SEM Assignment 2: Refactoring

SEM Group 1A

January 2022

Contents

1 Introduction

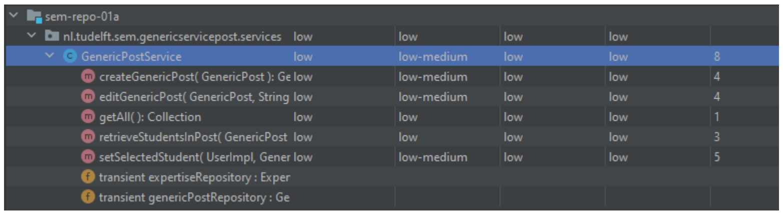
Assignment 2: Code refactoring Thanks to static analysis we can easily find code smells based on some well studied and documented metrics. We decided to run the CodeMR tool to easily and graphically see important code metrics on our codebase and then choose classes and methods that reported problematic values. In particular we looked at:

- Class cohesion: if it was reported to be higher than “low” we planned to reduce it by at least 0.1.
- Complexity: if it was reported to be higher than “low” we decided to reduce some or all the relevant metrics like methods called and accessed field until they were not reported to be problematic anymore by the tool.
- Coupling: to reduce coupling we look at similar metrics as complexity, such as methods called and so we planned a similar strategy as written above.

Based on this conditions we decided on the following classes and methods:

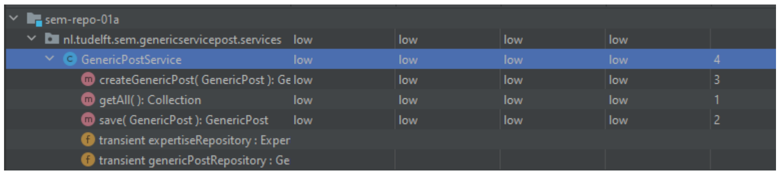
2 Class-Level Smells

2.1 Generic Service Post Services - Low-Medium Coupling



sem-repo-01a					
nl.tudelft.sem.genericservicepost.services	low	low	low	low	
GenericPostService	low	low-medium	low	low	8
createGenericPost(GenericPost): Ge	low	low-medium	low	low	4
editGenericPost(GenericPost, String	low	low-medium	low	low	4
getAll( ): Collection	low	low	low	low	1
retrieveStudentsInPost(GenericPost	low	low	low	low	3
setSelectedStudent(UserImpl, Gener	low	low-medium	low	low	5
transient expertiseRepository : Exper					
transient genericPostRepository : Ge					

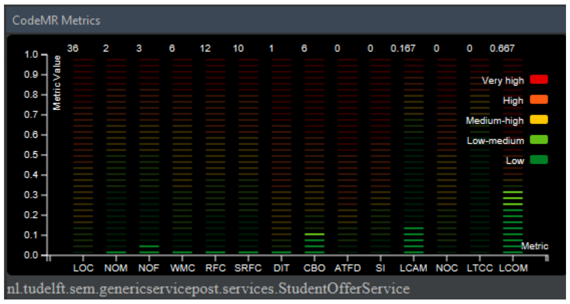
(a) GSP overview pre refactoring



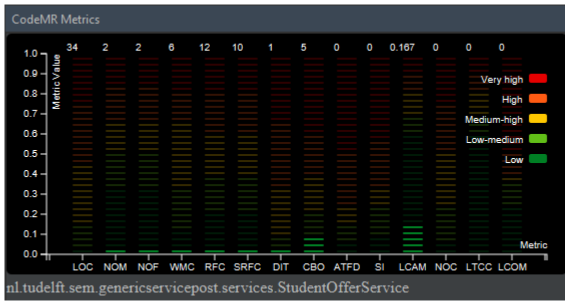
sem-repo-01a					
nl.tudelft.sem.genericservicepost.services	low	low	low	low	
GenericPostService	low	low	low	low	4
createGenericPost(GenericPost): Ge	low	low	low	low	3
getAll( ): Collection	low	low	low	low	1
save(GenericPost): GenericPost	low	low	low	low	2
transient expertiseRepository : Exper					
transient genericPostRepository : Ge					

(b) GSP overview post refactoring

Figure 1: The Analysis of code smells of GenericPostService.



(a) GSP overview pre refactoring



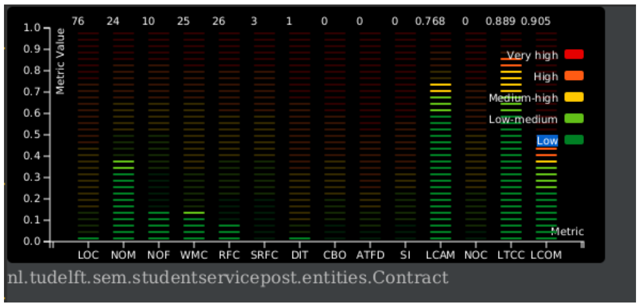
(b) GSP overview post refactoring

Figure 2: The Analysis of code smells of StudentOfferService.

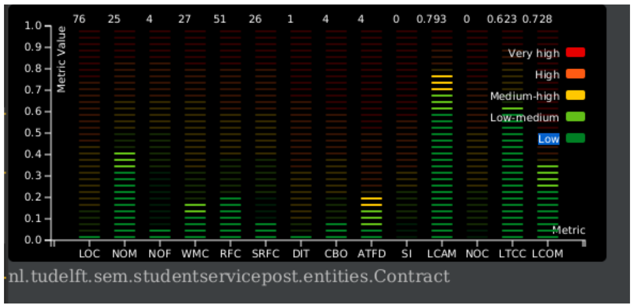
1. Split the GenericPostService Class into 3 separate classes, based on the methods functionalities and use of other Classes and Repositories. The Class was chosen, because of its Low-Medium Coupling and Low-Medium CBO. After Refactoring and using the Code Extraction Technique the Coupling and CBO Metric were reduced.

2. Removed unused repository in StudentOfferService to lower Coupling. The Class was chosen, because of its Low-Medium Coupling, CBO and LCOM Metrics. After Refactoring all Metrics were lowered to Low. Before Refactoring both Service Classes had a Low-Medium Class Coupling. By removing an unused repository for the StudentOfferService class and splitting the GenericPostService class, the class coupling was lowered to Low.

2.2 Contract in Student service post micro service - Lack of Cohesion



(a) Contract metrics pre refactoring

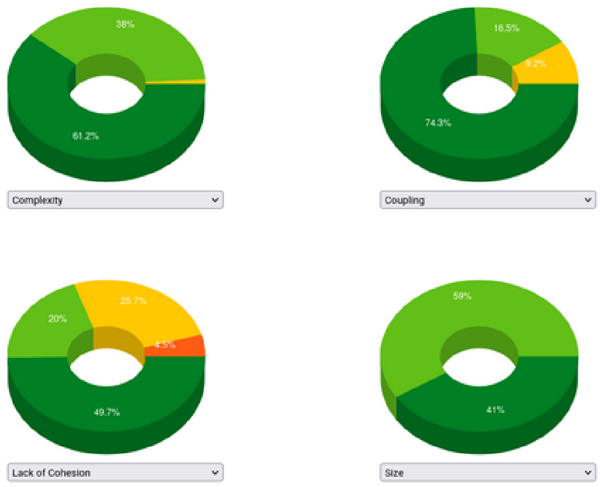


(b) Contract metrics post refactoring

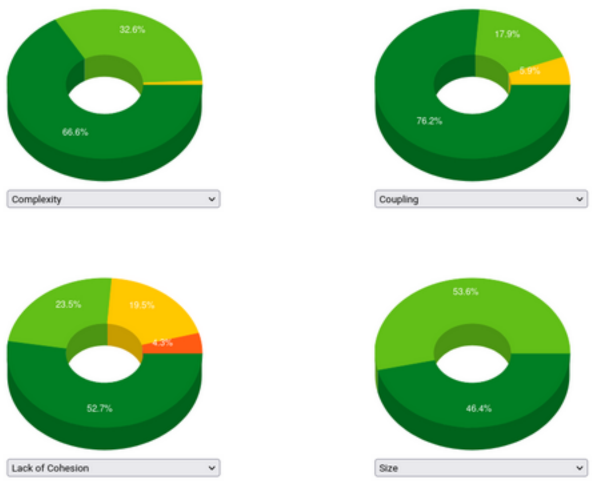
Figure 3: The Analysis of code smells in contract class in SSP microservice.

The class reported a high lack of cohesion, especially lack of tight class cohesion. The code smell of having a not cohesive class can be solved by using the "extract class" refactoring technique, this is based on extracting part of the class that do not share lots of functionalities and logical correlations between each other. For the contract class this was done by extracting field related to timing of the contracts in a new "ContractTimes" class that stores and handles logic for starting, ending and weekly times of the contract. Another extracted class based on information of the 2 parties involved in the contract name "ContractParties" was created in the same manner. These 2 extracted classes contain the fields that were extracted from the original Contract class and when a call to a getter or setter is done on the old class we delegate logic on those fields to the new class.

2.3 CompanyOfferService - Coupling



(a) CompanyOfferService pre refactoring



(b) CompanyOfferService post refactoring

Figure 4: The Analysis of code smells in CompanyOfferService.

This service class (CompanyOfferService.java) was found to contain multiple code smells that resulted in a higher than average coupling score. Most notably, this class injected the repository linked to another DAO as a dependency (PostRepository.java). This resulted in this service implementing code to access the repository that had already been implemented in the service linked to the

DAO (PostService.java). Not only that, but there were multiple methods that contained duplicate code snippets linked to accessing that repository, which increased the size and complexity metrics needlessly, which in the future may have caused problems if a change had to be introduced with how the repository was accessed.

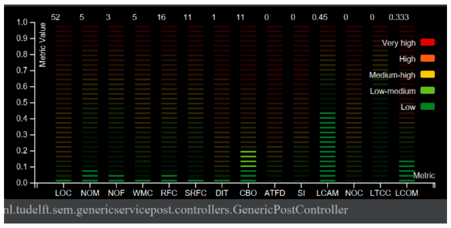
As a result, the following changes were made:

- The repository dependency linked to the DAO was removed and was replaced by a dependency injection of the service linked to that DAO (PostService.java).
- The duplicate code snippets were removed, instead replaced by a method call to a method within PostService.java.

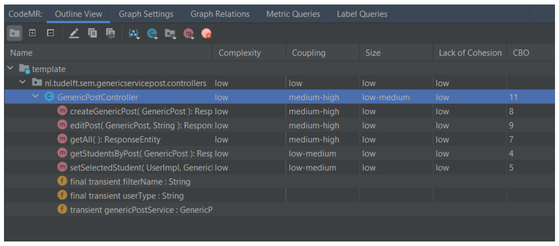
These changes resulted in a multitude of tests failing due to the different way of accessing the data from the repository, requiring PostService.java to be mocked in CompanyOfferServiceTest.java and test cases re-written.

Finally, as can be seen in the graphs, this refactoring operation resulted in 2.1% increase in low-medium coupling metrics as well as a small decrease in complexity and size. Notably, there was a 0.7% decrease in low lack of cohesion metrics, which unfortunately can become a side effect of optimising for coupling. This can probably be attributed to the additional method that had to be added to PostService.java in order to implement functionality needed by the refactored class, specifically the savePost() method. It only makes use of the postRepository field of the service, thus it lowers the overall LCOM score by a small margin.

2.4 GenericPostController - Coupling



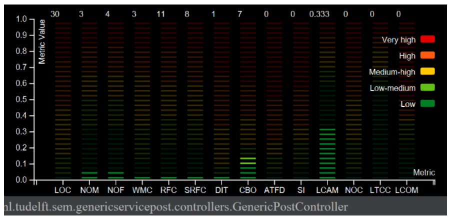
(a) GenericPostController metrics pre refactoring


CodeMR: Outline View Graph Settings Graph Relations Metric Queries Label Queries

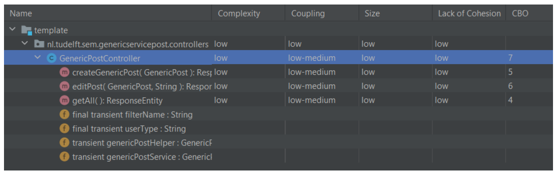
Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
template					
nl.tudelft.sem.genericservicepost.controllers	low	low	low	low	
GenericPostController	low	medium-high	low-medium	low	11
createGenericPost(GenericPost): Resp	low	medium-high	low	low	8
editPost(GenericPost, String): Respon	low	medium-high	low	low	9
getAll( ): ResponseEntity	low	medium-high	low	low	7
getStudentsByPost(GenericPost): Resp	low	low-medium	low	low	4
setSelectedStudent(UserImpl, GenericI	low	low-medium	low	low	5
final transient filterName : String					
final transient userType : String					
transient genericPostService : GenericP					

(b) GenericPostController overview pre refactoring

Figure 5: *Before* refactoring.The Analysis of GenericPostController.



(a) GenericPostController metrics post refactoring



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
template					
nl.tudelft.sem.genericservicepost.controllers	low	low	low	low	
GenericPostController	low	low-medium	low	low	7
createGenericPost(GenericPost): Res	low	low-medium	low	low	5
editPost(GenericPost, String): Respor	low	low-medium	low	low	6
getAll( ): ResponseEntity	low	low-medium	low	low	4
final transient filterName : String					
final transient userType : String					
transient genericPostHelper : GenericF					
transient genericPostService : GenericI					

(b) GenericPostController overview post refactoring

Figure 6: *After* refactoring.The Analysis of GenericPostController.

GenericPostController did not have strong code smells to begin with. However, it originally had a high coupling for its size, and most of the imported methods and classes had smaller tasks. It was therefore clear it was still possible to improve the code smell for this class. Additionally, although it had a low lack of cohesion overall, it had overall a pretty high LCAM (Lack of Cohesion Among Methods) and LCOM (Lack of Cohesion of Methods), and it was therefore close going from a "low" status to a "medium-low" status. To be able to fix both issues, I decided the best way to proceed was to do an "extract class refactoring" combined with "move method refactoring".

To perform an extract class- and move method refactoring I did the following:

- Created new helper methods filter, mappingJacksonPost and mappingJacksonCollection with the purpose of taking some coupling out of the original methods, and moved the newly created methods to a new helper class.
- Moved the original methods getStudentsByPost and setSelectedStudent from the class GenericPostController to another class StudentOfferController, as those methods fit with the StudentOffer more than the GenericPost.

As we can see from the graphs, the number of overall CBO (Coupling Between Object Classes) went down from 11 to 7, and also each method in GenericPostController got significantly less coupling than the original code. Although the main objective with this refactoring was to reduce coupling, the LCAM also went down from 0.45 to 0.33 and LCOM went down from 0.33 to 0. Another unintentional but welcoming side effect of the code extraction, is that the size of the class got reduced.

2.5 User in User micro-service - Medium-High Lack of Cohesion

In the users microservice the entity User had a medium-high lack of cohesion. The lack of cohesion among methods (LCOM) was 0.708, the lack of tight class cohesion (LTCC) was 0.462, the lack of cohesion methods (LCOM) was 0.536. The problem was that this class contained two methods -addFeedback and removeFeedback-that were performing actions that were not necessary.

```
public void addFeedback(Feedback feedback) {
        List<Feedback> newList = new ArrayList<>(this.feedbacks);
        newList.add(feedback);
        this.feedbacks = newList;
}

public void removeFeedback(Feedback feedback) {
        this.feedbacks.remove(feedback);
}
```

Consider the following example:

```
Student student = new UserFactory().createUser(netID, name, 0.0f, "student");
Feedback feedback = new Feedback("text", 1, student);
```

For adding a feedback, one could perform the following action:

```
student.addFeedback(feedback);
```

However, this is not needed:

```
student.getFeedbacks().add(feedback);
```

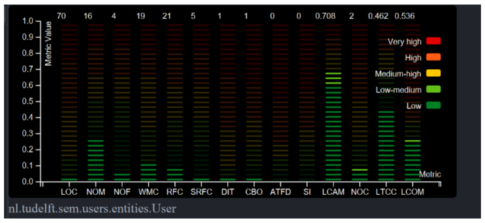
For removing a feedback:

```
student.removeFeedback(feedback);
```

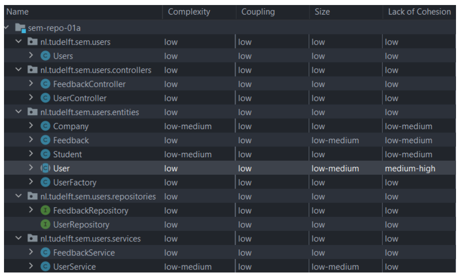
Again, this is not needed:

```
student.getFeedbacks().remove(feedback);
```

The methods addFeedback and removeFeedback were redundant. Therefore, after refactoring, they were removed and replaced with built-in methods. By doing this all metrics related to cohesion were improved: the lack of cohesion among methods (LCOM) dropped to 0.657, the lack of tight class cohesion (LTCC) was reduced to 0.436, the lack of cohesion methods (LCOM) dropped to 0.5. This resulted in a low-medium lack of cohesion, which was previously at a level of medium-high.



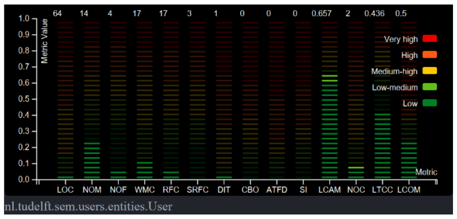
(a) User metrics pre refactoring



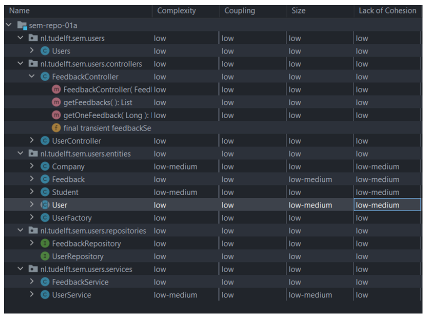
Name	Complexity	Coupling	Size	Lack of Cohesion
sem-repo-01a				
nl.tudelft.sem.users	low	low	low	low
Users	low	low	low	low
nl.tudelft.sem.users.controllers	low	low	low	low
FeedbackController	low	low	low	low
UserController	low	low	low	low
nl.tudelft.sem.users.entities	low	low	low	low
Company	low-medium	low	low	low-medium
Feedback	low	low	low-medium	low-medium
Student	low-medium	low	low	low-medium
User	low	low	low-medium	medium-high
UserFactory	low	low	low	low
nl.tudelft.sem.users.repositories	low	low	low	low
FeedbackRepository	low	low	low	low
UserRepository	low	low	low	low
nl.tudelft.sem.users.services	low	low	low	low
FeedbackService	low	low	low	low
UserService	low-medium	low	low-medium	low

(b) User overview pre refactoring

Figure 7: *Before* refactoring.The Analysis of User in User micro-service.



(a) User metrics post refactoring



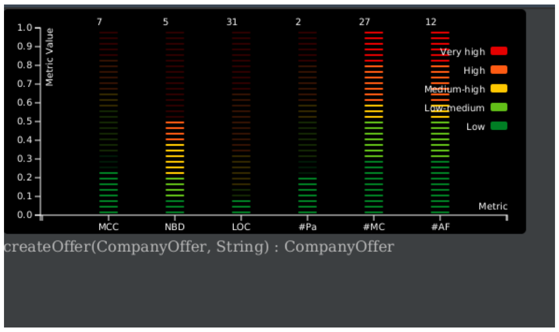
Name	Complexity	Coupling	Size	Lack of Cohesion
sem-repo-01a				
nl.tudelft.sem.users	low	low	low	low
Users	low	low	low	low
nl.tudelft.sem.users.controllers	low	low	low	low
FeedbackController	low	low	low	low
FeedbackController(Feedl	low	low	low	low
getFeedbacks( ): List	low	low	low	low
getOneFeedback(Long): I	low	low	low	low
final transient feedbackSe				
UserController	low	low	low	low
nl.tudelft.sem.users.entities	low	low	low	low
Company	low-medium	low	low	low-medium
Feedback	low	low	low-medium	low-medium
Student	low-medium	low	low	low-medium
User	low	low	low-medium	low-medium
UserFactory	low	low	low	low
nl.tudelft.sem.users.repositories	low	low	low	low
FeedbackRepository	low	low	low	low
UserRepository	low	low	low	low
nl.tudelft.sem.users.services	low	low	low	low
FeedbackService	low	low	low	low
UserService	low-medium	low	low-medium	low

(b) User overview post refactoring

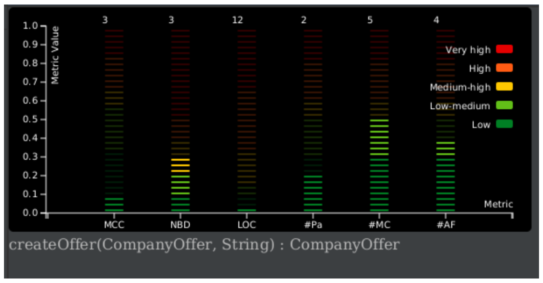
Figure 8: *After* refactoring.The Analysis of User in User micro-service.

3 Method-level smells

3.1 CreateOffer in CompanyOfferService - Coupling



(a) createOffer metrics pre refactoring

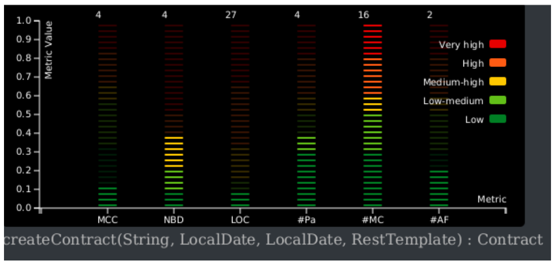


(b) createOffer metrics post refactoring

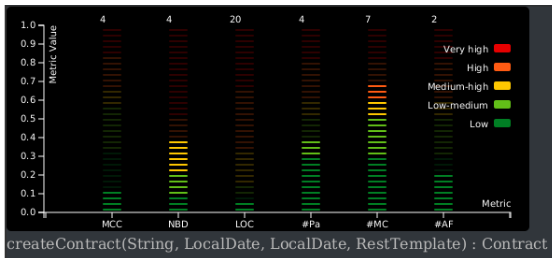
Figure 9: The Analysis of code smells in createOffer method.

The "CreateOffer" method in the "CompanyOfferService" reported very high numbers of external methods called and field accessed, in particular 27 for the former and 12 for the latter. This numbers are considered to be very high and problematic as it could indicate very high coupling between classes and methods, too long and complex methods. This kind of code smells is solved by extracting method functionalities in other methods/classes. For this method in particular lots of functionality that was relevant to the CompanyOfferService itself was moved to a new service that manages logic relevant to other entities, like Competencies and Requirements. This not only reduced the length of the method and the number of called methods, but also the coupling of the method and of the entire class, cyclomatic complexity and nested block depth of the method. In particular logic that acted on repositories different from the one for CompanyOffer objects was moved to a different service that manages exactly this kind of relations between objects. Methods that update already existing requirements and competencies were created and methods that check of validity of posts/users were implemented.

3.2 CreateContract in CompanyOfferService - Coupling



(a) createContract metrics pre refactoring



(b) createContract metrics post refactoring

Figure 10: The Analysis of code smells in createOffer method in CompanyOfferService.

This method was chosen to be refactored due to its absurdly high number of method calls, as per the codeMR metric classification. The main culprit behind the excessive number of method calls was an initialisation of the Contract.java DTO within the method, as most of its parameters (of which there are many) required a call to the getters of other objects.

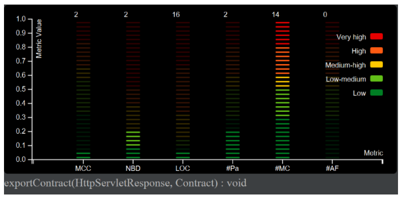
In order to fix this critical code smell (#MC), the following was changed:

- The initialisation of the Contract DTO was moved into the Contract.java Class as a static method, namely buildFromOffer().

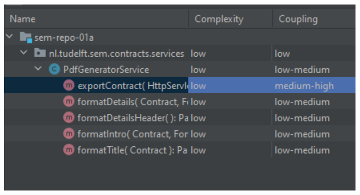
This resulted in 50% decrease in the number of method calls within the method, as these were delegated to the static method, which intuitively makes most sense to belong in that class.

3.3 ExportContract in ContractService - Coupling

In the contract microservice the code smells detected by codeMR were minimal. After careful inspection however, a coupling smell was detected in the project outline tab produced by codeMR as can be seen in figure 11a. Specifically the exportContract method in the PdfGeneratorService had medium-high coupling. According to the codeMR metrics of this method, the root cause of this high coupling was a high number of method calls(marked #MC in figure 11b and indicating a very high number of 14).



(a) exportContract metrics pre refactoring



Name	Complexity	Coupling
sem-repo-01a		
nl.tudelft.sem.contracts.services	low	low
PdfGeneratorService	low	low-medium
exportContract(HttpServl	low	medium-high
formatDetails(Contract, F	low	low-medium
formatDetailsHeader( ): Pa	low	low-medium
formatIntro(Contract, For	low	low-medium
formatTitle(Contract): Pa	low	low-medium

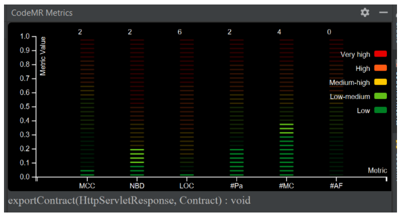
(b) PdfGeneratorService overview pre refactoring

Figure 11: *Before* refactoring.The Analysis of method- level code smells in contract micro-service.

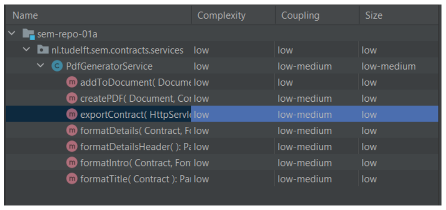
To explain how this problem was solved, first the purpose of this method will be explained. It was responsible for making a response document to be sent in the HttpServlettResponse, formatting paragraphs to include the contract details in the desired structure by using multiple methods I created within the same class and then finally adding these paragraphs to the document using methods of an external package.

With the objectives of the method in mind it was clear that the method calls could be further partitioned into new methods in a way that didn't negatively influence cohesion. A method called createPdf was made, to take care of all the formatting functions within the class and a method called addToDocument which contained all the calls to the external package used to add the different paragraphs to the response document.

By taking the approach described in the previous paragraph the method calls were reduced to a low number of 4 as seen in figure 12a. This in turn improved the coupling of the exportContract method to be low-medium as seen in figure 12b. Additionally figure 12b shows the new methods also have acceptable couple, being low-medium.



(a) exportContract metrics post refactoring

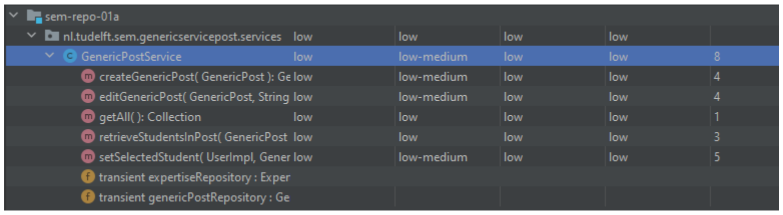


Name	Complexity	Coupling	Size
sem-repo-01a			
nl.tudelft.sem.contracts.services	low	low	low
PdfGeneratorService	low	low-medium	low-medium
addToDocument(Docume	low	low	low
createPDF(Document, Co	low	low-medium	low
exportContract(HttpServle	low	low-medium	low
formatDetails(Contract, F	low	low-medium	low
formatDetailsHeader( ): Pa	low	low-medium	low
formatIntro(Contract, Fon	low	low-medium	low
formatTitle(Contract): Pa	low	low-medium	low

(b) PdfGeneratorService overview post refactoring

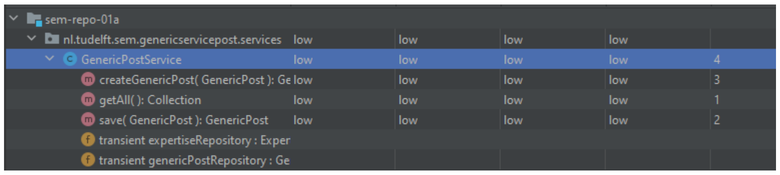
Figure 12: *After* refactoring. The Analysis of method-level code smells in contract micro-service.

3.4 2 methods in GenericServicePostService - Low-Medium Coupling



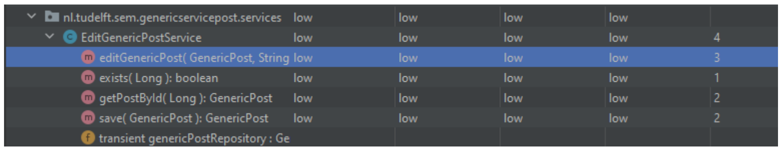
sem-repo-01a					
nl.tudelft.sem.genericservicepost.services	low	low	low	low	
GenericPostService	low	low-medium	low	low	8
createGenericPost(GenericPost): Ge	low	low-medium	low	low	4
editGenericPost(GenericPost, String	low	low-medium	low	low	4
getAll( ): Collection	low	low	low	low	1
retrieveStudentsInPost(GenericPost	low	low	low	low	3
setSelectedStudent(UserImpl, Gener	low	low-medium	low	low	5
transient expertiseRepository : Exper					
transient genericPostRepository : Ge					

(a) GSP overview pre refactoring



sem-repo-01a					
nl.tudelft.sem.genericservicepost.services	low	low	low	low	
GenericPostService	low	low	low	low	4
createGenericPost(GenericPost): Ge	low	low	low	low	3
getAll( ): Collection	low	low	low	low	1
save(GenericPost): GenericPost	low	low	low	low	2
transient expertiseRepository : Exper					
transient genericPostRepository : Ge					

(b) GSP createGenericPost() overview post refactoring



nl.tudelft.sem.genericservicepost.services	low	low	low	low	
EditGenericPostService	low	low	low	low	4
editGenericPost(GenericPost, String	low	low	low	low	3
exists(Long): boolean	low	low	low	low	1
getPostById(Long): GenericPost	low	low	low	low	2
save(GenericPost): GenericPost	low	low	low	low	2
transient genericPostRepository : Ge					

(c) GSP editGenericPost() overview post refactoring

Figure 13: The Analysis of code smells of GenericPostService at method level.

The two methods creteGenericPost and editGenericPost initially had a low-medium Coupling, by creating helper methods for them the CBO of those methods was lowered and therefore the Coupling as well.

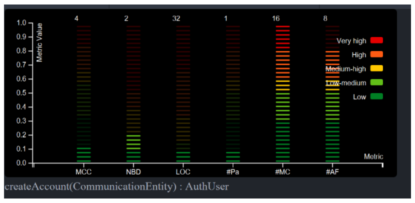
3.5 CreateAccount in Gateway microservice - Coupling

In the gateway microservice, the method createAccount in the GatewayService was long and coupled. We see that the afferent coupling (AF) was at 8 (high) before refactoring. The lines of code (LOC) was at 32 (low) and the McCabe complexity was at 16 (very high). The problem was that there was no separation between communicating with the authentication and users microservice and checking whether we could create a new account.

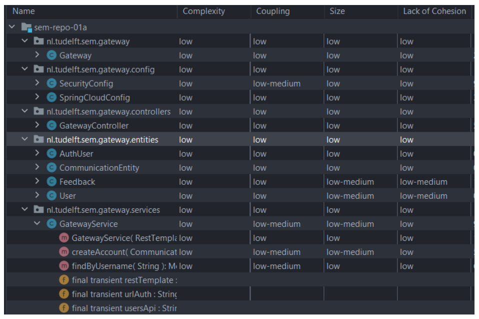
```
public AuthUser createAccount(CommunicationEntity communicationEntity)
                                    throws IllegalArgumentException {

    if (communicationEntity.getRole() == null) {
        throw new IllegalArgumentException("Role must not be null!");
    }

        User user = restTemplate.getForObject(
                        usersApi + communicationEntity.getNetID(), User.class
        );

        if (user != null) {
                throw new IllegalArgumentException("Net_ID_already_exists!");
        }

        AuthUser authenticatedUser = restTemplate.getForObject(
                        urlAuth + "/ADMIN?netID="
                        + communicationEntity.getNetID(),
                            AuthUser.class
        );

        if (authenticatedUser != null) {
                throw new IllegalArgumentException("NetID_does_not_exist!");
        }

        AuthUser authUser = new AuthUser(
                communicationEntity.getNetID(),
                        communicationEntity.getPassword(),
                        communicationEntity.getRole());

        restTemplate.execute(
                        urlAuth + "addAuthUser/"
                                        + communicationEntity.getNetID() + "/"
                                        + communicationEntity.getPassword()
                                        + "/"
                                        + communicationEntity.getRole(),
                        HttpMethod.POST, null, null
        );

        restTemplate.execute(
                        usersApi + "addUser/"
                                        + communicationEntity.getNetID() + "/"
                                        + communicationEntity.getName() + "/"
                                        + communicationEntity.getRole(),
                        HttpMethod.POST, null, null
        );

        return authUser;
}
```

After refactoring, the createAccount method only checks whether the netID exists in the authentication or users microservice. All metrics were improved. We see that the afferent coupling dropped to 0, the lines of code dropped to 15, the McCabe complexity was reduced to 9, which is still very high, but it is two times smaller than the value we had before the refactoring. We have created a method that connects to the users microservice and asks for a user with the corresponding netID. Likewise, a method that connects to the authentication microservice. Furthermore, we have a method that tells the authentication and users microservice to create a new account using the available information. Each of the newly added methods has low coupling and a small size.

```
public AuthUser createAccount(CommunicationEntity communicationEntity)
            throws IllegalArgumentException {

        if (communicationEntity.getRole() == null) {
                throw new IllegalArgumentException("Role must not be null!");
        }

        User user = getUser(communicationEntity.getNetID());

        if (user != null) {
                throw new IllegalArgumentException("Net ID already exists!");
        }

        AuthUser authenticatedUser = getAuthenticatedUser(
        communicationEntity.getNetID());

        if (authenticatedUser != null) {
                throw new IllegalArgumentException("NetID does not exist!");
        }

        AuthUser authUser = new AuthUser(communicationEntity.getNetID(),
                        communicationEntity.getPassword(),
                        communicationEntity.getRole());

        executePOSTS(communicationEntity);

        return authUser;
}

private User getUser(String netID) {
        try {
                return restTemplate.getForObject(
                                usersApi + netID, User.class
                );
        } catch (Exception e) {
                e.printStackTrace();
            }
            return null;
        }

        private AuthUser getAuthenticatedUser(String netID) {
            try {
                return restTemplate.getForObject(
                                urlAuth + "/ADMIN?netID="
                                + netID, AuthUser.class
                );
            } catch (Exception e) {
                e.printStackTrace();
            }
            return null;
        }

        private void executePOSTS(CommunicationEntity communicationEntity) {
            restTemplate.execute(
                        urlAuth + "addAuthUser/"
                                + communicationEntity.getNetID() + "/"
                                + communicationEntity.getPassword()
                                + "/"
                                + communicationEntity.getRole(),
                        HttpMethod.POST, null, null
            );

            restTemplate.execute(
                        usersApi + "addUser/"
                                + communicationEntity.getNetID() + "/"
                                + communicationEntity.getName() + "/"
                                + communicationEntity.getRole(),
                        HttpMethod.POST, null, null
            );
        }
```



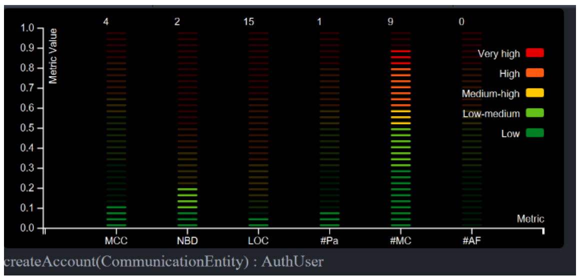
(a) createAccount metrics pre refactoring



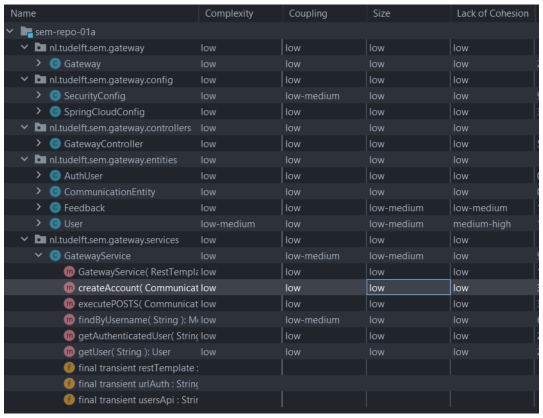
Name	Complexity	Coupling	Size	Lack of Cohesion
sem-repo-01a				
nl.tudelft.sem.gateway	low	low	low	low
Gateway	low	low	low	low
nl.tudelft.sem.gateway.config	low	low	low	low
SecurityConfig	low	low-medium	low	low
SpringCloudConfig	low	low	low	low
nl.tudelft.sem.gateway.controllers	low	low	low	low
GatewayController	low	low	low	low
nl.tudelft.sem.gateway.entities	low	low	low	low
AuthUser	low	low	low	low
CommunicationEntity	low	low	low	low
Feedback	low	low	low-medium	low-medium
User	low	low	low-medium	low-medium
nl.tudelft.sem.gateway.services	low	low	low	low
GatewayService	low	low-medium	low-medium	low
GatewayService(RestTempl:	low	low	low	low
createAccount(Communicat	low	low-medium	low-medium	low
findByUsername(String): M	low	low-medium	low	low
final transient restTemplate :				
final transient urlAuth : Strinç				
final transient usersApi : Strir				

(b) Gateway overview pre refactoring

Figure 14: *Before* refactoring.The Analysis of createAccount in Gateway micro-service.



(a) createAccount metrics post refactoring



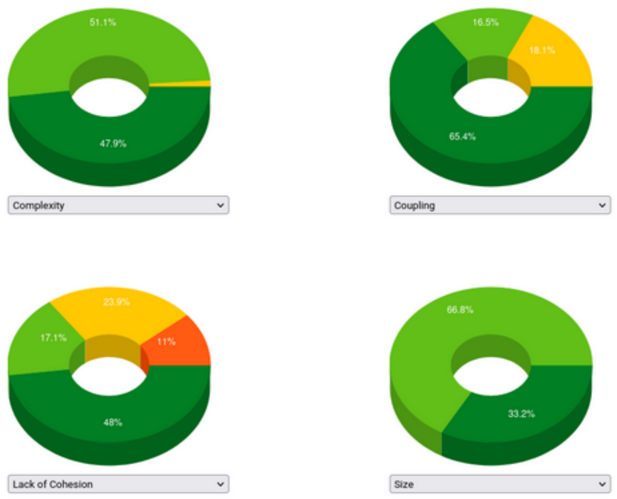
Name	Complexity	Coupling	Size	Lack of Cohesion
sem-repo-01a				
nl.tudelft.sem.gateway	low	low	low	low
Gateway	low	low	low	low
nl.tudelft.sem.gateway.config	low	low	low	low
SecurityConfig	low	low-medium	low	low
SpringCloudConfig	low	low	low	low
nl.tudelft.sem.gateway.controllers	low	low	low	low
GatewayController	low	low	low	low
nl.tudelft.sem.gateway.entities	low	low	low	low
AuthUser	low	low	low	low
CommunicationEntity	low	low	low	low
Feedback	low	low	low-medium	low-medium
User	low-medium	low	low-medium	medium-high
nl.tudelft.sem.gateway.services	low	low	low	low
GatewayService	low	low-medium	low-medium	low
GatewayService(RestTempl:	low	low	low	low
createAccount(Communicat	low	low	low	low
executePOSTS(Communicat	low	low	low	low
findByUsername(String): M	low	low-medium	low	low
getAuthenticatedUser(Strinç	low	low	low	low
getUser(String): User	low	low	low	low
final transient restTemplate :				
final transient urlAuth : Strinç				
final transient usersApi : Strir				

(b) Gateway overview post refactoring

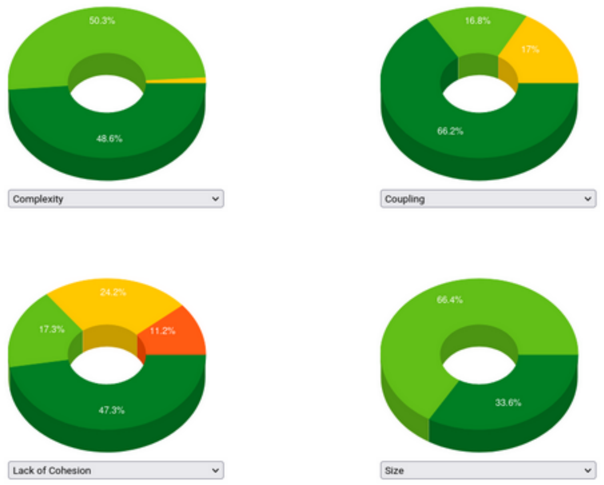
Figure 15: *After* refactoring.The Analysis of createAccount in Gateway micro-service.

4 Conclusion

Overall, we achieved improvements across all important metrics such as Cohesion, Coupling, Size and Complexity. Most notably, we achieved a decrease of 6% of classes with medium-high complexity.



(a) Entire project metrics pre refactoring



(b) Entire project metrics post refactoring

Figure 16: The Analysis of entire project.