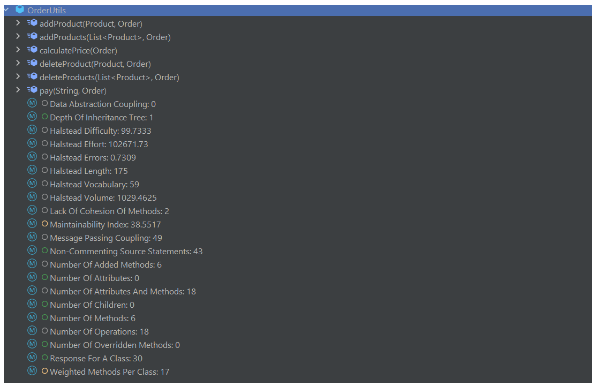
2.6 CouponTest

The class CouponTest which included all unit tests for the Coupon class in the Order Microservice had some unwanted metrics as identified by the tool that we used (can be seen in Figure 26). The main problems were the number of methods, which also increased the WMC and RFC.



```
Order()
Order(String, int)
Order(String, int, List<Product>)
equals(Object)
getData()
getOrderId()
getProducts()
getStrategy()
setOrderId(int)
setProducts(List<Product>)
setStrategy(PricingStrategy)
Data Abstraction Coupling: 3
Depth Of Inheritance Tree: 1
Halstead Difficulty: 22.1538
Halstead Effort: 12878.4115
Halstead Errors: 0.1831
Halstead Length: 103
Halstead Vocabulary: 50
Halstead Volume: 581.3172
Lack Of Cohesion Of Methods: 4
Maintainability Index: 42.8481
Message Passing Coupling: 10
Non-Commenting Source Statements: 19
Number Of Added Methods: 7
Number Of Attributes: 4
Number Of Attributes And Methods: 27
Number Of Children: 0
Number Of Methods: 11
Number Of Operations: 23
Number Of Overridden Methods: 1
Response For A Class: 21
Weighted Methods Per Class: 15
```

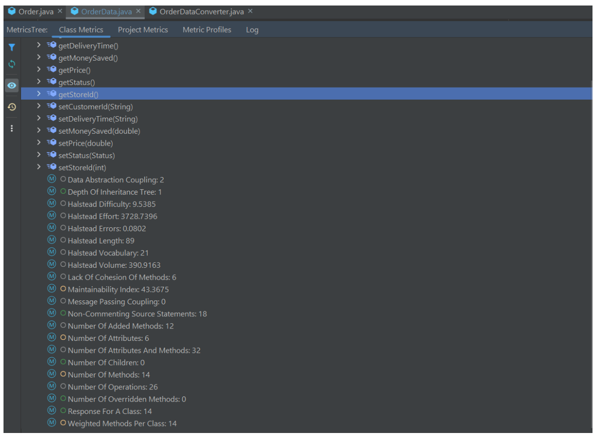
Figure 21: Order class metrics after refactoring



```
OrderUtils
addProduct(Product, Order)
addProducts(List<Product>, Order)
calculatePrice(Order)
deleteProduct(Product, Order)
deleteProducts(List<Product>, Order)
pay(String, Order)
Data Abstraction Coupling: 0
Depth Of Inheritance Tree: 1
Halstead Difficulty: 99.7333
Halstead Effort: 102671.73
Halstead Errors: 0.7309
Halstead Length: 175
Halstead Vocabulary: 59
Halstead Volume: 1029.4625
Lack Of Cohesion Of Methods: 2
Maintainability Index: 38.5517
Message Passing Coupling: 49
Non-Commenting Source Statements: 43
Number Of Added Methods: 6
Number Of Attributes: 0
Number Of Attributes And Methods: 18
Number Of Children: 0
Number Of Methods: 6
Number Of Operations: 18
Number Of Overridden Methods: 0
Response For A Class: 30
Weighted Methods Per Class: 17
```

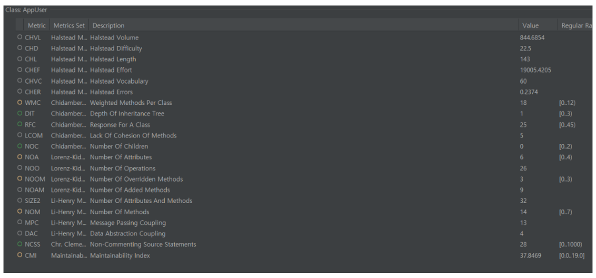
Figure 22: Class metrics for OrderUtils class

To combat their problems, we decided to split the class into two. One with the very simple methods and another one with the more complex ones. Doing this improved all the metrics to such a point that we were satisfied with it. The new values can be seen in Figure 27.



```
getDeliveryTime()
getMoneySaved()
getPrice()
getStatus()
getStoreId()
setCustomerId(String)
setDeliveryTime(String)
setMoneySaved(double)
setPrice(double)
setStatus(Status)
setStoreId(int)
Data Abstraction Coupling: 2
Depth Of Inheritance Tree: 1
Halstead Difficulty: 9.5385
Halstead Effort: 3728.7396
Halstead Errors: 0.0802
Halstead Length: 89
Halstead Vocabulary: 21
Halstead Volume: 390.9163
Lack Of Cohesion Of Methods: 6
Maintainability Index: 43.3675
Message Passing Coupling: 0
Non-Commenting Source Statements: 18
Number Of Added Methods: 12
Number Of Attributes: 6
Number Of Attributes And Methods: 32
Number Of Children: 0
Number Of Methods: 14
Number Of Operations: 26
Number Of Overridden Methods: 0
Response For A Class: 14
Weighted Methods Per Class: 14
```

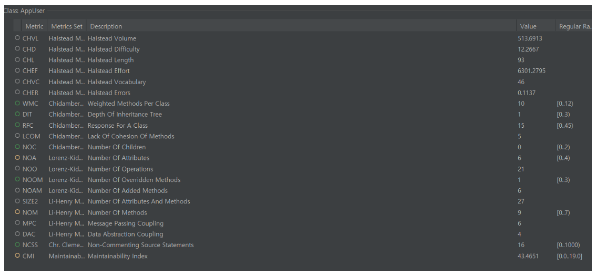
Figure 23: Class metrics for OrderData class



Class: AppUser

Metric	Metrics Set	Description	Value	Regular Ra
CHVL	Halstead M...	Halstead Volume	844.6854	
CHD	Halstead M...	Halstead Difficulty	22.5	
CHL	Halstead M...	Halstead Length	143	
CHEF	Halstead M...	Halstead Effort	19005.4205	
CHVC	Halstead M...	Halstead Vocabulary	60	
CHER	Halstead M...	Halstead Errors	0.2374	
WMC	Chidamber...	Weighted Methods Per Class	18	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	25	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	5	
NOC	Chidamber...	Number Of Children	0	[0..2)
NOA	Lorenz-Kid...	Number Of Attributes	6	[0..4)
NOO	Lorenz-Kid...	Number Of Operations	26	
NOOM	Lorenz-Kid...	Number Of Overridden Methods	3	[0..3)
NOAM	Lorenz-Kid...	Number Of Added Methods	9	
SIZE2	Li-Henry M...	Number Of Attributes And Methods	32	
NOM	Li-Henry M...	Number Of Methods	14	[0..7)
MPC	Li-Henry M...	Message Passing Coupling	13	
DAC	Li-Henry M...	Data Abstraction Coupling	4	
NCSS	Chr. Cleme...	Non-Commenting Source Statements	28	[0..1000)
CMI	Maintainab...	Maintainability Index	37.8469	[0.0..19.0]

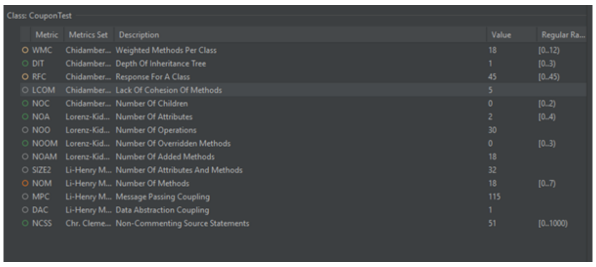
Figure 24: AppUser class metrics before the refactoring



Class: AppUser

Metric	Metrics Set	Description	Value	Regular Ra...
CHVL	Halstead M...	Halstead Volume	513.6913	
CHD	Halstead M...	Halstead Difficulty	12.2667	
CHL	Halstead M...	Halstead Length	93	
CHEF	Halstead M...	Halstead Effort	6301.2795	
CHVC	Halstead M...	Halstead Vocabulary	46	
CHER	Halstead M...	Halstead Errors	0.1137	
WMC	Chidamber...	Weighted Methods Per Class	10	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	15	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	5	
NOC	Chidamber...	Number Of Children	0	[0..2)
NOA	Lorenz-Kid...	Number Of Attributes	6	[0..4)
NOO	Lorenz-Kid...	Number Of Operations	21	
NOOM	Lorenz-Kid...	Number Of Overridden Methods	1	[0..3)
NOAM	Lorenz-Kid...	Number Of Added Methods	6	
SIZE2	Li-Henry M...	Number Of Attributes And Methods	27	
NOM	Li-Henry M...	Number Of Methods	9	[0..7)
MPC	Li-Henry M...	Message Passing Coupling	6	
DAC	Li-Henry M...	Data Abstraction Coupling	4	
NCSS	Chr. Cleme...	Non-Commenting Source Statements	16	[0..1000)
CMI	Maintainab...	Maintainability Index	43.4651	[0.0..19.0]

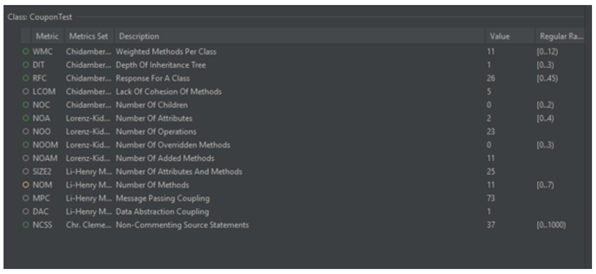
Figure 25: AppUser class metrics after the refactoring



Class: CouponTest

Metric	Metrics Set	Description	Value	Regular Ra...
WMC	Chidamber...	Weighted Methods Per Class	18	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	45	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	5	
NOC	Chidamber...	Number Of Children	0	[0..2)
NOA	Lorenz-Kid...	Number Of Attributes	2	[0..4)
NOO	Lorenz-Kid...	Number Of Operations	30	
NOOM	Lorenz-Kid...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Kid...	Number Of Added Methods	18	
SIZE2	Li-Henry M...	Number Of Attributes And Methods	32	
NOM	Li-Henry M...	Number Of Methods	18	[0..7)
MPC	Li-Henry M...	Message Passing Coupling	115	
DAC	Li-Henry M...	Data Abstraction Coupling	1	
NCSS	Chr. Cleme...	Non-Commenting Source Statements	51	[0..1000)

Figure 26: CouponTest class metrics prior to refactoring



Class: CouponTest

Metric	Metrics Set	Description	Value	Regular Ra...
WMC	Chidamber...	Weighted Methods Per Class	11	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	26	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	5	
NOC	Chidamber...	Number Of Children	0	[0..2)
NOA	Lorenz-Kid...	Number Of Attributes	2	[0..4)
NOO	Lorenz-Kid...	Number Of Operations	23	
NOOM	Lorenz-Kid...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Kid...	Number Of Added Methods	11	
SIZE2	Li-Henry M...	Number Of Attributes And Methods	25	
NOM	Li-Henry M...	Number Of Methods	11	[0..7)
MPC	Li-Henry M...	Message Passing Coupling	73	
DAC	Li-Henry M...	Data Abstraction Coupling	1	
NCSS	Chr. Cleme...	Non-Commenting Source Statements	37	[0..1000)

Figure 27: CouponTest class metrics after refactoring