Delft University of Technology

Software Engineering Methods
CSE2115

Assignment 2 Group 03B

Authors:
Antonios Barotsis
Nathan Klumpenaar
Dan Sochirca
Rado Todorov
Miloš Ristić
January 15, 2022

Introduction

For this assignment the code submitted previously was refactored using software analytics tools. The tools chosen are: **CodeMR**[1] for calculating class metrics, and Dr Maurício Aniche's **CK calculator**[2] for method metrics.

The thresholds used for all refactorings using CodeMR are that of CodeMR itself which correspond to the low-medium level. For example, CBO (Coupling Between Object Classes) has a threshold of 11.

For all method level refactorings there are no hard thresholds as the metrics used depend on the method (lines of code, cyclomatic complexity..) Therefore the goal here is to bring it down as much as possible.

Class level refactorings

TargetedCompanyOfferService - Offer microservice

Metrics before refactoring:

- **Coupling Between Object Classes:** 13 (threshold 11)
- **Lack of Cohesion of Methods:** 0.778 (threshold 0.4)

Metrics after refactoring:

- **Coupling Between Object Classes:** 9
- **Lack of Cohesion of Methods:** 0.2

The coupling of the class was caused by unnecessary code duplication between this class and the OfferService class. By removing method **saveOffer** and adjusting method **saveOfferWithResponse**, so that it properly uses the super class's method we were able to significantly reduce CBO and LCOM metrics, so that they are below our thresholds. In addition, due to removing an entire method we were able to also reduce the effective lines of code of the class, even though this was not the intial goal.

Exceptions - Offer microservice

Classes:

- LowRatingException
- UserNotFoundException
- UserNotAuthorException

Metrics before refactoring:

- **Depth of Inheritance Tree:** 4 (threshold 3)

Metrics after refactoring:

- **Depth of Inheritance Tree:** 3

The threshold for being in the green for DIT given by CodeMR is 3, which is also the threshold used in refactoring. The refactoring proved to be fairly simple as the only thing that needed to be done was make the Exceptions extend from Exception, instead of RunTimeException. The depth of inheritance tree metric might seem harmless or even useless in this particular case as the complexity doesn't really change, however it forced it so that a call to a method throwing any of these exceptions would be forced to handle the exception, increasing safety.

OfferServices - Offer microservice

Metrics before refactoring:

- **Lack of Tight Class Cohesion:** 0.833 (threshold 0.7)
- **Lack of Cohesion of Methods:** 0.917 (threshold 0.4)

Metrics after refactoring:

- **Lack of Tight Class Cohesion:** 0.667
- **Lack of Cohesion of Methods:** 0.333

The metrics given are for the main **OfferService** class. Here there was a problem that the methods of the class were not cohesive. This was due to the fact that there was a method there purely for communication with another microservice. This method was then moved to the **Utility** class that also already had other methods for this purpose. The **RestTemplate** attribute was removed from the **OfferService** class and moved to the **Utility** class. This was also done for the other offerservices which slightly improved their cohesion as well even though this was not the main focus. This also made it so that all calls to the **Utility** methods required one less argument. In the main **OfferService** class 3 global class variables that were only used by one method were moved to that specific method instead.

UserService - User microservice

Metrics before refactoring:

- **Coupling Between Object Classes:** 15 (threshold 11)

Metrics after refactoring:

- **Coupling Between Object Classes:** 7

In order to reduce the coupling between object classes for the **UserService**, all the methods directly relating to authentication (i.e. hashing, verifying, token generation) were moved to a new **AuthService** class. This also ensures a clear separation between authentication and general CRUD logic.

UserController - User microservice

Metrics before refactoring:

- **Coupling Between Object Classes:** 12 (threshold 11)

Metrics after refactoring:

- **Coupling Between Object Classes:** 10

Almost all of the **UserController** methods had the very similar, very densely packed, logic to generate a response. By extracting this logic and moving it into a new helper class (consisting mostly of a few different response builder methods) the **UserController** was made less complex and its coupling has been reduced.

ContractController - Contract microservice

Metrics before refactoring:

- **Coupling Between Object Classes:** 12 (threshold 11)

Metrics after refactoring:

- **Coupling Between Object Classes:** 8

There was a lot of coupling with the custom exception classes, which are present and caught in every method of the controller. Also, the controller depended a lot on the **ContractService** class and calls it once per method. Catching a single exception and extracting the check of what instance it is (Extract method) to a separate method lowered the coupling with the custom exception classes. Surrounding the **ContractService** by an interface (Extract interface) and referencing the **ContractService** by its interface also decoupled the controller class from any specifics of the **ContractService**, and improved the score.

Method level refactorings

Offer microservice

convertToDatabaseColumn - Offer microservice

Class:

- StringListConverter

Metrics before refactoring:

- **Lines of Code:** 11
- **Cyclomatic Complexity:** 4

Metrics after refactoring:

- **Lines of Code:** 1
- **Cyclomatic Complexity:** 2

This method had a bit more complexity than needed, there was a for loop with a conditional statement inside of the for loop increasing the cyclomatic complexity to 4. With a call to the Java String library the method was reduced to a single line with only two possible paths.

acceptOffer - Offer microservice

Class:

- StudentOfferService

Metrics before refactoring:

- **Lines of Code:** 30
- **Cyclomatic Complexity:** 7

Metrics after refactoring:

- **Lines of Code:** 8
- **Cyclomatic Complexity:** 1

It was discovered that this method's CC was too high and there was also concerned about the length of the method. When investigating the issue, it was observed that the method had logic which could easily be extracted into another method.
This is Extract Method Refactoring was applied.
Two new methods were created. The first one - **validateOffer** contains the validation logic, which makes sure that the input is valid. The second one - **saveAcceptance** handles the application of status changes and stores all changes in the database. After applying the refactoring both metrics were drastically lowered, while the new methods were also had low LOC and CC scores.

createOffer - Offer microservice

Class:

- OfferController

Metrics before refactoring:

- **Coupling Between Object Classes:** 15
- **Lines of Code:** 27

Metrics after refactoring:

- **Coupling Between Object Classes:** 11
- **Lines of Code:** 19

It was discovered that this method had a very high CBO score. This method is used as a facade for creating all types of offers, so it would naturally involve more coupling. However, it was observed that one third of the coupling was caused by unnecessary usage of Offer classes as variables.
The solution was rather simple as instead of storing the transformed input in a variable we would simply use it directly in the method calls. This resulted in lowering of the method's CBO score, because of the removal of the variables, as well as lines of code reduction. Moreover, the class metric for CBO was also lowered and according to CodeMR the **OfferController** class has gone from medium-high score to low-medium. In addition, we decided to extract the functionality of creating a Company Offer into another method - **create-CompanyOffer**, so that the **createOffer** method only has to differentiate between student and company offers.

validateContractProposal - Contract microservice

Class:

- OfferController

Metrics before refactoring:

- **Weighted Method Count:** 9
- **Lines of Code:** 36

Metrics after refactoring:

- **Weighted Method Count:** 6
- **Lines of Code:** 16

There were a lot of decision points (if statements) in this method, which also caused more lines of code. Some part of the method logic were extracted in a new method: validateNumberOfWeeks, which reduced both the weighted method complexity and the number of lines of code.

getContract - Contract microservice

Class:

- ContractService

Metrics before refactoring:

- **Lines of Code:** 22
- **Cyclomatic Complexity:** 8

Metrics after refactoring:

- **Lines of Code:** 8

- **Cyclomatic Complexity:** 2

This method had a very high cyclomatic complexity as well as functionalities, which could be easily moved to a separate method. We used Extract Method Refactoring to deal with this issue. 2 new methods were created in order to handle the different behaviours of the initial method. **getActive** is used in order to get an active contract, while **getNewest** is used when we want to get the most recent contract.
After refactoring LOC and CC scores have been lowered. In addition, the added methods help a lot with maintaining clean and easily understandable code.

Class and method level refactoring

FeedbackService - Feedback microservice

Metrics before refactoring:

- **Coupling Between Object Classes:** 25 (very-high)

Metrics after refactoring:

- **Coupling Between Object Classes:** 12 (medium-low)

At first some methods were extracted from the existing methods so that these existing methods would be smaller. Then the decision was made to move these extracted methods into a new class because this class had a combination of business logic and validation and the coupling of the FeedBackService class was very high. A new helper class was created to handle the validation and cross service communication. This refactoring changed things both on a class level and method level as validation checks are now done elsewhere but the coupling of the original class has also been greatly reduced.

References

[1] CodeMR, “Java code metrics calculator (codemr).” `https://www.codemr.co.uk/`.

[2] M. Aniche, “Java code metrics calculator (ck).” `https://github.com/mauricioaniche/ck/`, 2015.