Assignment 2 - group 15a

Part 1

To compute software code metrics, we decided to use the Intelij plugin CodeMR software. At the method level, we looked at the method level metrics generated by CodeMR such as complexity, coupling, size, lack of cohesion, McCabe cyclomatic complexity (MCC), LOC, method calls (MC), etc. At the class level, we took into consideration the complexity, coupling, size, lack of cohesion, coupling between objects (CBO), lack of cohesion among methods (LCAM), lack of cohesion of methods (LCOM), etc. When identifying problematic methods or classes, we used the color coding (dark green, green, orange, yellow) as an indication of problematic areas of the software before diving deeper into the actual method values. Our metric values overall were mostly in the low region so we focused on those that were the worst out of all of them. However, we did these selections with some of the key metrics with an problematic threshold in mind, namely:

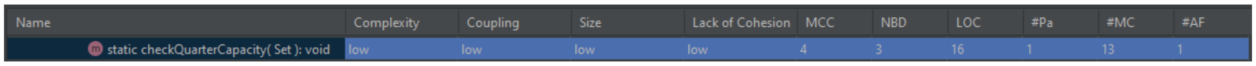
- Complexity > low-medium
- Coupling > low-medium
- Size > low - big methods and classes are hard to read and process, as well as the logic inside huge method is difficult, thus, is easily breakable by adding new features
- Lack of cohesion > low-medium
- MCC > 4 - too complex methods may introduce bugs and hence we should limit the possible decision points in methods
- MC > 10 - having higher metric here may introduce higher coupling due to relying too much on other methods
- CBO > 9 - according to CodeMR, values above 10 are considered medium-high. Having in mind the simplicity of our project, we decided to bring the coupling below.

For other metrics that were more abstract and difficult to meaningfully select a threshold, we used a selecting the worst of the batch approach.

Part 2

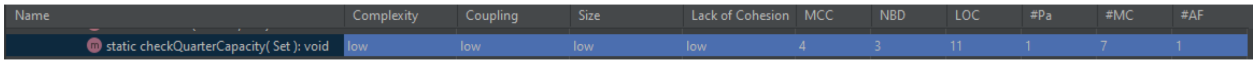
Method Refactoring

Method checkQuarterCapacity in course/…/java/services/StudentService.java



Name	Complexity	Coupling	Size	Lack of Cohesion	MCC	NBD	LOC	#Pa	#MC	#AF
static checkQuarterCapacity(Set): void	low	low	low	low	4	3	16	1	13	1

The method has a very high number of method calls of 13. This is one of the highest values for this field in the Course module. This is attributed to some String parsing logic used to extract data out of a string. The refactoring applied is Extract Method refactoring of the string parsing logic.

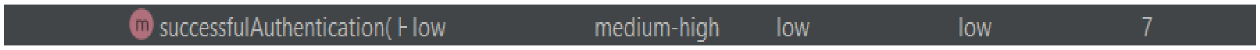


Name	Complexity	Coupling	Size	Lack of Cohesion	MCC	NBD	LOC	#Pa	#MC	#AF
static checkQuarterCapacity(Set): void	low	low	low	low	4	3	11	1	7	1

After refactoring, the number of method calls reduces to 7.
Commit **4ad63251**

Method successfulAuthentication in authentication/…/java/filter/CustomAuthenticationFilter.java

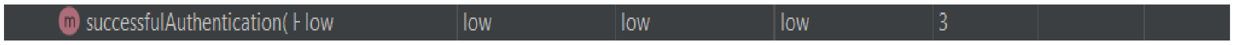
Before refactoring:



successfulAuthentication(H	low	medium-high	low	low	7

The method has a very high coupling due to its CBO (Coupling Between Objects) having a value of 7. This is caused by the method having to do a couple of different functionalities that can be refactored. The method was trying to get the recent Authentication object, create a JWT token and put this token in a response object. These should have been different methods overall, and have been extracted using the Extract Method. This extraction decreased CBO to 3.

After refactoring:



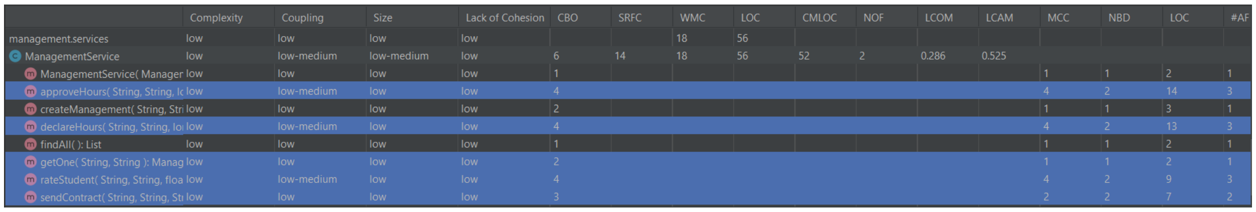
successfulAuthentication(H	low	low	low	low	3		

Commit: **92f7336**

Method in management/…/java/services/ManagementService.java

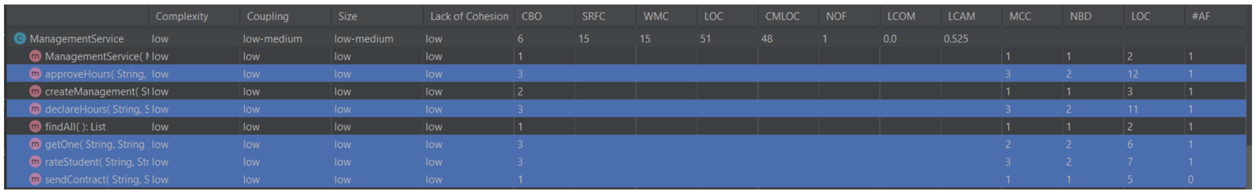
Previously methods **declareHours**, **approveHours**, **rateStudent** and **sendContract** were each unnecessary checking whether a Management object exists in the database. We have applied the Extract Method Technique to move this check into the **getOne** method. This way the duplication of logic has been reduced and ultimately the Cyclomatic Complexity and the Lines of

Code. The coupling has also reduced and the cohesion has increased, because we only throw the exception at one place and remove the duplication of logic.
Before refactoring:



	Complexity	Coupling	Size	Lack of Cohesion	CBO	SRFC	WMC	LOC	CMLOC	NOF	LCOM	LCAM	MCC	NBD	LOC	#AF
management.services	low	low	low	low			18	56								
ManagementService	low	low-medium	low-medium	low	6	14	18	56	52	2	0.286	0.525				
ManagementService(Manager	low	low	low	low	1								1	1	2	1
approveHours(String, String, lc	low	low-medium	low	low	4								4	2	14	3
createManagement(String, Stri	low	low	low	low	2								1	1	3	1
declareHours(String, String, lo	low	low-medium	low	low	4								4	2	13	3
findAll( ): List	low	low	low	low	1								1	1	2	1
getOne(String, String): Manag	low	low	low	low	2								1	1	2	1
rateStudent(String, String, floa	low	low-medium	low	low	4								4	2	9	3
sendContract(String, String, St	low	low	low	low	3								2	2	7	2

After refactoring:

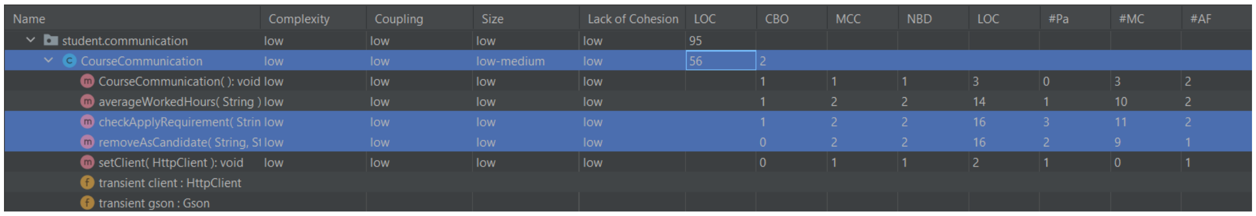


	Complexity	Coupling	Size	Lack of Cohesion	CBO	SRFC	WMC	LOC	CMLOC	NOF	LCOM	LCAM	MCC	NBD	LOC	#AF
ManagementService	low	low-medium	low-medium	low	6	15	15	51	48	1	0.0	0.525				
ManagementService(N	low	low	low	low	1								1	1	2	1
approveHours(String,	low	low	low	low	3								3	2	12	1
createManagement(St	low	low	low	low	2								1	1	3	1
declareHours(String, S	low	low	low	low	3								3	2	11	1
findAll( ): List	low	low	low	low	1								1	1	2	1
getOne(String, String	low	low	low	low	3								2	2	6	1
rateStudent(String, Str	low	low	low	low	3								3	2	7	1
sendContract(String, S	low	low	low	low	1								1	1	5	0

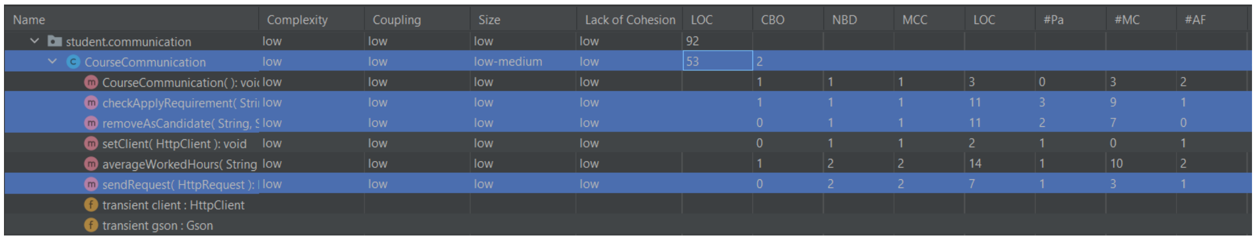
Commit **69926d8f**

Methods checkApplyRequirement() and removeAsCandidate() in student/…/java/communication/CourseCommunication.java

Although they are not necessarily problematic from looking at the metrics, there was some code duplication in the checkApplyRequirement() and removeAsCandidate() methods. They both have the same process of sending the request to the Course microservice, and checking whether the request succeeded. Therefore we used the Extract Method refactoring tactic here.



Name	Complexity	Coupling	Size	Lack of Cohesion	LOC	CBO	MCC	NBD	LOC	#Pa	#MC	#AF
student.communication	low	low	low	low	95							
CourseCommunication	low	low	low-medium	low	56	2						
CourseCommunication( ): void	low	low	low	low		1	1	1	3	0	3	2
averageWorkedHours(String)	low	low	low	low		1	2	2	14	1	10	2
checkApplyRequirement(Strin	low	low	low	low		1	2	2	16	3	11	2
removeAsCandidate(String, St	low	low	low	low		0	2	2	16	2	9	1
setClient(HttpClient): void	low	low	low	low		0	1	1	2	1	0	1
transient client : HttpClient												
transient gson : Gson												



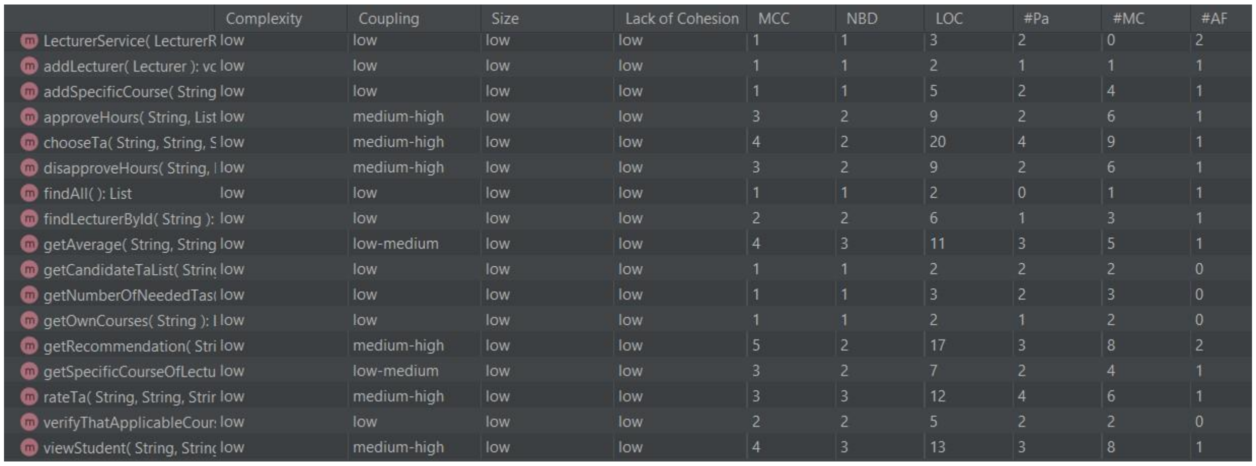
Name	Complexity	Coupling	Size	Lack of Cohesion	LOC	CBO	NBD	MCC	LOC	#Pa	#MC	#AF
student.communication	low	low	low	low	92							
CourseCommunication	low	low	low-medium	low	53	2						
CourseCommunication( ): voi	low	low	low	low		1	1	1	3	0	3	2
checkApplyRequirement(Stri	low	low	low	low		1	1	1	11	3	9	1
removeAsCandidate(String, S	low	low	low	low		0	1	1	11	2	7	0
setClient(HttpClient): void	low	low	low	low		0	1	1	2	1	0	1
averageWorkedHours(String	low	low	low	low		1	2	2	14	1	10	2
sendRequest(HttpRequest):	low	low	low	low		0	2	2	7	1	3	1
transient client : HttpClient												
transient gson : Gson												

We can see that after the refactoring, the total lines of code in the class lowered from 56 to 53. There were 5 lines of code in both methods that have been moved to the new sendRequest() method, which is still a decent improvement, especially if additional features were to be implemented that also follow this communication pattern.
Commit **a7a8d503**

Methods in java/…/lecturer/services/LecturerService.java

approveHours, chooseTA, disapproveHours, getAverage, getRecommendation, getSpecificCourseOfLecturer, viewStudent

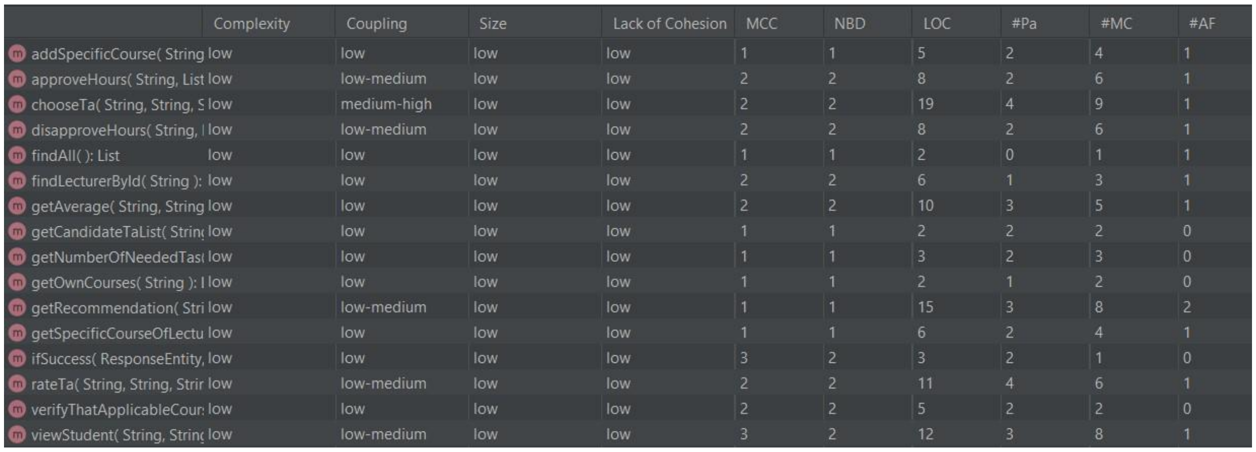
Initial:



	Complexity	Coupling	Size	Lack of Cohesion	MCC	NBD	LOC	#Pa	#MC	#AF
LecturerService(LecturerR	low	low	low	low	1	1	3	2	0	2
addLecturer(Lecturer): vc	low	low	low	low	1	1	2	1	1	1
addSpecificCourse(String	low	low	low	low	1	1	5	2	4	1
approveHours(String, List	low	medium-high	low	low	3	2	9	2	6	1
chooseTa(String, String, S	low	medium-high	low	low	4	2	20	4	9	1
disapproveHours(String, l	low	medium-high	low	low	3	2	9	2	6	1
findAll( ): List	low	low	low	low	1	1	2	0	1	1
findLecturerById(String):	low	low	low	low	2	2	6	1	3	1
getAverage(String, String	low	low-medium	low	low	4	3	11	3	5	1
getCandidateTaList(Strin	low	low	low	low	1	1	2	2	2	0
getNumberOfNeededTas	low	low	low	low	1	1	3	2	3	0
getOwnCourses(String): l	low	low	low	low	1	1	2	1	2	0
getRecommendation(Stri	low	medium-high	low	low	5	2	17	3	8	2
getSpecificCourseOfLectu	low	low-medium	low	low	3	2	7	2	4	1
rateTa(String, String, Strir	low	medium-high	low	low	3	3	12	4	6	1
verifyThatApplicableCour	low	low	low	low	2	2	5	2	2	0
viewStudent(String, Strin	low	medium-high	low	low	4	3	13	3	8	1

As we see, coupling of some methods is high, so an attempt was made to lower it and also lower MCC. This was made by extracting duplicate code from multiple functions. Previously all methods checked for code 200, sometimes even twicely, this was substituted with a function. This way, methods do not access status internally as well as exceptions - this all happens in helper function.

Result:



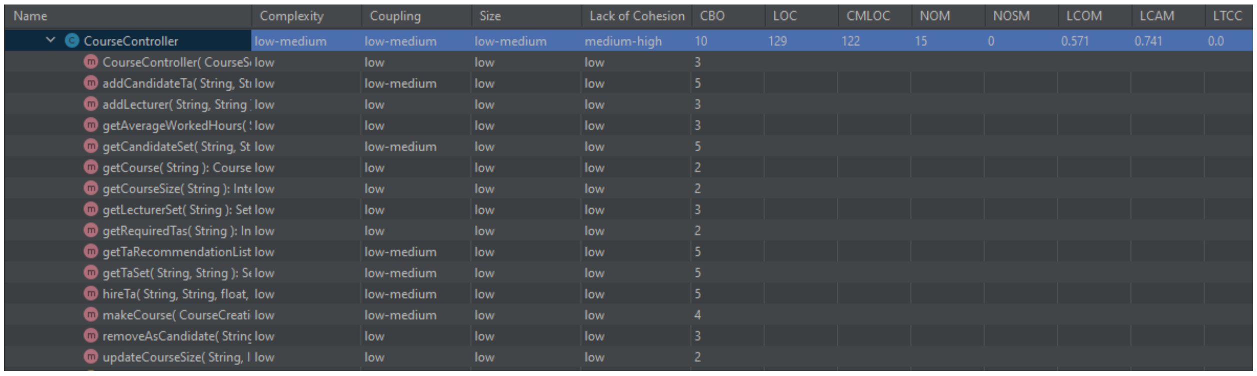
	Complexity	Coupling	Size	Lack of Cohesion	MCC	NBD	LOC	#Pa	#MC	#AF
addSpecificCourse(String	low	low	low	low	1	1	5	2	4	1
approveHours(String, List	low	low-medium	low	low	2	2	8	2	6	1
chooseTa(String, String, S	low	medium-high	low	low	2	2	19	4	9	1
disapproveHours(String, l	low	low-medium	low	low	2	2	8	2	6	1
findAll( ): List	low	low	low	low	1	1	2	0	1	1
findLecturerById(String):	low	low	low	low	2	2	6	1	3	1
getAverage(String, String	low	low	low	low	2	2	10	3	5	1
getCandidateTaList(Strin	low	low	low	low	1	1	2	2	2	0
getNumberOfNeededTas	low	low	low	low	1	1	3	2	3	0
getOwnCourses(String): l	low	low	low	low	1	1	2	1	2	0
getRecommendation(Stri	low	low-medium	low	low	1	1	15	3	8	2
getSpecificCourseOfLectu	low	low	low	low	1	1	6	2	4	1
ifSuccess(ResponseEntity,	low	low	low	low	3	2	3	2	1	0
rateTa(String, String, Strir	low	low-medium	low	low	2	2	11	4	6	1
verifyThatApplicableCour	low	low	low	low	2	2	5	2	2	0
viewStudent(String, Strin	low	low-medium	low	low	3	2	12	3	8	1

As seen, during refactoring, MCC got lower, as well as NBD and LOC. A new function ifSuccess also has low parameters, which confirms that after refactoring code is more structured.

Commits: **d111b879**, **5864b164**, **1ec67975**, **1b4c3c21**

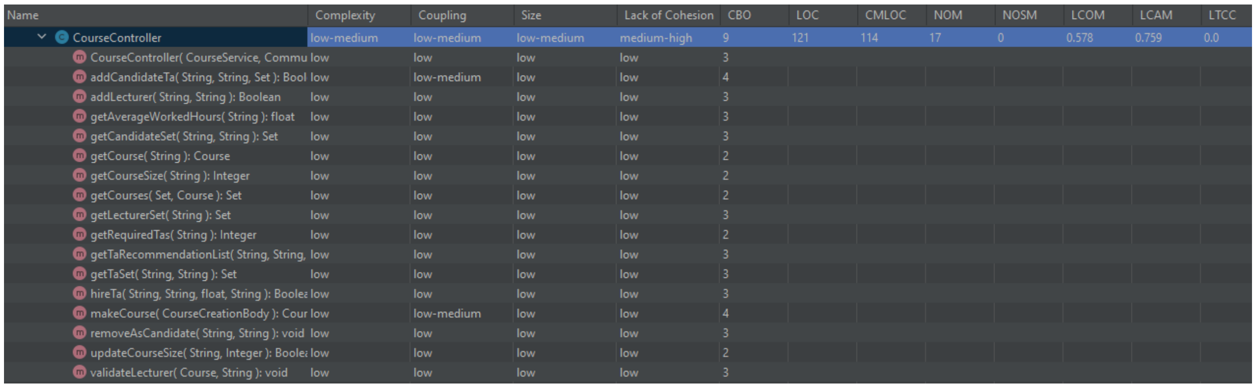
Class Refactoring

Class CourseController in course/…/java/controllers/CourseController.java



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	LOC	CMLOC	NOM	NOSM	LCOM	LCAM	LTCC
CourseController	low-medium	low-medium	low-medium	medium-high	10	129	122	15	0	0.571	0.741	0.0
CourseController(CourseS	low	low	low	low	3							
addCandidateTa(String, St	low	low-medium	low	low	5							
addLecturer(String, String	low	low	low	low	3							
getAverageWorkedHours(	low	low	low	low	3							
getCandidateSet(String, St	low	low-medium	low	low	5							
getCourse(String): Course	low	low	low	low	2							
getCourseSize(String): Int	low	low	low	low	2							
getLecturerSet(String): Set	low	low	low	low	3							
getRequiredTas(String): In	low	low	low	low	2							
getTaRecommendationList	low	low-medium	low	low	5							
getTaSet(String, String): S	low	low-medium	low	low	5							
hireTa(String, String, float,	low	low-medium	low	low	5							
makeCourse(CourseCreati	low	low-medium	low	low	4							
removeAsCandidate(String	low	low	low	low	3							
updateCourseSize(String, I	low	low	low	low	2							

The method CourseController has one of the worst metrics out of all the other classes in Course microservice. It has low-medium complexity and size, medium-high lack of cohesion, CBO of 10, and relatively high LCOM and LCAM metrics compared to other classes. As a controller class though, it takes incoming request inputs and delegates other classes to handle the logic. Therefore, the level of cohesion of the class would not be high since methods “endpoints” may have quite different behaviour. However, the CBO value does indicate that the controller is interacting with a lot of other classes. Upon inspection, the majority of methods implement some sort of logic that should belong in a different class. A case of coupled classes.
The refactoring method we used was primarily to move method from the CourseController classes to the appropriate service or entity class.



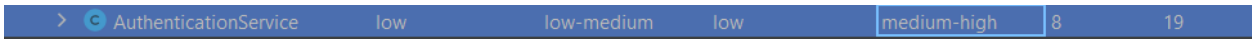
Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	LOC	CMLOC	NOM	NOSM	LCOM	LCAM	LTCC
CourseController	low-medium	low-medium	low-medium	medium-high	9	121	114	17	0	0.578	0.759	0.0
CourseController(CourseService, Commu	low	low	low	low	3							
addCandidateTa(String, String, Set): Bool	low	low-medium	low	low	4							
addLecturer(String, String): Boolean	low	low	low	low	3							
getAverageWorkedHours(String): float	low	low	low	low	3							
getCandidateSet(String, String): Set	low	low	low	low	3							
getCourse(String): Course	low	low	low	low	2							
getCourseSize(String): Integer	low	low	low	low	2							
getCourses(Set, Course): Set	low	low	low	low	2							
getLecturerSet(String): Set	low	low	low	low	3							
getRequiredTas(String): Integer	low	low	low	low	2							
getTaRecommendationList(String, String,	low	low	low	low	3							
getTaSet(String, String): Set	low	low	low	low	3							
hireTa(String, String, float, String): Boolea	low	low	low	low	3							
makeCourse(CourseCreationBody): Cour	low	low-medium	low	low	4							
removeAsCandidate(String, String): void	low	low	low	low	3							
updateCourseSize(String, Integer): Boolea	low	low	low	low	2							
validateLecturer(Course, String): void	low	low	low	low	3							

After refactoring, the CBO value for the class decreased by 1, and as well the coupling metric of 4 methods reduced from low-medium to low. However the coupling of methods within the class such as LCOM and LCAM slightly increased, this is expected since CourseController now almost processes no logic and rather delegates most of them. Therefore methods will be less cohesive. Another benefit of this refactoring is that the CourseController test suite simplified since the logic it was implementing before no longer needed to be tested by its respective test suite.

Commit **95f979ec** and **a6c65caf**

Class AuthenticationService in authentication/…/java/services/AuthenticationService.java

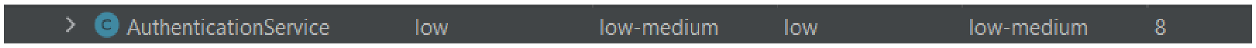
Before refactoring:



AuthenticationService	low	low-medium	low	medium-high	8	19

AuthenticationService class had a medium-high Lack of Cohesion due to some of the methods added to the class becoming redundant over the course of the lifetime of the project. As some of the functionalities of the class were decided to be not used and some of the methods temporarily added to the class for the sake of easier development, removing these methods became a priority after the development cycle. One of the methods to be used as an example is the findAll() method. This method was used to manually test the H2 database, however became completely redundant after the progress of the project.

After refactoring:



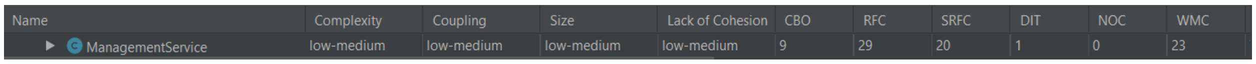
AuthenticationService	low	low-medium	low	low-medium	8

Commit: **92f7336**

Class InvalidHoursException in management/…/java/exceptions/InvalidHoursException .java and Class ManagementService in management/…/java/services/ManagementService.java

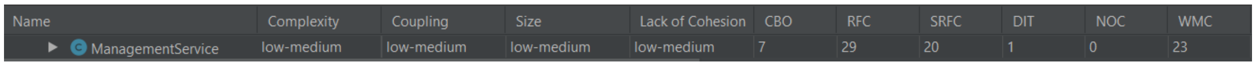
There was a coupling between objects in ManagementService of 9. We noticed that there were multiple redundant custom exception throws such as InvalidContractHoursException, InvalidApprovedHoursException, InvalidDisapprovedHoursException. Hence we have generalized those 3 exceptions in a single InvalidHoursException, which reduced the CBO metric to 7 in ManagementService.

Before refactoring:



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT	NOC	WMC
ManagementService	low-medium	low-medium	low-medium	low-medium	9	29	20	1	0	23

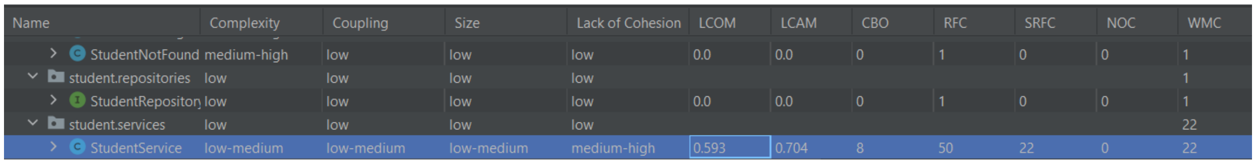
After refactoring:



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT	NOC	WMC
ManagementService	low-medium	low-medium	low-medium	low-medium	7	29	20	1	0	23

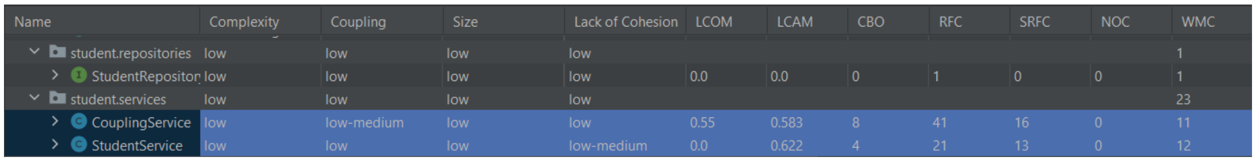
Commit **4bf48e62**

Class StudentService in student/…/java/service/StudentService.java



Name	Complexity	Coupling	Size	Lack of Cohesion	LCOM	LCAM	CBO	RFC	SRFC	NOC	WMC
StudentNotFound	medium-high	low	low	low	0.0	0.0	0	1	0	0	1
student.repositories	low	low	low	low							1
StudentRepositon	low	low	low	low	0.0	0.0	0	1	0	0	1
student.services	low	low	low	low							22
StudentService	low-medium	low-medium	low-medium	medium-high	0.593	0.704	8	50	22	0	22

The StudentService had the worst metrics in the Student microservice. It has medium-high Lack of Cohesion, and low-medium in the other main groups. To combat this, we decided to move part of the functionality to a new class with the Extract Class refactoring tactic.
All of the methods that required sending a request to other microservices, using the CourseCommunication and ManagementCommunication classes, have been moved to the new CouplingService class. CouplingService still has a dependency on StudentService, and both are used in the StudentController class. This gave much better metrics in all categories:

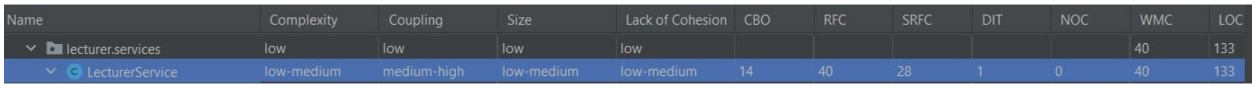


Name	Complexity	Coupling	Size	Lack of Cohesion	LCOM	LCAM	CBO	RFC	SRFC	NOC	WMC
student.repositories	low	low	low	low							1
StudentRepositon	low	low	low	low	0.0	0.0	0	1	0	0	1
student.services	low	low	low	low							23
CouplingService	low	low-medium	low	low	0.55	0.583	8	41	16	0	11
StudentService	low	low	low	low-medium	0.0	0.622	4	21	13	0	12

The LCOM value of the original StudentService dropped to 0, because it doesn't have a dependency on the Communication classes anymore. And the new CouplingService dropped a bit of the original value as well. The other LoC metric, LCAM, lowered in both as well.

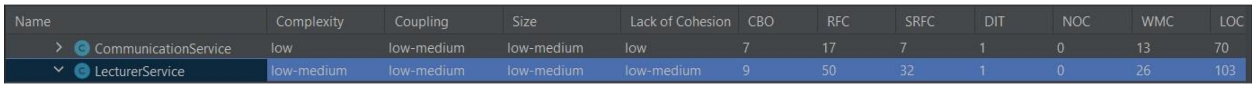
Commit **6567fd62**

Class LecturerService in java/…/lecturer/services/LecturerService.java:



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT	NOC	WMC	LOC
lecturer.services	low	low	low	low						40	133
LecturerService	low-medium	medium-high	low-medium	low-medium	14	40	28	1	0	40	133

As seen, coupling is quite high - CBO is 14. This is much higher than other classes, and the class is not isolated and it accesses lots of other “foreign” data.
The decision was to isolate all communication in additional service and connect lecturer service with that one. Right now communicationService will handle all communication between microservices as well as failor of it.



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	DIT	NOC	WMC	LOC
CommunicationService	low	low-medium	low-medium	low	7	17	7	1	0	13	70
LecturerService	low-medium	low-medium	low-medium	low-medium	9	50	32	1	0	26	103

Here, coupling is lower (CBO got from 14 to 9), plus the code is more structured now. CommunicationService also has low parameters, which means the isolation of it does not overload it.

Commit: **e48c3323**, **35a9f74a**

*Note! Because of this refactoring, method refactoring described above got “lost”. However, the idea used there still hold for new CommunicationService - code is checked in separate function to avoid duplication & high coupling of methods in CommunicationService