

## SEM Assignment 2 - Code Metrics

### Group 52

#### Part 1: Overview of Problematic Classes with Code Smells:

For this assignment we have chosen to use CodeMR.

We identified that the *LoggerService* class needed to be improved, since it gave a high score on LoC (Lack of Cohesion) - 0.833.

The reason why we decided to fix this issue is due to maintenance.

With a high LoC score it is difficult to make changes only in one place and you should refactor your whole class.

Another class we chose for refactoring was the *FoodForwardController*, we took into consideration that the coupling was relatively high, compared to other classes. We wanted to make this class as independent as possible on other modules. Coupling is also closely related with LoC, which is one of the most important things for the maintainability of our product.

We noticed that the *AuthenticationController* class also has smells; we primarily noticed medium to high coupling. Next to this, it has a lack of cohesion and contains too much code. Classes with high coupling require more maintenance, because there exist more dependencies, and that is why we decided to also refactor this class.

The *CheckConflicts* class suffers from medium complexity and medium coupling. These are some issues we want to avoid so we also opted to refactor this class in order to get lower complexity and less coupling.

Another class which suffered from a couple of small code smells was the *UserAuthenticationService*. It has low to medium complexity, coupling and size. This class handles a lot of functionality so we also decided to include this class in our refactoring process.

For methods we isolated *forwardToHouseMicroservice()* and *forwardToFoodMicroservice()*, as such which need to be improved. We noticed that in those 2 methods, we had medium-high coupling and low-medium size. We wanted to improve the readability and maintainability of these methods.

After computing the code metrics for the House Management microservice we found that although nearly all methods and classes had low for most metrics, there were a couple of methods in the *StudentHouseController* that had medium-high Coupling, something to be improved.

For the FoodManagement we achieved mainly low and medium-low results from the very beginning on. The only thing we achieved was a medium cohesion, so we looked into it. However after a couple of tries we realized that by changing cohesion at one place our code quality decreased for other classes. Therefore we focused on different issues. We noticed that *takePortions* and *takePortionsTogether* in the *FoodProductController* performed almost identical code, so we extracted the functionality and created a method that combines their functionality, to avoid duplicate code.

In the same class, we also found that we had some methods that were hard to read, mainly caused by nested if-statements. As this is recognized as an object-oriented abusers, and decreases readability of the method, we tried to get rid of as many of them as possible in the methods: *updateFoodProducts*, *deleteFoodProducts*, *getStorage*.

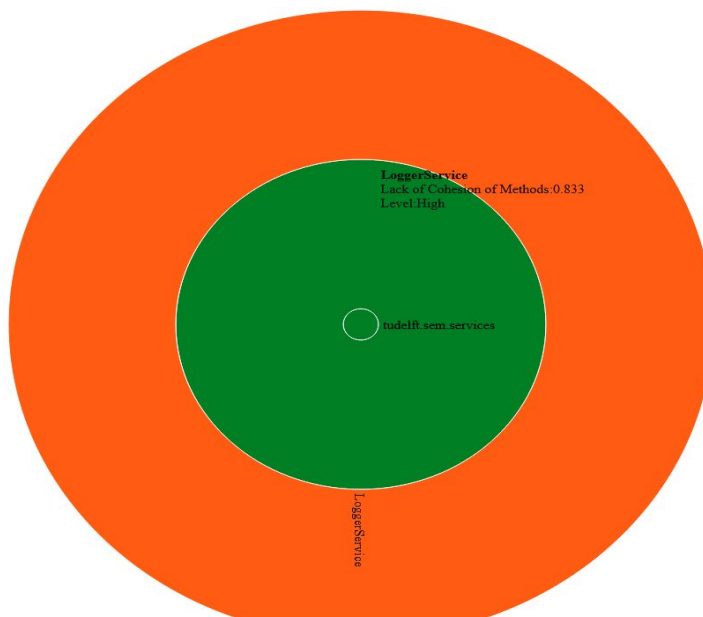
## Part 2: Refactoring of classes and methods

### Improved classes:

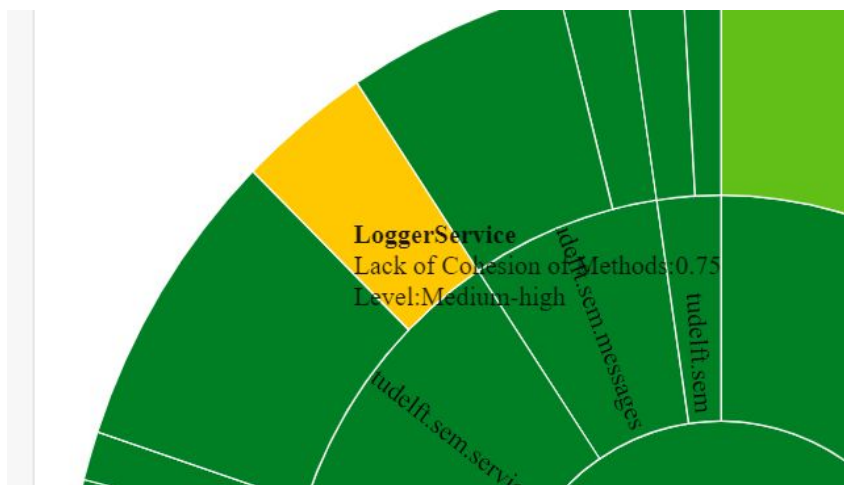
#### 1 - LoggerService - API Gateway

- I. Metrics Before: High Lack of Cohesion of Methods - 0.833
- II. Metrics After: Medium-High Lack of Cohesion of Methods - 0.75

View from the HTML report:



View from the HTML report:



#### III. Types of refactoring used:

The LoggerService suffered from High Lack of Cohesion of Methods partly due to the fact there was a static field in the LoggerService class which was only used by its private constructor and not any of its other public methods. In order to improve this, we realised we can just move the aforementioned static field to a configuration class, which was more

appropriate for the object. As a result, we improved the Lack of Cohesion of Methods from High to Medium, which made it more modularly independent and maintainable.

## 2 - FoodForwardController - API Gateway

### I. Metrics Before:

- A. Low Complexity
- B. Medium Coupling
- C. Medium Size
- D. Low Lack of Cohesion

- View from IntelliJ:

FoodForwardController	low	low-medium	low-medium	low
-----------------------	-----	------------	------------	-----

- View from the HTML Report:

CODEMR										
DASHBOARD										
DETAILED METRIC LIST										
METRIC EXPLANATIONS										
CODEMR GRAPHS										
List of all classes (#13)										
ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	AuthenticationFor...	■	■	■	■	152	low-medium	low-medium	low	low-medium
2	StudentHouseForwa...	■	■	■	■	55	low	low-medium	low	low-medium
3	FoodForwardContro...	■	■	■	■	52	low	low-medium	low	low-medium
4	Microservice	■	■	■	■	7	low-medium	low	low	low
5	ParameterIdsFilter	■	■	■	■	31	low	low	low	low
6	RequestService	■	■	■	■	31	low	low	low	low
7	FormatRequestsAda...	■	■	■	■	27	low	low	low	low
8	HttpMessages	■	■	■	■	23	low	low	low	low
9	RequestHouseLink	■	■	■	■	21	low	low	low	low

### II. Metrics After:

- A. Medium Complexity
- B. Low Coupling
- C. Low Size
- D. Low Lack of Cohesion

- View from IntelliJ:

nl.tudelft.sem.controllers	low	low-medium	low-medium	low							47
> AuthenticationForwardController	low-medium	low-medium	low-medium	low	10	59	27	1	0	18	
> BaseForwardController	low	low-medium	low	low	7	44	18	1	2	8	
> CheckLoginFilter	low	low	low	low	2	16	11	1	0	6	
> FoodForwardController	low-medium	low	low	low	3	47	1	2	0	5	
> ParameterIdsFilter	low	low	low	low	2	13	8	1	0	5	
> StudentHouseForwardController	low-medium	low	low	low-medium	4	53	3	2	0	5	

- View from the HTML Report:

List of all classes (#15)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	AuthenticationFor...					117	low-medium	low-medium	low	low-medium
2	BaseForwardContro...					36	low	low-medium	low	low
3	StudentHouseForwa...					23	low-medium	low	low-medium	low
4	FoodForwardContro...					19	low-medium	low	low	low
5	Microservice					7	low-medium	low	low	low
6	CheckLoginFilter					31	low	low	low	low
7	ParameterIdsFilter					31	low	low	low	low
8	RequestService					31	low	low	low	low
9	FormatRequestsAda...					27	low	low	low	low

### III. Types of refactoring used:

Our refactoring managed to decrease the Size of the FoodForwardController class from *Medium* to *Low*, to decrease the Coupling from *Medium* to *Low*, while in the meantime it increased the Complexity from *Low* to *Medium*. However, it is logical to think that the latter is a normal consequence from the former. Our Controller classes are the entry point for processing requests on the Gateway and because this usually requires multiple different operations to be performed that span many classes, we cannot easily achieve Low Coupling and Low Complexity at the same time. Either the controller class will perform most of the operations itself, which makes it less complex but requires more external classes which coupling, or it will delegate most of the work to some external classes which will reduce coupling but increase complexity.

In order to reduce the size of the class, we used inheritance and splitting of business logic into separate classes.




1. First, we extracted the common code for making requests to other application components in a single RequestService class.

2. After that, we created a `BaseForwardController` base class, which would implement the base logic for forwarding a received request, checking the status code of the response and handling exceptions in the process. Then we refactored our `FoodForwardController` and `StudentHouseForwardController` classes to inherit from this base class. In addition, we utilized the aforementioned `RequestService` class in the base class to have a proper division of responsibilities in it and to further reduce its size.
3. Eventually, both `FoodForwardController` and `StudentHouseForwardController` were improved with regards to size and coupling, while their complexity was slightly increased because of the increased number of methods. However, as a result we created a highly reusable code base that makes the Gateway component more scalable in case more Microservices are added in the future.

### 3 - AuthenticationController - Authentication

#### I. Metrics Before:

- A. Low complexity
- B. Medium-high coupling
- C. Low-medium lack of cohesion
- D. Low-medium size

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	AuthenticationCon...					63	low	medium-high	low-medium	low-medium

#### II. Metrics After:

- A. Low complexity
- B. Low-medium coupling
- C. Low lack of cohesion
- D. Low size

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	AuthenticationCon...					50	low	low-medium	low	low

#### III. Types of refactoring used:

- A. In order to obtain lower coupling in the *AuthenticationController* class, we have removed the logic that initializes the chain of responsibility to the *UserAuthenticationService* class, where it makes more sense to initialize it. Of course, this would not fix the issue rather than moving it to a different location. Therefore, we have redistributed the initialization procedure to another class in the *Registrator* package, called *InitializeChain*. This class now solely initializes the chain and runs its *handle* methods and is not involved with any other classes.
- B. In addition, we have compacted statements in the entire class so the class becomes more understandable and has less lines of code. The amount of code was already also reduced by moving the chain of responsibility to other, more appropriate, classes.

- C. Next to this, we have made sure that all the methods carry similar signatures. This way we can make sure our lack of cohesion reaches the lowest level. Beforehand, some methods had separate usernames and passwords as parameters. We have improved the cohesion by changing the parameters of the methods to accept a *sem.model.User* entity, instead of accepting the detached parameters of this same entity class.

#### 4 - CheckConflicts - Authentication

##### I. Metrics Before:

- A. Low-medium coupling
- B. Low-medium complexity
- C. Low lack of cohesion
- D. Low size

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
2	CheckConflicts					13	low-medium	low-medium	low	low

##### II. Metrics After:

- A. Low coupling
- B. Low complexity
- C. Low lack of cohesion
- D. Low size

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	CheckConflicts					12	low	low	low	low

##### III. Types of refactoring used:

- A. We had low to medium coupling, and this was partly caused by encrypting the passwords locally in the class. We have delegated this to the *sem.model.User* class. After doing this, we received low coupling.
- B. The *CheckConflicts* class has medium complexity, so we split it into multiple methods, which caused the class to become more readable, and it has lower complexity because of this.

#### 5 - UserAuthenticationService - Authentication

##### I. Metrics Before:




- A. Low-medium complexity
- B. Low-medium coupling
- C. Low lack of cohesion
- D. Low-medium size

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	UserAuthenticatio...					79	low-medium	low-medium	low	low-medium

##### II. Metrics After:

- A. Low complexity

- B. Low-medium coupling
- C. Low lack of cohesion
- D. Low-medium size

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	UserAuthenticatio...					59	low	low-medium	low	low-medium

### III. Types of refactoring used:

- A. We noticed a couple of code smells in this class, but were unfortunately unable to repair all of them. We have shifted more logic regarding the chain of responsibility to the *InitializeChain* class. In addition, we have moved all logic regarding encoding and matching passwords to the User class, where these operations are more suitable. This caused the complexity to drop to low levels. The *BCryptPasswordEncoder* is no longer used anywhere in the class.
- B. After investigating, we noticed that we were unable to decrease the amount of coupling. This was caused by the chain of responsibility being called by the class. Because this is a necessary operation, we were unable to repair this, but we believe we have minimized the impact on the class.
- C. Finally, the method *getIdsForUsernames* had the largest impact on the coupling and size. Attempts at refactoring this method would not work out, as its code can not be moved to another class. Moving the code would not make any sense, since the repository is used in a direct manner.



## Improved Methods:

### 1,2 - API Gateway:

- **FoodForwardController - forwardToFoodService()**
- **StudentHouseForwardController - forwardToHouseManagementService()**

#### I. Metrics Before:

- A. Low-Medium Size
- B. Medium-High Coupling

- View from IntelliJ:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
template					
> nl.tudelft.sem	low	low	low	low	
> nl.tudelft.sem.communication	low	low	low	low	
> nl.tudelft.sem.controllers	low	low	low-medium	low	
> AuthenticationForwardController	low-medium	low-medium	low-medium	low	10
> FoodForwardController	low	low-medium	low-medium	low	8
FoodForwardController( HttpClient	low	low	low	low	0
forwardToFoodService( HttpSe	low	medium-high	low-medium	low	8
final transient httpClient : Httpc					
static final LOGGER_SERVICE : L					
> ParameterIdsFilter	low	low	low	low	2
> StudentHouseForwardController	low	low-medium	low-medium	low	9
StudentHouseForwardController	low	low	low	low	1
forwardToHouseManagementS	low	medium-high	low-medium	low	9
final transient httpClient : Httpc					
final transient requestHouseLin					
static final LOGGER_SERVICE : L					
> nl.tudelft.sem.messages	low	low	low	low	
> nl.tudelft.sem.services	low	low	low	low	

#### II. Metrics After:

- A. Low Size
- B. Low Coupling

- View from IntelliJ:

name	Complexity	Coupling	Size	Lack of Cohesion	CBO
final transient logger : LoggerS					
final transient requestService : f					
> CheckLoginFilter	low	low	low	low	2
> FoodForwardController	low-medium	low	low	low	3
FoodForwardController( Reque	low	low	low	low	1
forwardToFoodService( HttpSe	low	low	low	low	1
getMicroservice( ): Microservice	low	low	low	low	1
processResponse( HttpRespons	low	low	low	low	0
receivedRequestMessage( ): Str	low	low	low	low	0
> LoginForwardController	low-medium	low-medium	low	low-medium	8
> ParameterIdsFilter	low	low	low	low	2
> RegisterForwardController	low-medium	low-medium	low	low	6
> StudentHouseForwardController	low-medium	low	low	low-medium	4
StudentHouseForwardControll	low	low	low	low	2
forwardToHouseManagementS	low	low	low	low	1
getMicroservice( ): Microservice	low	low	low	low	1
processResponse( HttpRespons	low	low	low	low	1
receivedRequestMessage( ): Str	low	low	low	low	0
final transient requestHouseLin					
> UnregisterForwardController	low-medium	low-medium	low	low	7
> nl.tudelft.sem.messages	low	low	low	low	

### III. Types of refactoring used:

As explained in the *Types of refactoring used* in the [FoodForwardController](#) class section, we used inheritance to reduce the size of our controller classes forwarding requests. This in turn reduced the size of these two methods to Low and their Coupling from Medium-High to Low. Even though in the Base controller class there is one method with Low-Medium Coupling, overall the class and its methods are small, coherent and with a small complexity, which is doubtlessly an improvement.

## 3 - FoodProductController - takePortion & takePortionTogether - Food Management

- I. Metrics Before: 63 Lines of code
- II. Metrics After: 42 Lines of code
- III. Types of refactoring used: The code smell found in these methods was duplicate code, since these methods perform very similar tasks. To refactor this, I extracted the part of the method, essentially combining the two methods into one, using some additional if-statements where necessary. Before the refactoring, the combined lines of the two methods was 63. After refactoring, the combined lines of the two methods plus the extra methods is 42 lines.

## 4 - FoodProductController - updateFoodproduct, deleteFoodproduct, getStorage - Food Management

- I. Metrics Before:
  - A. updateFoodproduct: CC(Cyclomatic Complexity) of 5
  - B. deleteFoodproduct: CC of 4
  - C. getStorage: CC of 7
- II. Metrics After:

- A. updateFoodproduct: CC(Cyclomatic Complexity) of 5
  - B. deleteFoodproduct: CC of 3
  - C. getStorage: CC of 7
- III. Types of refactoring used:  
 Since the methods (specifically updateFoodproduct) included nested if statements, they are harder to comprehend, whereas now you have a more direct and clearer view of what is going on, without the functionality being impacted. All methods are now clearer, especially if you do not know the code very well. An if closes sooner, so you know better when one if ends and the according else starts. This fixes the code smell of object-oriented abusers, which was going against the Open-Close principle. In our own words, we can adapt the method more easily now, if functionality is added in the future.

### **5 - StudentHouseController - registerUser**

- I. Metrics Before: Complexity, Size and Lack of Cohesion low; Coupling medium-high.
- II. Metrics After: Complexity, Size and Lack of Cohesion low; Coupling low-medium.
- III. Types of refactoring used:  
 The code smell I found was couplers. The method in StudentHouseController made a lot of calls to houseManagementService, namely to the repositories. To decrease and thus improve the Coupling metric I moved the calls into a method inside houseManagementService, registerUser() with the same parameters as the controller method in StudentHouseController. This way the Coupling metric went from medium-high to low-medium.

### **6 - StudentHouseController - addHousemate**

- I. Metrics Before: Complexity, Size and Lack of Cohesion low; Coupling medium-high.
- II. Metrics After: Complexity, Size and Lack of Cohesion low; Coupling low-medium.
- III. Types of refactoring used:  
 The code smell I found was couplers. In the addHousemate method in the controller class some inefficient calls were made to both the User and StudentHouse class. When a request was made to add someone to a house, the controller method would call the User class to set the their house attribute to the house to join but also made a call to the House class to add the user to the list of residents. I made this more efficient by allowing setters and getters to strictly set and get and creating separate methods for moving in and out, combined in one method in a User or House class instead of needing to call both of them. This didn't change the Coupling metric however so I moved the calls that were made to repositories via houseManagementService to a

dedicated method inside houseManagementService, namely addHousemate() with parameters being a User instance of the current resident and a User instance of the housemate-to-be. This way the Coupling method went from medium-high to low-medium.