Delft University of Technology

SUBJECT: Software Engineering Methods
COURSE CODE: CSE2115

Assignment 2

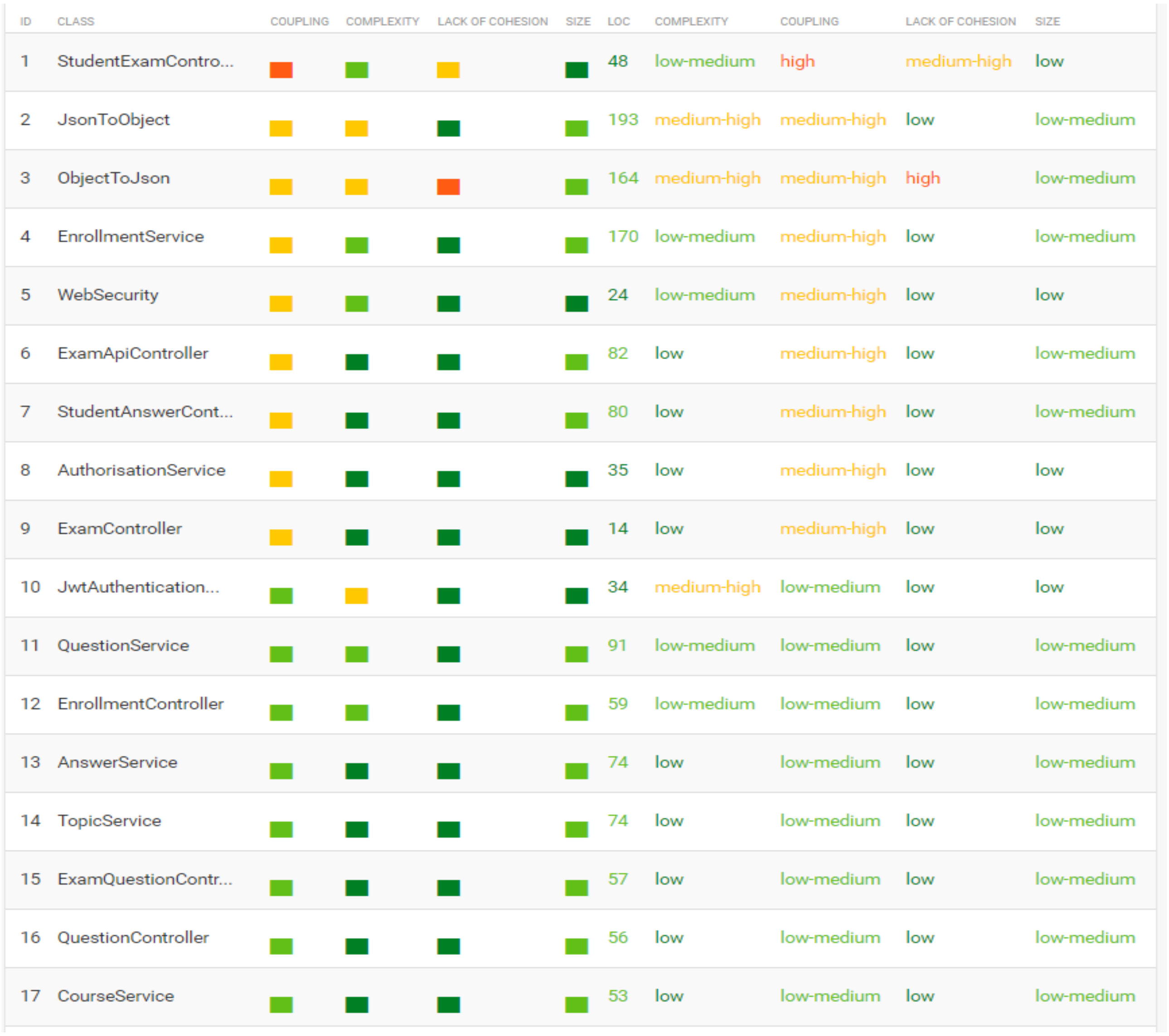
Authors:
Ben Carp
Denis Corlade
Andrejs Karklins
Daniel Poolman
Justin Rademaker
Auke Sonneveld

OP15-SEM29
January 15, 2021

1 Code Metrics

To gain insight in the code-quality of the project, code metrics have to be calculated. For this purpose, we used the static analysis tool 'CodeMR'. For this assignment we focused on the following quality metrics: Coupling, Lack Of Cohesion and Complexity.

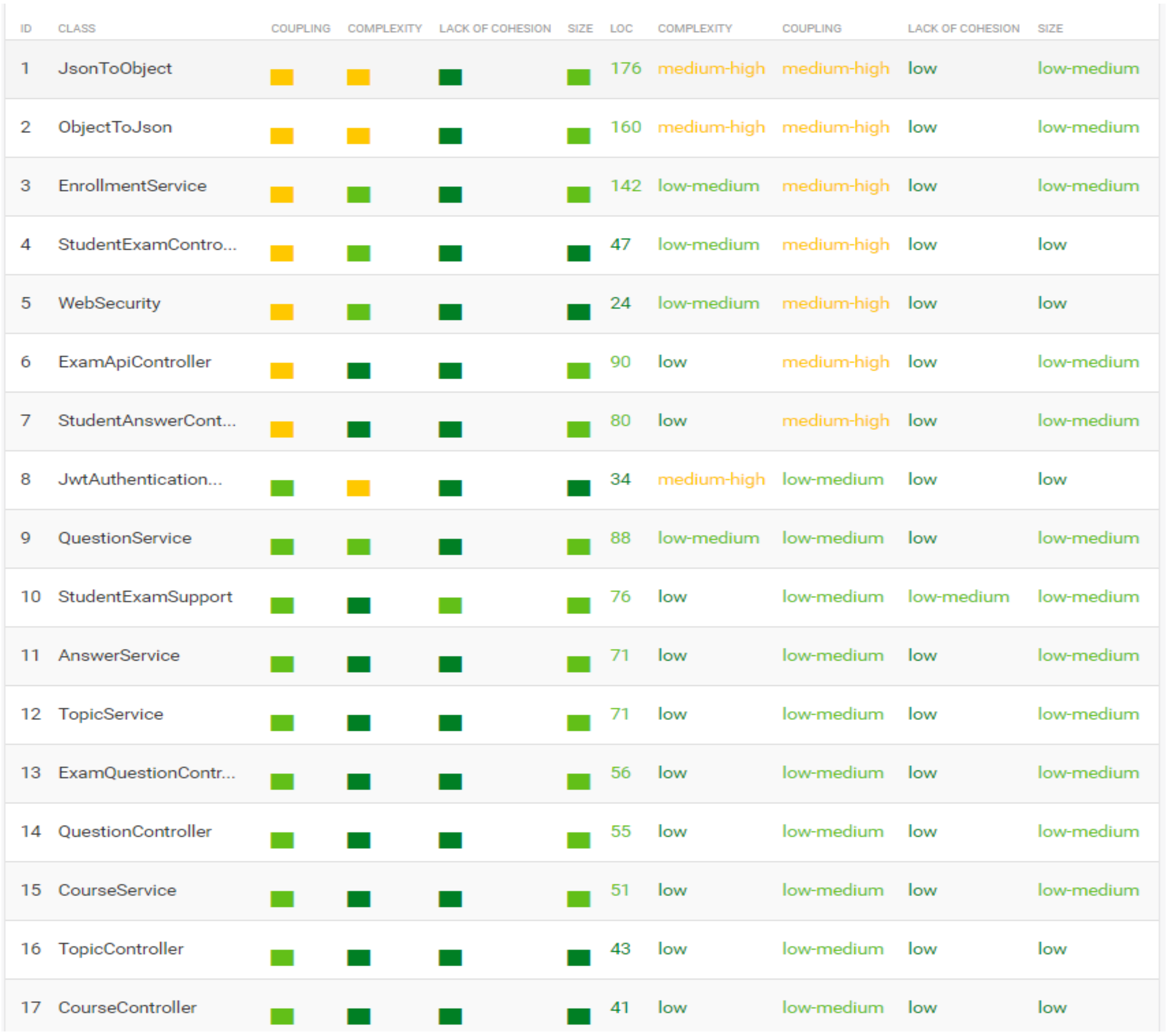
Figure 1 presents a general overview of the top classes with most room for improvement, before refactoring.



ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	StudentExamContro...	🟥	🟩	🟨	🟢	48	low-medium	high	medium-high	low
2	JsonToObject	🟨	🟨	🟢	🟩	193	medium-high	medium-high	low	low-medium
3	ObjectToJson	🟨	🟨	🟥	🟩	164	medium-high	medium-high	high	low-medium
4	EnrollmentService	🟨	🟩	🟢	🟩	170	low-medium	medium-high	low	low-medium
5	WebSecurity	🟨	🟩	🟢	🟢	24	low-medium	medium-high	low	low
6	ExamApiController	🟨	🟢	🟢	🟩	82	low	medium-high	low	low-medium
7	StudentAnswerCont...	🟨	🟢	🟢	🟩	80	low	medium-high	low	low-medium
8	AuthorisationService	🟨	🟢	🟢	🟢	35	low	medium-high	low	low
9	ExamController	🟨	🟢	🟢	🟢	14	low	medium-high	low	low
10	JwtAuthentication...	🟩	🟨	🟢	🟢	34	medium-high	low-medium	low	low
11	QuestionService	🟩	🟩	🟢	🟩	91	low-medium	low-medium	low	low-medium
12	EnrollmentController	🟩	🟩	🟢	🟩	59	low-medium	low-medium	low	low-medium
13	AnswerService	🟩	🟢	🟢	🟩	74	low	low-medium	low	low-medium
14	TopicService	🟩	🟢	🟢	🟩	74	low	low-medium	low	low-medium
15	ExamQuestionContr...	🟩	🟢	🟢	🟩	57	low	low-medium	low	low-medium
16	QuestionController	🟩	🟢	🟢	🟩	56	low	low-medium	low	low-medium
17	CourseService	🟩	🟢	🟢	🟩	53	low	low-medium	low	low-medium

Figure 1: Overview of classes with most room for improvement and refactoring

Figure 2 shows the same overview after refactoring the code. This way, it is easy to see the difference.



ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	JsonToObject	🟨	🟨	■	🟩	176	medium-high	medium-high	low	low-medium
2	ObjectToJson	🟨	🟨	■	🟩	160	medium-high	medium-high	low	low-medium
3	EnrollmentService	🟨	🟩	■	🟩	142	low-medium	medium-high	low	low-medium
4	StudentExamContro...	🟨	🟩	■	■	47	low-medium	medium-high	low	low
5	WebSecurity	🟨	🟩	■	■	24	low-medium	medium-high	low	low
6	ExamApiController	🟨	■	■	🟩	90	low	medium-high	low	low-medium
7	StudentAnswerCont...	🟨	■	■	🟩	80	low	medium-high	low	low-medium
8	JwtAuthentication...	🟩	🟨	■	■	34	medium-high	low-medium	low	low
9	QuestionService	🟩	🟩	■	🟩	88	low-medium	low-medium	low	low-medium
10	StudentExamSupport	🟩	■	🟩	🟩	76	low	low-medium	low-medium	low-medium
11	AnswerService	🟩	■	■	🟩	71	low	low-medium	low	low-medium
12	TopicService	🟩	■	■	🟩	71	low	low-medium	low	low-medium
13	ExamQuestionContr...	🟩	■	■	🟩	56	low	low-medium	low	low-medium
14	QuestionController	🟩	■	■	🟩	55	low	low-medium	low	low-medium
15	CourseService	🟩	■	■	🟩	51	low	low-medium	low	low-medium
16	TopicController	🟩	■	■	■	43	low	low-medium	low	low
17	CourseController	🟩	■	■	■	41	low	low-medium	low	low

Figure 2: Overview of classes with most room for improvement and refactoring

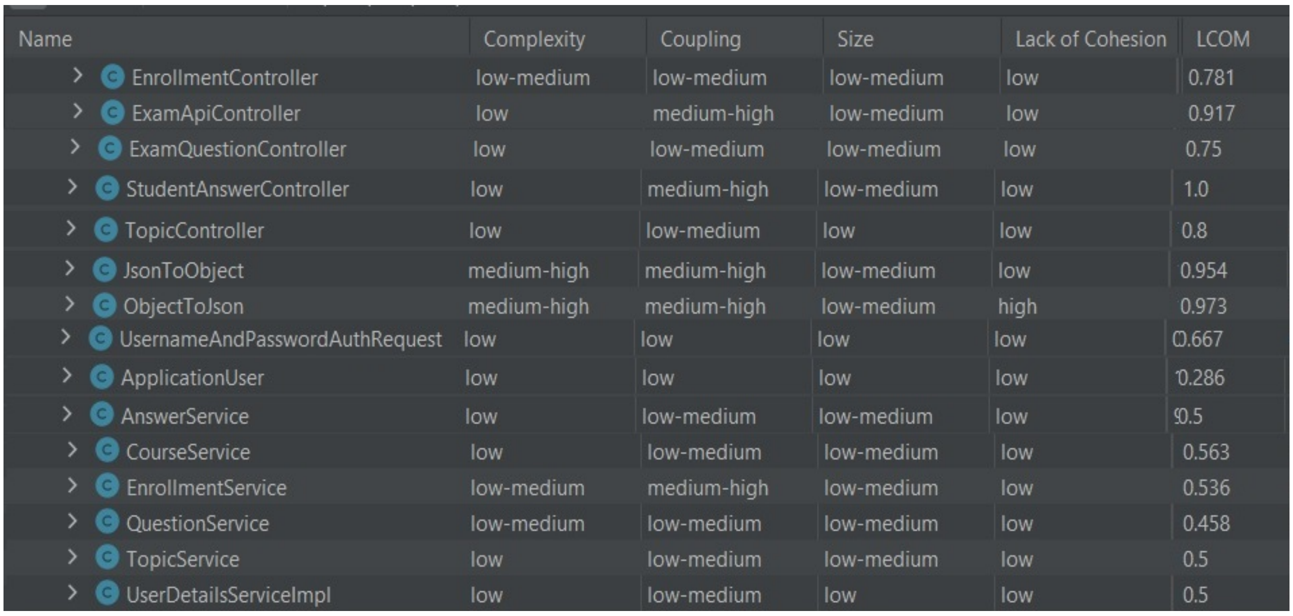
CodeMR specifies thresholds for each metric to classify the state of the application in that area. We used these thresholds to identify which classes and methods were most important to improve. The following classes have been refactored: StudentExamController, EnrollmentService, ObjectToJson, NonAuthorizedException and QuestionService . Also the following methods have been refactored: serialize, deserialize, receiveSession, examQuestionsByCourseId and updateStudentAnswer.

The subsequent sections go more in-depth about the specific refactoring and associated metrics.

2 General Refactoring

The first refactoring operation regards the Lack Of Cohesion Of Methods (LCOM) metric. This measures to what extend the class attributes are used by all class methods. A characteristic of our application was that nearly all classes had certain literals as attributes. This was a response to the PMD requirement to generalize literals when they occur often. However, it lead to a high LCOM.

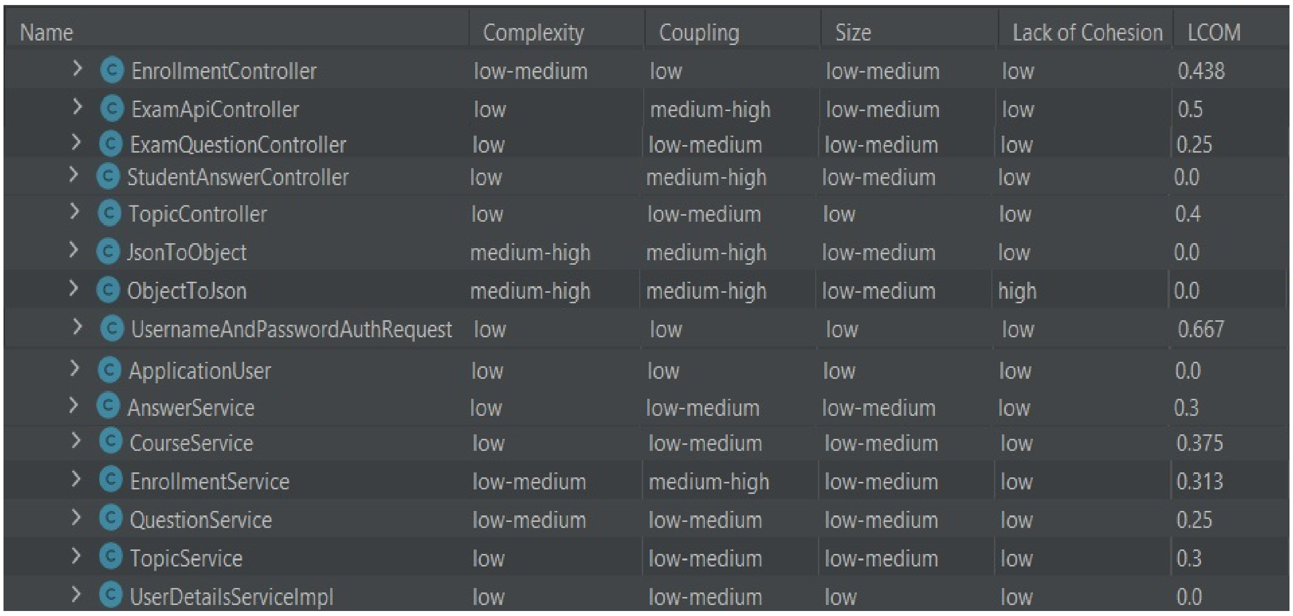
To classify the metric scores, the thresholds provided by CodeMR are used. low: 0 - 0.5, low-medium: 0.5 - 0.7, Medium-high: 0.7 - 0.8, high: 0.8 - 1, Very high: > 1. The following figures shows classes with LCOM higher than 0.



Name	Complexity	Coupling	Size	Lack of Cohesion	LCOM
EnrollmentController	low-medium	low-medium	low-medium	low	0.781
ExamApiController	low	medium-high	low-medium	low	0.917
ExamQuestionController	low	low-medium	low-medium	low	0.75
StudentAnswerController	low	medium-high	low-medium	low	1.0
TopicController	low	low-medium	low	low	0.8
JsonToObject	medium-high	medium-high	low-medium	low	0.954
ObjectToJson	medium-high	medium-high	low-medium	high	0.973
UsernameAndPasswordAuthRequest	low	low	low	low	0.667
ApplicationUser	low	low	low	low	0.286
AnswerService	low	low-medium	low-medium	low	0.5
CourseService	low	low-medium	low-medium	low	0.563
EnrollmentService	low-medium	medium-high	low-medium	low	0.536
QuestionService	low-medium	low-medium	low-medium	low	0.458
TopicService	low	low-medium	low-medium	low	0.5
UserDetailsServiceImpl	low	low-medium	low	low	0.5

Figure 3: Relevant classes for the refactoring regarding LCOM (before refactoring)

To improve this, a new 'Constants' class has been created in the 'shared' module. For all classes (including tests) such attributes have been moved to this Constants class. This coupled the classes to an extra class, but significantly reduced the LCOM. The following image shows the metric scores after refactoring.



Name	Complexity	Coupling	Size	Lack of Cohesion	LCOM
EnrollmentController	low-medium	low	low-medium	low	0.438
ExamApiController	low	medium-high	low-medium	low	0.5
ExamQuestionController	low	low-medium	low-medium	low	0.25
StudentAnswerController	low	medium-high	low-medium	low	0.0
TopicController	low	low-medium	low	low	0.4
JsonToObject	medium-high	medium-high	low-medium	low	0.0
ObjectToJson	medium-high	medium-high	low-medium	high	0.0
UsernameAndPasswordAuthRequest	low	low	low	low	0.667
ApplicationUser	low	low	low	low	0.0
AnswerService	low	low-medium	low-medium	low	0.3
CourseService	low	low-medium	low-medium	low	0.375
EnrollmentService	low-medium	medium-high	low-medium	low	0.313
QuestionService	low-medium	low-medium	low-medium	low	0.25
TopicService	low	low-medium	low-medium	low	0.3
UserDetailsServiceImpl	low	low-medium	low	low	0.0

Figure 4: Relevant classes for the refactoring regarding LCOM (after refactoring)

As can be seen, the old distribution was as follows. high: 6, medium-high: 2, low-medium: 6, low: 2. The new distribution is as follows: low-medium: 2, low: 13.

3 Methods Refactored

3.1 Serialize & Deserialize

Serialization was introduced as part of Assignment 1, following the "Method Factory" design pattern to convert objects to their respective JSON representations and vice versa. The serialize method takes an object, then calls the right method in the ObjectToJson class for serialization, depending on which class the object is

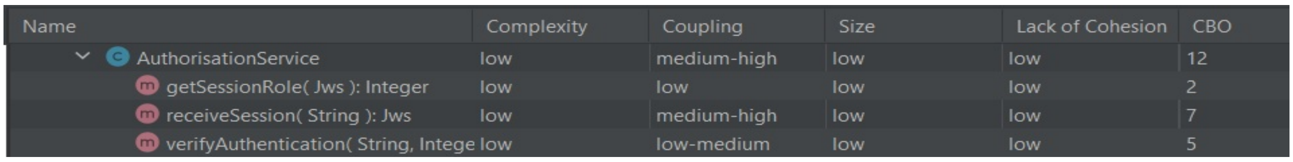
an instance of. The deserialize method does the opposite, taking a String in JSON format and a corresponding class type, then calling the correct method in JsonToObject, which creates a new instance of the given class featuring the fields specified in the JSON.
Previously, both methods used switch cases to call the corresponding method based on class type. This led to an unnecessary "low-medium" complexity of the Serializer class in CodeMR. It also meant that for every new model class, the developer would have to add a new switch case (twice), in addition to the corresponding ObjectToJson and JsonToObject methods. To remove this unneeded complexity, the switch cases were replaced by run-time reflections. The correct conversion method is now computed by using the convention that the method is spelled exactly the same as the class, with the minor change of the first letter being lowercase. The behavior for an unknown model class detected at run-time is the same as the "default" case in the previously used switch statement. CodeMR now awards the Serializer class with a low complexity, and adding new models now no longer involves the Serializer class at all: As long as there are correctly named methods in ObjectToJson and JsonToObject, the Serializer will do its job without fail.

3.2 ReceiveSession

The receiveSession method is part of the Authorisation microservice. CodeMR directs attention to the metric 'Coupling Between Object Classes'(CBO). In the original situation, this method was coupled to seven other classes, contributing to a total CBO of 12 for the class AuthorisationService. CodeMR classifies a CBO of 10-20 as medium-high, whilst classifying a CBO of 5-10 as low-medium.

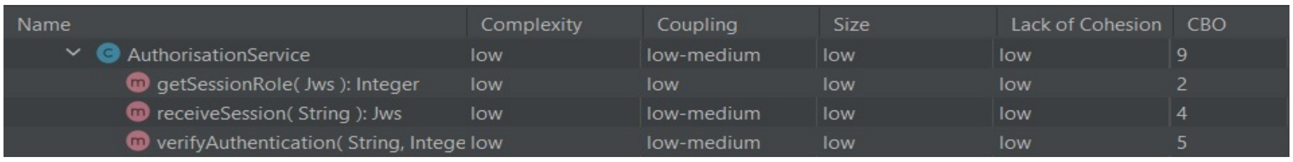
Further investigation showed that the method made use of three different Exception classes to cover all possible exceptions. This was redundant. Another note is that the method is private and used by the method 'verifyAuthentication'. Since the latter method already handled a possible exception, all exception handling in 'receiveSession' could be removed. This reduced the CBO by 3 which promoted the metric to a low-medium level.

Finally, altough the cyclomatic complexity is not measured by CodeMR, this has also been reduced by the refactoring from two to zero.
The following figures show the metric scores before and after the refactoring.



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
AuthorisationService	low	medium-high	low	low	12
getSessionRole(Jws): Integer	low	low	low	low	2
receiveSession(String): Jws	low	medium-high	low	low	7
verifyAuthentication(String, Intege	low	low-medium	low	low	5

Figure 5: Method receiveSession before refactoring



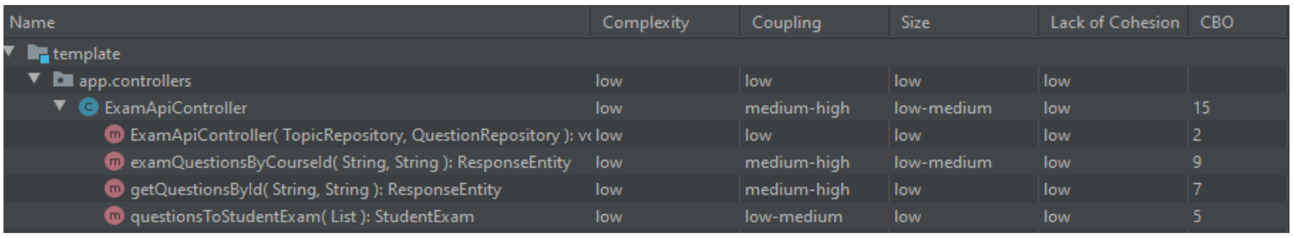
Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
AuthorisationService	low	low-medium	low	low	9
getSessionRole(Jws): Integer	low	low	low	low	2
receiveSession(String): Jws	low	low-medium	low	low	4
verifyAuthentication(String, Intege	low	low-medium	low	low	5

Figure 6: Method receiveSession after refactoring

3.3 ExamQuestionsByCourseId

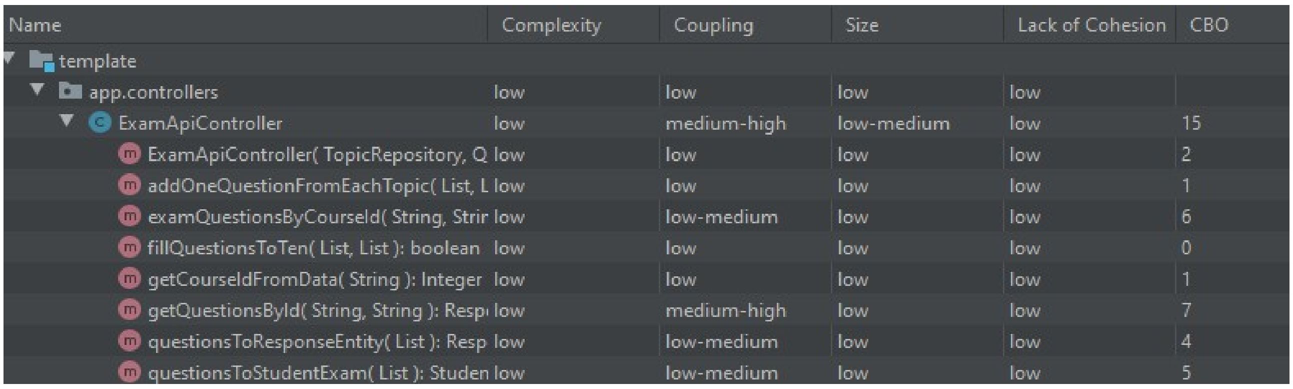
In the class ExamApiController we have the method examQuestionsByCourseId. This method takes as an parameter a courseId and then returns a studentExam containing questions and answers. These questions and answers belong to the topics related to the course with the given courseId. Before refactoring, this method performed all needed operations without calling any helper methods. This caused the method to become rather large and hard to understand. As you can see in figure ... the size meteric of codeMR was evaluated to be low-medium. Furthermore the coupling metric was evaluated to be medium-high. Before refactoring the method had a CBO score of 9. By refactoring the method we reduced the size and the coupling of the method as well as improve overall readability. We did this by creating additional methods and moving functionality to them. These methods are: getCourseIdFromData, addOneQuestionFromEachTopic, fillQuestionsToTen, questionsToResponseEntity and questionsToStudentExam. The names already indicate what each method does but nevertheless we will describe each of them briefly. GetCourseIdFromData parses the string from the HttpRequest body to find the courseId and returns it. AddOneQuestionFromEachTopic takes as parameters a list of questions and a list of topics. The list of questions is empty to start with and the method will add questions to it. The method takes one question from each topic in the list of topics and adds it to the list of questions. The method returns a boolean indicating whether or not it was successful. FillQuestionsToTen takes the same parameters as before. Now the list of questions will already contain the questions added by addOneQuestionFromEachTopic. This method will add random questions from the topics until there are ten. Similar as before it will return a boolean indicating whether it succeeded. QuestionsToStudentExam takes as a parameter the list of questions. It will create a StudentExam and insert the questions into it. It will then return this StudentExam. QuestionsToResponseEntity takes as a parameter the list of questions and uses it to call questionsToStudentExam. The received StudentExam is put parsed to a ResponseEntity and returned.

After moving functionality and dependencies to these methods we can see a clear improvement in the codeMR analysis. As you can see in figure ... the size metric was reduced to low and the coupling metric was reduced to low-medium. We can also see that the CBO score went from 9 to 6.



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
template					
app.controllers	low	low	low	low	
ExamApiController	low	medium-high	low-medium	low	15
ExamApiController(TopicRepository, QuestionRepository): v	low	low	low	low	2
examQuestionsByCourseId(String, String): ResponseEntity	low	medium-high	low-medium	low	9
getQuestionsById(String, String): ResponseEntity	low	medium-high	low	low	7
questionsToStudentExam(List): StudentExam	low	low-medium	low	low	5

Figure 7: Method examQuestionsByCourseId before refactoring

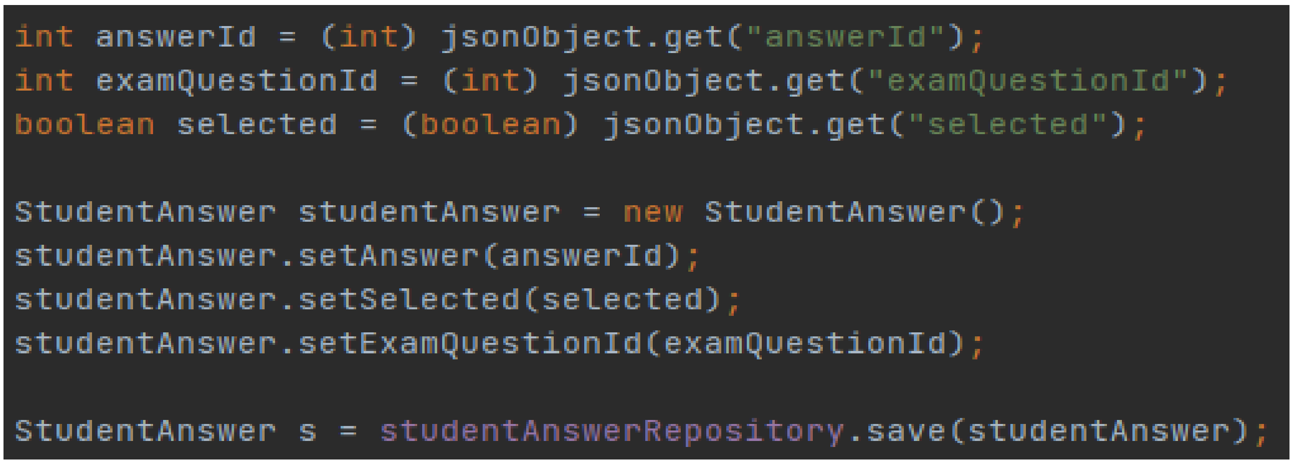


Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
template					
app.controllers	low	low	low	low	
ExamApiController	low	medium-high	low-medium	low	15
ExamApiController(TopicRepository, Q	low	low	low	low	2
addOneQuestionFromEachTopic(List, L	low	low	low	low	1
examQuestionsByCourseId(String, Strir	low	low-medium	low	low	6
fillQuestionsToTen(List, List): boolean	low	low	low	low	0
getCourseIdFromData(String): Integer	low	low	low	low	1
getQuestionsById(String, String): Resp	low	medium-high	low	low	7
questionsToResponseEntity(List): Resp	low	low-medium	low	low	4
questionsToStudentExam(List): Studen	low	low-medium	low	low	5

Figure 8: Method examQuestionsByCourseId after refactoring

3.4 UpdateStudentAnswer

UpdateStudentAnswer method was unreadable and contained unnecessary lines of code. After refactoring reduced Lines of Code, as also improved readability. Since methods nature is to convert json string to object and save it to database, it was redundant to do conversion from json to object by hand, because there was already implementation for that in serializer. Unfortunately it was impossible to reduce coupling in the method, due to structure of our application.

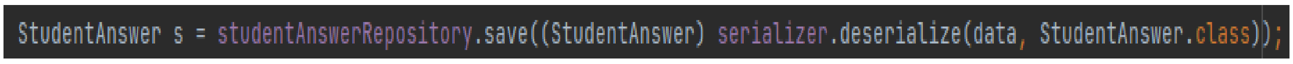


```
int answerId = (int) jsonObject.get("answerId");
int examQuestionId = (int) jsonObject.get("examQuestionId");
boolean selected = (boolean) jsonObject.get("selected");

StudentAnswer studentAnswer = new StudentAnswer();
studentAnswer.setAnswer(answerId);
studentAnswer.setSelected(selected);
studentAnswer.setExamQuestionId(examQuestionId);

StudentAnswer s = studentAnswerRepository.save(studentAnswer);
```

Figure 9: Method updateStudentAnswer before refactoring



```
StudentAnswer s = studentAnswerRepository.save((StudentAnswer) serializer.deserialize(data, StudentAnswer.class));
```

Figure 10: Method updateStudentAnswer after refactoring

4 Classes Refactored

4.1 StudentExamController

StudentExamController had Lack of Cohesion. Methods of the class were huge and we have decided to split big methods into smaller pieces. This solution was extremely useful, but raised previously mentioned class code smell. Tackling of this problem was quite easy,and so we have decided to create a StudentExamSupport class, which would take responsibilities of all helper methods. This made code more readable, as also this change reduced class coupling, which was an additional benefit gained from the refactoring.

4.2 EnrollmentService

The EnrollmentService had medium-high coupling. Every method in the EnrollmentService was using a try-catch structure. In the try-block an attempt was made to perform some operation on the database, e.g. retrieving some data. When this operation would fail, because for instance the data wasn't there, in the try-block we would throw an exception that we would then immediately catch and deal with in the catch-block. First of all, this is an inefficient and overly complex way to deal with the failure. Secondly because of this structure, the EnrollmentService now depends on Exception. By removing the try-catch block, we remove this dependency and thus lower the coupling. Furthermore, this also improves the general code quality and readability of the class. We can achieve this refactoring by moving the contents of the catch-block to the location where we throw the exception.

4.3 ObjectToJson

The ObjectToJson class converts objects to their JSON representations, featuring one static method per class. Before refactoring, the ObjectToJson class suffered from very low (= high lack of) cohesion. This can be attributed to the fact that every method had a different type of parameter (An object of the respective class), while the Serializer took care of casting.
To raise cohesion, every method now accepts the same parameter: a plain object (instance of Object class), passed by the serializer, which is no longer responsible for casting. Instead, each method now casts the generic object it receives to the subtype of object itself. ObjectToJson now features high (= low lack of) cohesion, while the serializer focuses exclusively on its core task.

4.4 NotAuthorizedException

The NotAuthorizedException is a custom exception that is used whenever a user is not authorized to perform a certain task. The NotAuthorizedException had a medium high complexity, because it was hard to understand how it worked. To decrease the complexity, we decided to remove the NotAuthorizedException and go with a more generic exception, SecurityException. This exception is included with the base Java libraries. Our code still functions the same, but it's more generic and better understandable.

4.5 QuestionService

The QuestionService is used to create, read, update and delete questions. The most important method in this class is the "getRandomQuestions" method, because it provides the random questions that are used during an exam. The class itself had a low-medium complexity, but there were still a few quirks, like try-catch blocks that weren't needed. After refactoring the code, the complexity went down to low.