Assignment 2 Code Refactoring

In order to get started with this assignment, we had to identify what needed refactoring. To do so, we used the tool CodeMR to scan through our whole codebase to see what needed refactoring. These were the results we obtained:

1	StudentService					246	low-medium	high	high	low-medium
2	CompanyService					230	low-medium	high	medium-high	low-medium
3	AuthenticationCon...					69	low	high	low	low-medium
4	RequestService					173	low-medium	medium-high	high	low-medium
5	StudentController					136	low-medium	medium-high	medium-high	low-medium
6	CompanyController					136	low-medium	medium-high	medium-high	low-medium

As our criteria for refactoring, we decided to refactor the top 6 most problematic classes ranked by coupling and lack of cohesion. We first did class-based refactoring, and then we moved on to the methods

Student Service Refactoring

Class-Level Refactoring

To refactor the StudentService class, we decided to use the Class Extraction method to separate the business logic of the interaction of the Student microservice with every other microservice. This allowed for our coupling and lack of cohesion to decrease, as one class does not contain all the service logic. Since our business logic is split up, this allows our code to be more maintainable, understandable, robust and reliable.
Before this particular refactoring step, the StudentService class looked as follows:

Classes with high coupling (#1)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	CBO	CBO APP	CBO LIB	RFC
1	StudentService					246	22	15	7	41

After applying the class extraction method and creating the classes StudentServiceContract, StudentServiceFeedback, and StudentServiceRequest (one class for each of the microservice the Student microservice interacts with), our Coupling and Lack of Cohesion looks like this:

List of all classes (#27)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	StudentServiceReq...					105	low-medium	medium-high	medium-high	low-medium
2	StudentController...					80	low-medium	medium-high	medium-high	low-medium
3	StudentServiceCon...					57	low-medium	low-medium	low-medium	low-medium
4	StudentController...					45	low-medium	low-medium	low-medium	low
5	StudentServiceFee...					31	low-medium	low-medium	low	low
6	StudentController...					24	low-medium	low-medium	low	low
7	StudentService					30	low	low-medium	low-medium	low
8	StudentController					29	low	low-medium	low-medium	low

As you can see, none of the metrics cross the threshold of being problematic now and we reduced our coupling and lack of cohesion from high to medium-high.

After changing some of the return types of some of the endpoints, the status of the StudentService class went from this:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
template					
nl.tudelft.sem.student.service	low	low	low	low	
StudentService	low-medium	high	low-medium	high	22

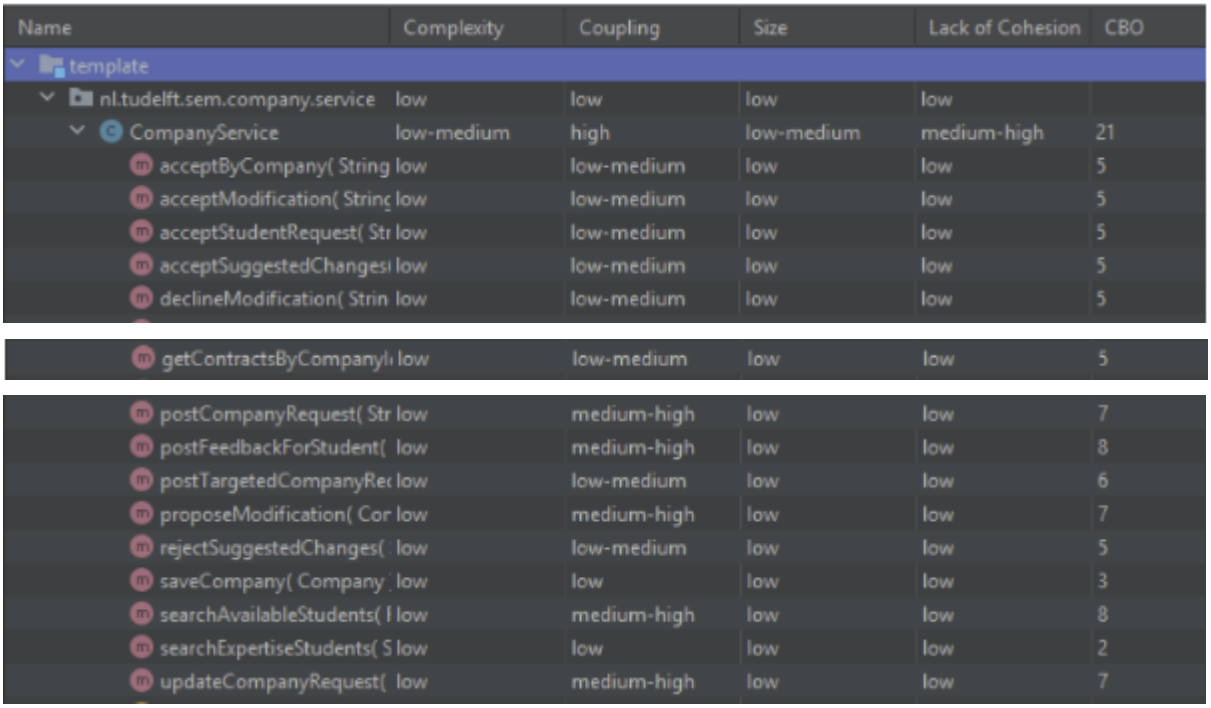
To this:

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
template					
nl.tudelft.sem.student.service	low	low	low	low	
StudentService	low-medium	medium-high	low-medium	high	20

Company Service Refactoring

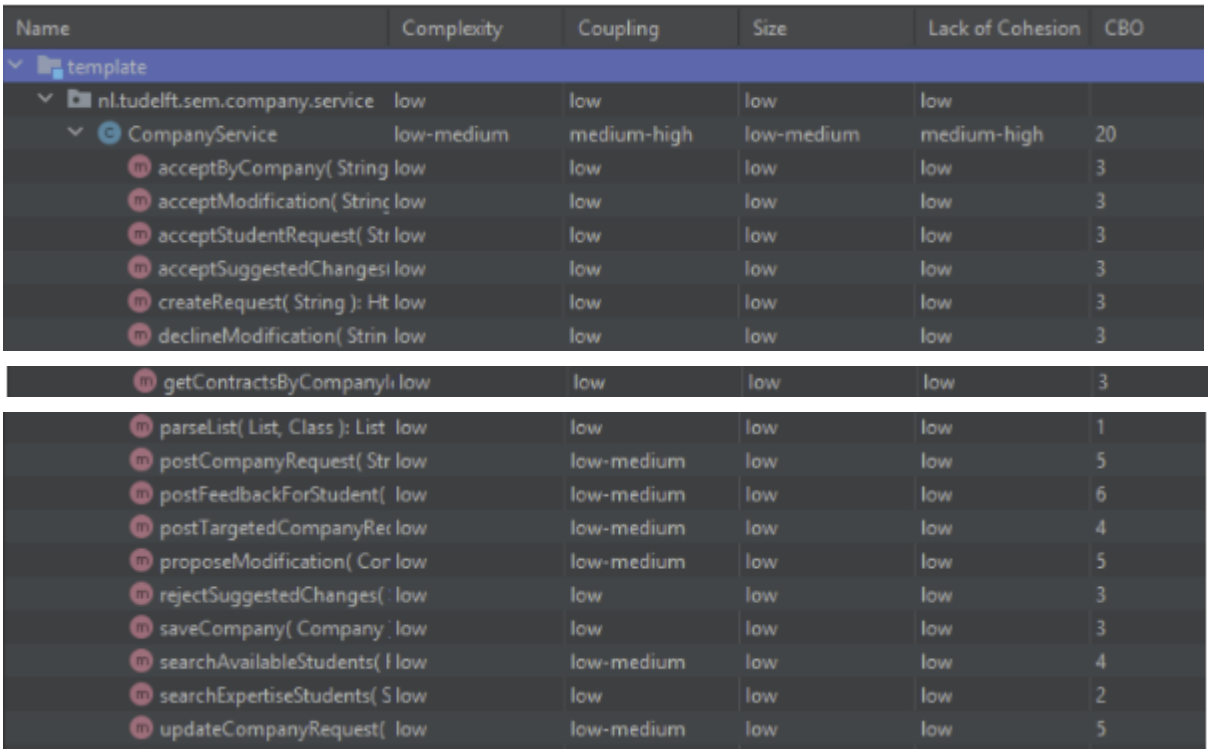
Class-Level Refactoring

Before these particular refactoring steps, the CompanyService class looked as follows:



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
template					
nl.tudelft.sem.company.service	low	low	low	low	
CompanyService	low-medium	high	low-medium	medium-high	21
acceptByCompany(String	low	low-medium	low	low	5
acceptModification(Strinç	low	low-medium	low	low	5
acceptStudentRequest(Str	low	low-medium	low	low	5
acceptSuggestedChangesi	low	low-medium	low	low	5
declineModification(Strin	low	low-medium	low	low	5
getContractsByCompanyI	low	low-medium	low	low	5
postCompanyRequest(Str	low	medium-high	low	low	7
postFeedbackForStudent(	low	medium-high	low	low	8
postTargetedCompanyRe	low	low-medium	low	low	6
proposeModification(Cor	low	medium-high	low	low	7
rejectSuggestedChanges(	low	low-medium	low	low	5
saveCompany(Company	low	low	low	low	3
searchAvailableStudents(I	low	medium-high	low	low	8
searchExpertiseStudents(S	low	low	low	low	2
updateCompanyRequest(	low	medium-high	low	low	7

Upon several method-level and class-level changes, the coupling was improved both at both the method-level and class-level:

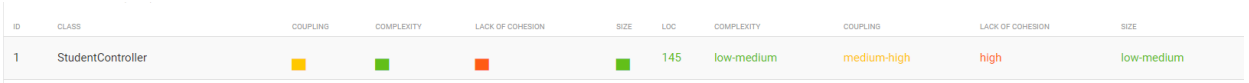


Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
template					
nl.tudelft.sem.company.service	low	low	low	low	
CompanyService	low-medium	medium-high	low-medium	medium-high	20
acceptByCompany(String	low	low	low	low	3
acceptModification(Strinç	low	low	low	low	3
acceptStudentRequest(Str	low	low	low	low	3
acceptSuggestedChangesi	low	low	low	low	3
createRequest(String): Ht	low	low	low	low	3
declineModification(Strin	low	low	low	low	3
getContractsByCompanyI	low	low	low	low	3
parseList(List, Class): List	low	low	low	low	1
postCompanyRequest(Str	low	low-medium	low	low	5
postFeedbackForStudent(	low	low-medium	low	low	6
postTargetedCompanyRe	low	low-medium	low	low	4
proposeModification(Cor	low	low-medium	low	low	5
rejectSuggestedChanges(	low	low	low	low	3
saveCompany(Company	low	low	low	low	3
searchAvailableStudents(I	low	low-medium	low	low	4
searchExpertiseStudents(S	low	low	low	low	2
updateCompanyRequest(	low	low-medium	low	low	5

Student Controller refactoring

Class-Level Refactoring

To refactor the student controller, we decided to use the same strategy as with the student service, namely the Extract Class method. The extraction we did was based on the idea to split the interaction between the multiple microservices. This means that there is a controller class for the interaction for the Feedback, Request and Contract microservices. This separation would cause a decrease in LOC, because the amount of different classes that the original class was relying on decreased immensely. Before refactoring, the StudentController class looked as follows:



ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	StudentController					145	low-medium	medium-high	high	low-medium

After class extraction, it looked like (note there are other classes too, but that is because of the ordering):

List of all classes (#27)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	StudentServiceReq...					104	low-medium	medium-high	medium-high	low-medium
2	StudentController...					79	low-medium	medium-high	medium-high	low-medium
3	StudentServiceCon...					57	low-medium	low-medium	low-medium	low-medium
4	StudentController...					44	low-medium	low-medium	low-medium	low
5	StudentServiceFee...					31	low-medium	low-medium	low	low
6	StudentController...					23	low-medium	low-medium	low	low
7	StudentController					32	low	low-medium	low-medium	low
8	StudentService					30	low	low-medium	low-medium	low
9	ContractModification					39	low	low	low	low

The main difference is the lack of cohesion, as expected, but it also improved the lines of code because of the extraction.

Request Service refactoring

Class-Level Refactoring

1. Before

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	DIT	NOC	WMC	LOC	NOM	LCOM	LCAM	LTCC
SEM Project													
nl.tudelft.sem.request.service	low	low	low	low				54	205				
ModificationService	low	low	low	low	4	1	0	7	32	4	0.167	0.312	0.0
RequestService	low-medium	medium-high	low-medium	high	14	1	0	47	173	30	0.828	0.808	0.469

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	RequestService					173	low-medium	medium-high	high	low-medium

2. After

Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	DIT	NOC	WMC	LOC	NOM	LCOM	LCAM	LTCC
SEM Project													
nl.tudelft.sem.request.service	low	low	low	low				58	219				
CompanyRequestService	low-medium	low-medium	low-medium	medium-high	6	2	0	25	90	18	0.529	0.715	0.0
ModificationService	low	low	low	low	4	1	0	7	32	4	0.167	0.312	0.0
RequestService	low	low-medium	low	low	7	1	2	3	16	3	1.0	0.444	0.667
StudentRequestService	low-medium	low-medium	low-medium	medium-high	7	2	0	23	81	13	0.667	0.721	0.0

1	CompanyRequestSer...					90	low-medium	low-medium	medium-high	low-medium
2	StudentRequestSer...					81	low-medium	low-medium	medium-high	low-medium
3	RequestService					16	low	low-medium	low	low

3. Description:

For this refactoring, the “Extract Class” technique was applied in such a way that the responsibilities of the RequestService were split between StudentRequest and CompanyRequest. The initial RequestService has become abstract and it is the parent of StudentRequestService and CompanyRequestService. Since the common method was sendGeneratedContract, this method remained in the abstract class, but the other methods were moved to their respective classes, taking into account what type of request is used in each specific method. In this way, both coupling and the lack of cohesion have been reduced.

Authentication Controller refactoring

Class-Level Refactoring

(1) Before:

QualifiedName	Complexity	Coupling	Size	Lack of Cohesion
AuthenticationController	low	high	low-medium	low

CBO	RFC	SRFC	DIT	NOC	WMC	LOC	CMLOC	NOF	NOSF	NOM	NOSM	NORM	LCOM	LCAM	LTCC	ATFD
21	45	27	1	0	10	69	55	6	0	5	0	0	0.917	0.5	1.0	1

(2) After:

QualifiedName	Complexity	Coupling	Size	Lack of Cohesion
AuthenticationController	low	medium-high	low	low

CBO	RFC	SRFC	DIT	NOC	WMC	LOC	CMLOC	NOF	NOSF	NOM	NOSM	NORM	LCOM	LCAM	LTCC	ATFD
15	46	18	1	0	6	44	33	4	0	4	0	0	0.875	0.5	1.0	1

(3) Description:

For this refactoring I applied the “Move Method” technique basically moving the logic from the AuthenticationController class to the AuthenticationService class. This way I replaced 3 of the fields in the controller with just 1 (the authentication service) thus reducing the coupling and the number of fields of the class. Furthermore, by moving the method I also reduced the lines of code of the class and some of the other metrics.

Authentication Service refactoring

Method-Level Refactoring

(1) Before:

QualifiedName	Complexity	Coupling	Size	Lack of Cohesion	MCC	NBD	LOC	#Pa	#MC	#AF
AuthenticationService.checkRole(String): boolean	low	low	low	low	1	1	3	1	3	0
AuthenticationService.loadUserByUsername(String): UserDetails	low	low-medium	low	low	1	1	6	1	3	1
AuthenticationService.register(User): boolean	low	medium-high	low	low	5	2	20	1	12	6

(2) After:

QualifiedName	Complexity	Coupling	Size	Lack of Cohesion	MCC	NBD	LOC	#Pa	#MC	#AF
AuthenticationService.checkIfValidForRegistration(User): boolean	low	low	low	low	1	1	4	1	5	1
AuthenticationService.generateRequest(String, String): HttpEntity	low	low-medium	low	low	1	1	6	2	2	1
AuthenticationService.loadUserByUsername(String): UserDetails	low	low-medium	low	low	1	1	6	1	3	1
AuthenticationService.register(User): boolean	low	low	low	low	4	2	9	1	6	1
AuthenticationService.saveNewCompany(String, String): void	low	low	low	low	1	1	3	2	2	1
AuthenticationService.saveNewStudent(String, String): void	low	low	low	low	1	1	4	2	2	1

(3) Description:

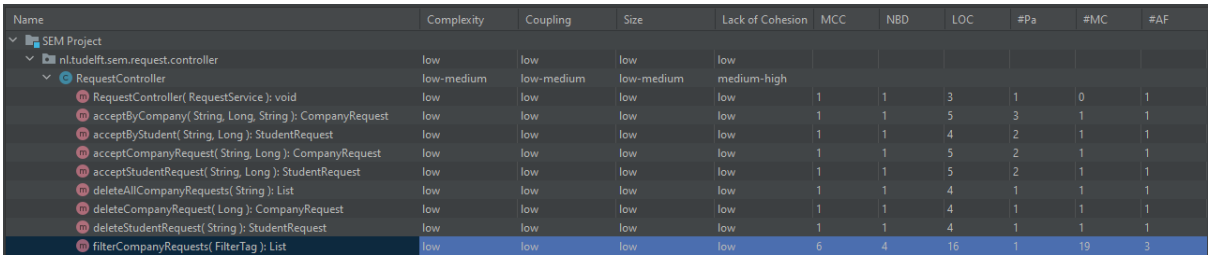
For this refactoring I applied the “Split Method” technique by splitting the logic of the method into multiple smaller methods. This way instead of the register being responsible for 4 different things and thus being highly coupled I added a method for checking whether the user is valid for registration, a method for generating HttpEntities and 2 methods for sending requests to the Student or Company microservices. This way I reduced the coupling, lines of code and the methods called of the register method.

Request Controller refactoring

Method-Level Refactoring

FilterCompanyRequests method Refactoring:

1. Before



Name	Complexity	Coupling	Size	Lack of Cohesion	MCC	NBD	LOC	#Pa	#MC	#AF
SEM Project										
nl.tudelft.sem.request.controller	low	low	low	low						
RequestController	low-medium	low-medium	low-medium	medium-high						
RequestController(RequestService): void	low	low	low	low	1	1	3	1	0	1
acceptByCompany(String, Long, String): CompanyRequest	low	low	low	low	1	1	5	3	1	1
acceptByStudent(String, Long): StudentRequest	low	low	low	low	1	1	4	2	1	1
acceptCompanyRequest(String, Long): CompanyRequest	low	low	low	low	1	1	5	2	1	1
acceptStudentRequest(String, Long): StudentRequest	low	low	low	low	1	1	5	2	1	1
deleteAllCompanyRequests(String): List	low	low	low	low	1	1	4	1	1	1
deleteCompanyRequest(Long): CompanyRequest	low	low	low	low	1	1	4	1	1	1
deleteStudentRequest(String): StudentRequest	low	low	low	low	1	1	4	1	1	1
filterCompanyRequests(FilterTag): List	low	low	low	low	6	4	16	1	19	3

2. After

filterCompanyRequests(FilterTag): List	low	low	low	low	3	2	8	1	5	0
filterCompanyRequestsByHoursPerWeek(FilterTag): List	low	low	low	low	2	2	6	1	6	1
filterCompanyRequestsBySalary(FilterTag): List	low	low	low	low	2	2	6	1	6	1
filterCompanyRequestsByTotalHours(FilterTag): List	low	low	low	low	2	2	5	1	6	1

3. Description

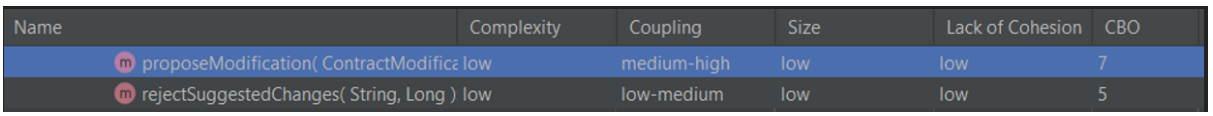
For this refactoring, the “Split Method” technique was applied by splitting the logic of the method into multiple smaller methods. This way instead of the filter method being responsible for 4 different things hence having a big cyclomatic complexity, 3 methods were added for checking what type of filtering is requested. Now the initial filterCompanyRequests is only responsible to check if the values put in the filter tag are correct. In consequence, the cyclomatic complexity was reduced and the implementation of the feature is more understandable.

Company Service refactoring

Method-Level Refactoring

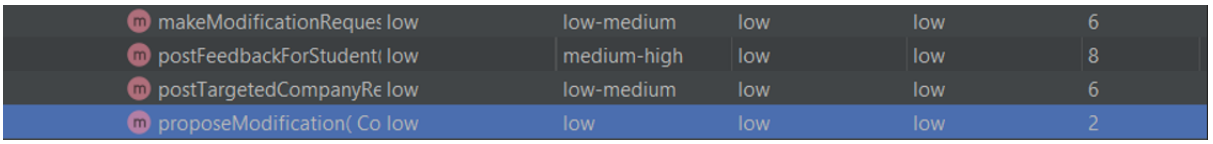
proposeModification method Refactoring:

1. Before



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
proposeModification(ContractModifica	low	medium-high	low	low	7
rejectSuggestedChanges(String, Long)	low	low-medium	low	low	5

2. After



makeModificationReques	low	low-medium	low	low	6
postFeedbackForStudent(	low	medium-high	low	low	8
postTargetedCompanyRe	low	low-medium	low	low	6
proposeModification(Co	low	low	low	low	2

3. Description

For this refactoring, the “Split Method” technique was applied by splitting the logic of the method into smaller methods. The proposeModification method also made the Modification request, this resulted in high coupling. Now the proposeModification

method is only responsible for setting values and makeModificationRequest makes the request.

Method-Level Refactoring

In order to improve the coupling within the methods themselves, we realized that we were duplicating code to generate a request and to convert a response to a list. We did not realize this while we were developing. Hence, we created these methods and replaced each of the duplicated codes as a method, and this improved the CBO of a variety of methods.

```
/**
 * Converts response to a List of type T.
 *
 * @param response response
 * @param className className
 * @param <T> T
 * @return returnList
 */
public <T> List<T> responseToList(List<LinkedHashMap> response, Class<T> className) {
    List<T> returnList = new ArrayList<>();
    for (LinkedHashMap o : response) {
        T obj = mapper.convertValue(o, className);
        returnList.add(obj);
    }
    return returnList;
}
```

```
/**
 * Creates a request with a JSON body.
 *
 * @param json The JSON body
 * @return The HTTP request to be sent.
 */
public HttpEntity<String> requestCreator(String json) {
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    return new HttpEntity<>(json, headers);
}
```