Assignment 2

OP3-SEM6

Eva Noritsyna, Carlotta Lichtenauer, Annabel Simons,
Mathanrajan Sundarrajan, Yuanze Xiong and Matteo Fregonara

January 15, 2021

1 Part 1: Code metrics

1.1 Introduction into *CodeMR* and metric-diagrams

We chose *CodeMR* to analyze our system regarding our system's code quality. *CodeMR* creates a visual representation of certain quality attributes and of the code structure. With these visuals, we could identify where our code had bad-code smells and apply related refactoring methods to them. Figure 1 shows how the level of the metrics are classified depending on certain conditions. As one can see lower number of metrics conclude to better code quality. For our code quality analysis, we chose to look for the bad-code smells related to Coupling, Complexity, Lack of Cohesion, and Size as they are the basis of the Chidamer and Kemerer Metrics.

We chose the Coupling metrics (CBO) since it shows the connection between two classes. We want to have this as low since we want our system to be as failure robust as possible. When having low Coupling we reduce the dependency between classes, concluding that if one breaks the other one doesn't.

Secondly, we chose to focus on the metrics of Complexity (WMC, RFC, DIT) since our system should be designed in such a way that other engineers can easily contribute to the code.

Thirdly, we chose the Lack of Cohesion metrics (LCAM) since it measures how well methods inside a class are related to each other. If this number is high it shows that a class has a lot of logic that is not connected. By improving on this part the robustness, reliability, reusability, and understandability of our code are increasing.

Lastly, we focused on the length of our classes (LOC) and the number of methods (NOM) inside of it. Since it becomes harder to maintain the code if it's overcrowded.

For all microservices, we generated the corresponding *CodeMR* diagrams. (See Figures 2, 4 and 6). Based on that, we chose our methods and classes to improve. Also, we decided to look more detailed into the Course microservice since its numbers were the highest. (See Figures 3, 5 and 7).

1.2 Explanation of the threshold

Looking at the individual microservice figures/tables, our goal is to fix every class that has a metric level of medium-high or above. A threshold of medium-high for Coupling, Size and Complexity is good since looking at the conditions on how the level of each metric is calculated we thought it is not possible to reach a level of lower than low-medium for some parts. Especially since for example, our microservice architecture requires a way of communicating with each other which will lead to Coupling. For Lack of Cohesion, we choose a threshold of low-medium, because we think this is achievable.

1.3 Motivation for refactoring chosen and non-chosen classes

We decided to choose to manly refactor classes and methods in the Course microservice, except for one method in the Room microservice, for refactoring. Since in the Authentication microservice, we have only two classes that show a metric level of medium-high but since these are related to *SpringSecurity* and this requires a lot of Coupling, we didn't choose to refactor these classes. The same issue appears in the Room microservice where the authentication part takes place. Additionally, we chose to not refactor the entity classes. Entity classes are known for Lack of Cohesion, but this isn't a bad practice. It's just the way an entity class is structured. Therefore we didn't improve the metrics for them.

Another argument for choosing only classes and methods related to the Course microservice is that this microservice entails the most logic of our system, as well as has the biggest connections to other microservices. When looking at the class-level code smells, one thing that stood out for example was that the `CourseController` had a very high number for Coupling. Further, we chose to refactor the
`EnrollmentController`, which had a Coupling level of medium-high. On method-level code smells, we noticed that many methods related to the `WebClient` communication had high Coupling which is due to the data requests send from one service to another. Further, the method `removeTeacherFromCourse()` in the `EnrollmentController` has a very-high level in Coupling, this probably due to a more code-smells and therefore needs a lot of improvement. Similar to the `WebClient` communication Coupling in many methods, we noticed that we have many methods coupled to the `SecurityContextHolder` and the `UserInfo` class which leads to a high score in the method-level metrics.

2 Part 2: Method-level code smells

2.1 `WebClient`-related refactoring: Couplers: Feature Envy

When we were looking at the Coupling score for some methods inside the `Course Microservice`, we found that many classes had a bad-code smell of "Feature Envy"[1]. All the methods which had to send a `GET` request to a different micro-service where using a call to the `WebClient` which accesses data frequently from other classes. These methods (e.g. `doFilterInternal()` method in `JWTRequestFilter` class for validating requests) have a relatively high Coupling score. The Coupling score ranged from medium-high to very-high. So, we decided to use the "Extract Method Refactoring"-method by creating a `ServiceCommunication` class which has a static method `sendGetRequest()`, which sends a `GET` request based on a few parameters. This allowed us to replace the current way of sending `GET` requests, which required us to include the `WebClient`-related, lengthy dependencies each time we send a web request. This refactoring therefore also reduced the Coupling score of the refactored methods.

The purpose of this type of refactoring is to reduce the length of these methods by outsourcing the similar functionalities of these
methods to a dedicated method using "Extract Method Refactoring" technique. Methods after the refactoring will be more focused on their purposes instead of dwelling on the details. The code is also more maintainable since changing the configurations of the requests now only requires changes in one single method instead of multiple.

For instance, we have extracted the lines that were needed to create the web request with a simple
`ServiceCommunication.sendGetRequest()` method call with the corresponding parameters. We've achieved a reduction in the Coupling by as much as five per each refactor in each method. The same idea applies to other methods that we have refactored.

The list of methods which were refactored using `ServiceCommunication`:
Refer to the figures to compare the metrics from before and after.

- `CourseCreateController, createCourses()`:
  (CBO = 10) (Figures 8 and 9)
- `JWTRequestFilter, doFilterInternal()`:
  (CBO = 5) (Figures 10 and 11)
- `ClearRoomValidator, handle()`:
  (CBO = 2) (Figures 12 and 13)
- `RoomAvailibilityValidator, roomAvailabilityCheck()`:
  (CBO = 3) (Figures 14 and 15)
- `RoomFitValidator, roomFitCheck()`:
  (CBO = 3) (Figures 16 and 17)

[1] https://refactoring.guru/smells/feature-envy

2.2 `UserInfoHelper`-related refactoring: Couplers: Feature Envy

A similar Coupling of code in the repository before the refactoring is that we had to go through the `SecurityContextHolder` and the `UserInfo` to acquire the `NetID` of the user that is interacting with the application, this intoduces a bad-code smell of "Feature Envy"[2]. It involved two extra types of objects thus applying the "Extract Method Refactoring" to the code would reduce the Coupling of these methods and, consequently, reduce the overall Coupling of these classes too. Indubitably, the length of the methods is also reduced.

We decided to create a helper class that handles the extractions of the `NetID` from the `SecurityContextHolder`. By implementing the helper, therefore we reduce the Coupling of any method that uses this helper and thereby achieving an overall reduction of the Coupling in the `(Un)enrollmentController` class and the Course-related controller classes.

Similar to the `WebClient` refactor, `UserInfoHelper` also reduces the Coupling by 3 per each method refactoring and reduced the length of the methods. It increases the focus of these methods and makes a single point of edit possible to alter the behavior of parsing the `NetID` from the incoming web request.

For instance, we've, previously, performed multiple method calls to the `SecurityContextHolder` to acquire a `UserPrinciple` object. Then, we had to cast it to a UserInfo object so that we could make use of the `getNetID` method. All of these are being done by just one simple method call to `UserInfoHelper.getNetID()`.

The list of methods which were refactored using `UserInfoHelper`: Refer to the figures to compare the metrics from before and after.

- `CourseCreateController, createCourse()` (Figures 18 and 19)
- `UnenrollmentController, unenrollStudent()`
- `UnenrollmentController, removeTeacherFromCourse()`
- `UnenrollmentController, removeStudentFromCourse()`
- `EnrollmentController, enrollInCourse()` (Figures 20 and 21)

2.3 `UnenrollmentController, removeTeacherFromCourse()`: Couplers: Innappropriate Intimacy; Dispensables: Speculative Generality

The method removeTeacherFromCourse() had several code smells one of them being "Inappropriate Intimacy" and this was reflected in it having very-high Coupling metric and a score of 12 for Coupling Between Objects (CBO) (seen in Figure 20) and another being "Speculative Generality"[3]. The latter happened when a logic check whose sole purpose was communication to a repository was called only from within this method. To solve this we used the inline method[4] refactoring technique.

[2]https://refactoring.guru/smells/feature-envy
[3]https://refactoring.guru/smells/speculative-generality
[4]https://refactoring.guru/inline-method

Later, to refactor this method we used the technique replace temp with query[5] where instead of first making a `UserInfo` temp and then querying a variable from there, we query the variable straight away. This relates to the "Inappropriate Intimacy" issue and decreases the CBO because we are not accessing the internal fields of the Context Class but instead use a public method from a different class which provides the necessary String, without the code in `removeTeacherFromCourse()` having to know the internal structure.

Lastly, we refactored this method by applying the collapse hierarchy refactoring technique [6] related to the request handlers in the method which refer to the same object as in that case they do not have to be chained and can be removed. Despite the creation of these handlers being loose Coupling instead of tight Coupling, it still dropped the CBO in this method. At the end we have arrived to a CBO score of 4, as seen in Figure 22), which is a big improvement.

2.4 `CourseUpdateController, updateCourse()`: Bloaters: Long Parameter List

The `updateCourse()` featured a "Long Parameter List" code smell [7], requiring five fields to update a course. Four of these parameters were the fields from the course to update (course code, description, etc...), passed along as request parameters. By including these in the request body instead (and then de-serializing the request body to extract the parameters), it was possible to reduce the number of parameters to two. Since this type of refactoring aimed more for making the code easier to understand and open to extensions, it did not improve any metrics.

2.5 `JwtRequestFilter, doFilterInternal()`: Couplers: Feature Envy

Before any refactoring, `doFilterInternal()` method had "very-high" Coupling score and a score 14 for CBO . Due to the "Feature Envy"[8] code smell in this method. It was found that this was due to multiple calls made by codeWebClient and the block of code which sets the Authentication token for the current request. The block of code which creates the Authentication token has been extracted into `authenticateRequest()` method, which takes a HTTP request as input and authenticates the current request. After refactoring by extracting the WebClient and `authenticateRequest()`, the Coupling score has reduced to "low-medium" and a CBO score of 5. Since both Course microservice and Room microservice both have the same implementation of the filter, they have the same score after refactoring as you can see in Figures 10 and 11.

[5] https://refactoring.guru/replace-temp-with-query
[6] https://refactoring.guru/collapse-hierarchy
[7] https://refactoring.guru/smells/long-parameter-list
[8] https://refactoring.guru/smells/feature-envy

3 Part 2: Class-level code smells

3.1 `CourseController`: Bloaters : Large Class

The `CourseController` class before the refactoring was almost a God-like class (Large Class Code Smell)[9]. This caused Coupling and Lack of Cohesion problems and a CBO score of 28 and a LCAM score of 0.733. (See Figure 23).

Consequently, we reduced these problems by applying "Move Method Refactoring". `CourseController` was split up in four different classes and the `CourseController` methods were moved to the new classes respecting the Single Responsibility Principle. Methods with similar dependencies were put together. This resulted in the classes:

- `CourseCreateController` (CBO = 18, LCAM = 0.4)
- `CourseDeleteController` (CBO = 7, LCAM = 0.417)
- `CourseUpdateController` (CBO = 8, LCAM = 0.25)
- `CourseOverviewController` (CBO = 5, LCAM = 4)

This led to a significant improvement. (See Figure 24). The Lack of Cohesion problem completely disappeared as now related logic was together and most method pairs used similar attributes. The Coupling problem became a lot less problematic on a class level, by reducing the CBO score for `CourseCreateController` to 18, for `CourseDeleteController` 7, for `CourseUpdateController` 8, and finally for `CourseOverviewController` 5.

3.2 `EnrollmentController` Bloaters: Large Class; Change Preventers: Shotgun Surgery

In the class `EnrollmentController`, prior to refactoring the Size of the class and Lack of Cohesion were low-medium as can be seen on Figure 20. To decrease these code smells the first step we did was extract a class, `UnenrollmentController`, which took on a separate type of behaviour and follows the "Single Responsibility Principle". Since enrollment and unenrollment go through different logic checks, dividing those functionalities into two classes allowed them to stay concise and clear.

However, this still did not solve the "Lack of Cohesion between Methods" issue as there were small one line methods that led to a higher number of method pairs that do not share any attribute, thus negatively affecting cohesion. This code metric is an indication of the "Shotgun Surgery" code smell[10] which means that if a change has to be done to a part of this class, then many smaller methods within it would have to be searched for and changed. This makes the maintenance of the class more difficult and the organization of the methods within it counter-intuitive. The refactoring that we have done is change these methods, like `teacherAlreadyEnrolled()` and the repeated `UserInfoHelper`.

[9] https://refactoring.guru/smells/large-class
[10] https://blog.ndepend.com/lack-of-cohesion-methods/

By adjusting the methods in the class into a more universally cohesive method `isTeacherOfCourse()` which works for both methods to remove a student and a teacher, we have reduced LCOM of the class to 0.167 and the Lack of Cohesion metric to low as can be seen on Figure 28.

3.3 `CourseUpdateController/CourseUpdateRequestModel`: Bloaters: Data Clump

The endpoint for updating courses requested five different fields, four of which are course parameters (course code, description, max size, enrollment status) that were provided in the URL as request parameters. This can be considered a "Data Clump" code smell[11], as the parameters are passed through the endpoint separately, instead of being properly grouped into a single object.

To remedy this problem, we introduced a new class, `CourseUpdateRequestModel`, which possesses four attributes, corresponding to the course parameters mentioned above. This allows us to change the way these parameters are provided to the endpoint: they can now be included in the request body as JSON, instead of being passed as request parameters. This then allows us to de-serialize the JSON into a `CourseUpdateRequestModel` type.

Overall, this change resulted in more organized code, improving readability and maintainability.

3.4 `NotNullValidator`: Dispensable: Lazy Class

`NotNullValidator` used to be part of the validator chain that performs various checks. Later on, we've realized that these objects, by definition, cannot be `Null`. Adding a preceding `@Valid` annotation makes sure that the `Course` object is instantiated and the `HttpServletRequest` must be instantiated for this API to be triggered. This class can therefore be categorized as a "Lazy Class" bad-code smell and can be removed from the code base. Removing this validator and its usage in the `createCourse` method further reduces the Complexity as well. Refer to the Figure 25 and 26 to compare the effect on RFC by removing one extra check.

3.5 `EnrollmentController`: Couplers: Message Chains, Feature Envy

Prior to refactoring, `EnrollmentController` had a medium-high Coupling which indicated it having the code smells: "Message Chains" and "Feature Envy". While the latter was solved in previous refactoring of extracting a class, message chains smell couldn't be solved in the same way as it was linked closely to the "Chain of Responsibility" chain building and simply extracting that as a class would interfere with the intention of why it was implemented in the first place. Despite the Chain of Responsibility by itself already decreasing tight Coupling with several objects that potentially handle the requests from the controller class, the loose Coupling it had still resulted in a higher metric than our thresh hold (CBO of 12). Initially, the "client" class, being the `EnrollmentController`

[11] https://refactoring.guru/refactoring/smells/bloaters

was constructing the chains on its own and while it is also a good practice, the fact that multiple almost identical chains had to be constructed really emphasized the code smells mentioned earlier. One possibility would be to combine some of the handlers in the chain. We decided against it as it undermines the essential idea that a handler deals with a particular object, and instead would increase the Coupling between objects for the theoretical "total-handler".

Instead we decided to switch to the other possibility of the controller class receiving the already build basic chain from factory classes in order for the correct environment of the chain to be respected[12]. This means that we have used the refactoring techniques "Replace Method with Method Object"[13] in order to construct the basic chain and "Replace Constructor with Factory Method"[14] in order to respect the benefits of Chain of Responsibility and not generate an over complicated constructor. Moreover, the basic chain can be built upon by any method if any other handlers are needed. By only calling to those objects when necessary by the specific method instead of relocating those checks inside of the controller class, it increases the cohesion of methods of the class.

The effect of the refactoring is being supported by the metrics from *CodeMR*. We have achieved low-medium level of Coupling which was medium-high before refactoring. Refer to the Figure 27 and 28

[12]Point 4 under "How to Implement" https://refactoring.guru/design-patterns/chain-of-responsibility

[13]https://refactoring.guru/replace-method-with-method-object

[14]https://refactoring.guru/replace-constructor-with-factory-method

4 Conclusion

Concluding, looking at our new generated metrics in Figures 29, 30 31, 32 33, and 34. We can see that through the detection bad-code smells and there individual refactoring, we achieved our goal. Reducing every possible class to below our previous mentioned threshold. As expected the entity classes and the *SpringSecurity* classes aren't below the threshold for named reasons. All the other classes and methods have now a low level for Lack of Cohesion and at least a low-medium level of Coupling.

Based on these results, we can say that we successfully refactored our code and therefore achieved higher quality of our code which leads to better reusability, understandability, and resistance.

A Figures

Enable	Level	Quality Attribute	Condition
☑	low	Coupling	(CBO <= 5)
☑	low-medium	Coupling	(CBO > 5 AND CBO <= 10)
☑	medium-high	Coupling	(CBO > 10 AND CBO <= 20)
☑	high	Coupling	(CBO > 20 AND CBO <= 30)
☑	very-high	Coupling	(CBO > 30)
☑	low	Complexity	(WMC <= 20) OR (RFC <= 50) OR (DIT <= 1)
☑	low-medium	Complexity	(WMC > 20 AND WMC <= 50) OR (RFC > 50 AND RFC <= 100) OR (DIT > 1 AND DIT <= 3)
☑	medium-high	Complexity	(WMC > 50 AND WMC <= 101) OR (RFC > 100 AND RFC <= 150) OR (DIT > 3 AND DIT <= 10)
☑	high	Complexity	(WMC > 101 AND WMC <= 120) OR (RFC > 150 AND RFC <= 200) OR (DIT > 10 AND DIT <= 20)
☑	very-high	Complexity	(WMC > 120) OR (RFC > 200) OR (DIT > 20)
☑	low	Lack of Cohesion	(LCAM <= 0.6)
☑	low-medium	Lack of Cohesion	(LCAM > 0.6 AND LCAM <= 0.7)
☑	medium-high	Lack of Cohesion	(LCAM > 0.7 AND LCAM <= 0.8)
☑	high	Lack of Cohesion	(LCAM > 0.8 AND LCAM <= 0.9)
☑	very-high	Lack of Cohesion	(LCAM > 0.9)
☑	low	Size	(LOC <= 50) OR (NOM <= 20)
☑	low-medium	Size	(LOC > 50 AND LOC <= 300) OR (NOM > 20 AND NOM <= 30)
☑	medium-high	Size	(LOC > 300 AND LOC <= 900) OR (NOM > 30 AND NOM <= 40)
☑	high	Size	(LOC > 900 AND LOC <= 1500) OR (NOM > 40 AND NOM <= 50)
☑	very-high	Size	(LOC > 1500) OR (NOM > 50)

Figure 1: Calculation of Quality attributes/metrics.

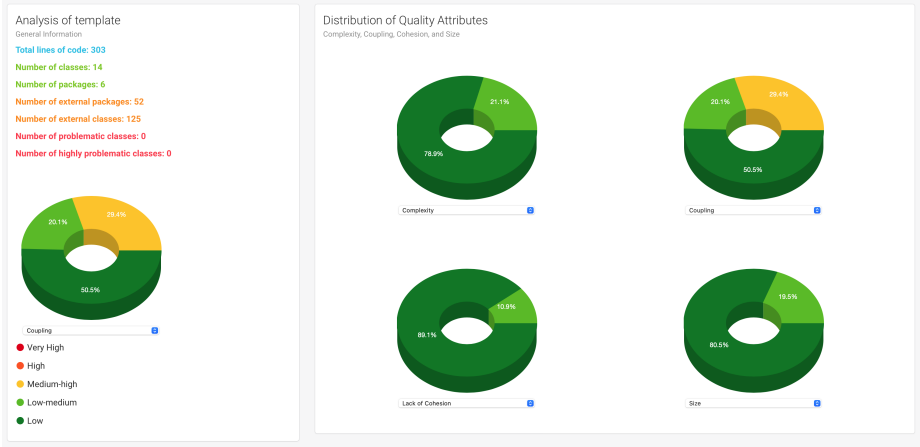


Figure 2: Class-level metric from `authentication-service` package before refactoring.

List of all classes (#14)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	SecurityConfigura...					30	low-medium	medium-high	low	low
2	AuthenticationRes...					59	low	medium-high	low	low-medium
3	JwtRequestFilter					28	low-medium	low-medium	low	low
4	JwtUtil					33	low	low-medium	low-medium	low
5	MethodSecurityConfig					6	low-medium	low	low	low
6	User					39	low	low	low	low
7	UserPrincipal					36	low	low	low	low
8	AuthenticationReq...					15	low	low	low	low
9	UserInfo					14	low	low	low	low
10	UserService					13	low	low	low	low
11	UserRepository					10	low	low	low	low
12	UserController					10	low	low	low	low
13	AuthenticationRes...					6	low	low	low	low
14	AuthenticationSer...					4	low	low	low	low

Figure 3: List of the classes from `authentication-service` package before refactoring.

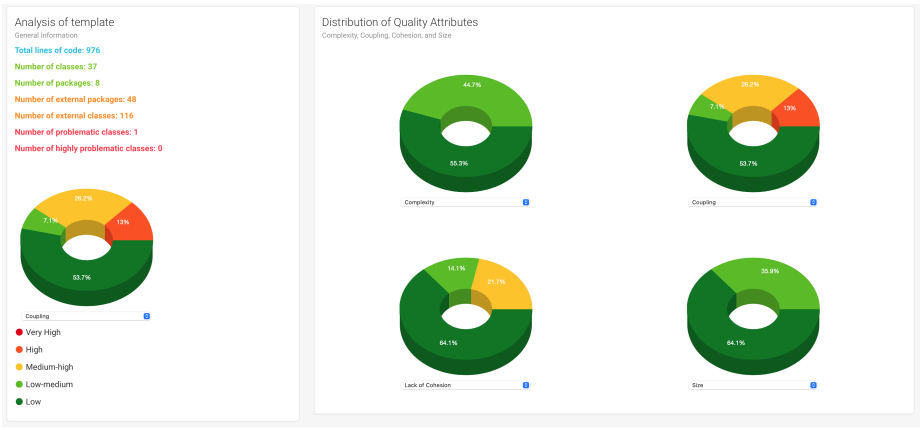
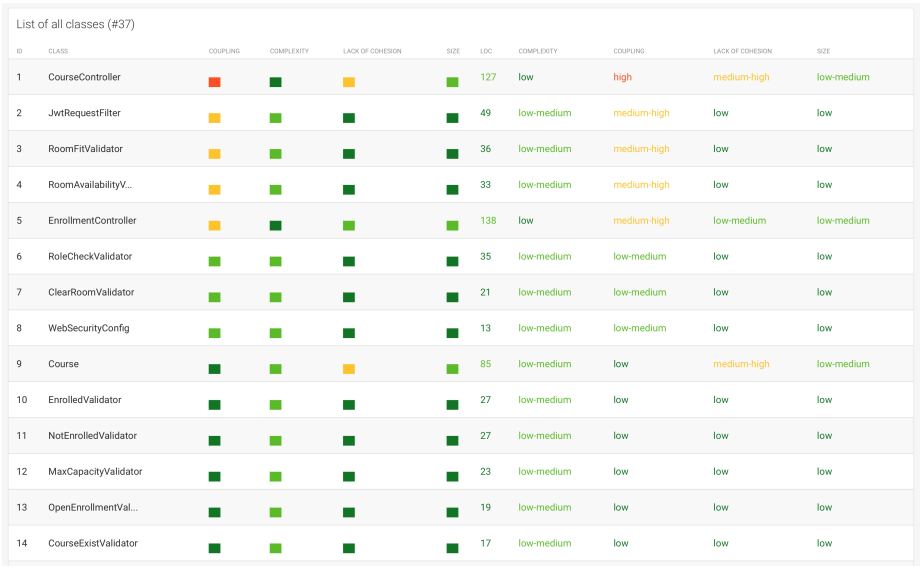


Figure 4: Class-level metric from `course-service` package before refactoring.



List of all classes (#37)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	CourseController					127	low	high	medium-high	low-medium
2	JwtRequestFilter					49	low-medium	medium-high	low	low
3	RoomFitValidator					36	low-medium	medium-high	low	low
4	RoomAvailabilityV...					33	low-medium	medium-high	low	low
5	EnrollmentController					138	low	medium-high	low-medium	low-medium
6	RoleCheckValidator					35	low-medium	low-medium	low	low
7	ClearRoomValidator					21	low-medium	low-medium	low	low
8	WebSecurityConfig					13	low-medium	low-medium	low	low
9	Course					85	low-medium	low	medium-high	low-medium
10	EnrolledValidator					27	low-medium	low	low	low
11	NotEnrolledValidator					27	low-medium	low	low	low
12	MaxCapacityValidator					23	low-medium	low	low	low
13	OpenEnrollmentVal...					19	low-medium	low	low	low
14	CourseExistValidator					17	low-medium	low	low	low

Figure 5: List (top 14) of the classes from `course-service` package before refactoring.

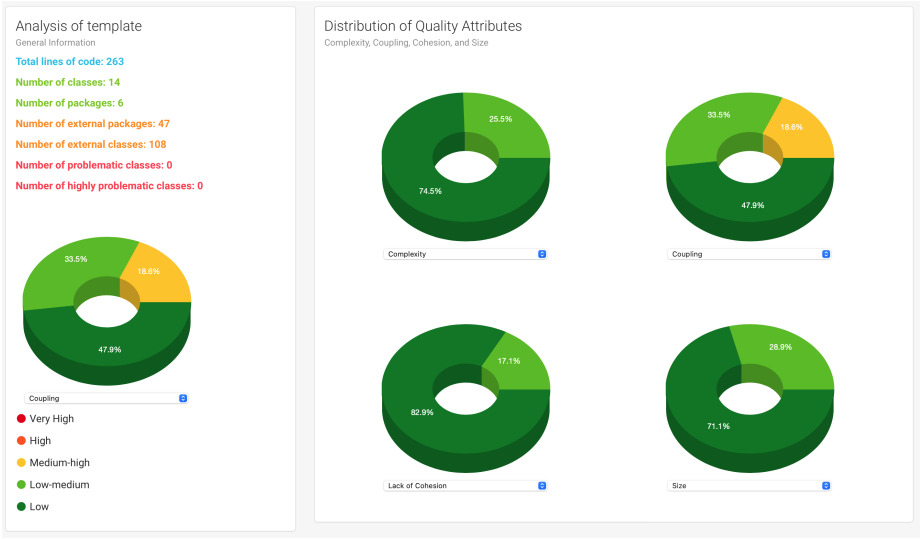
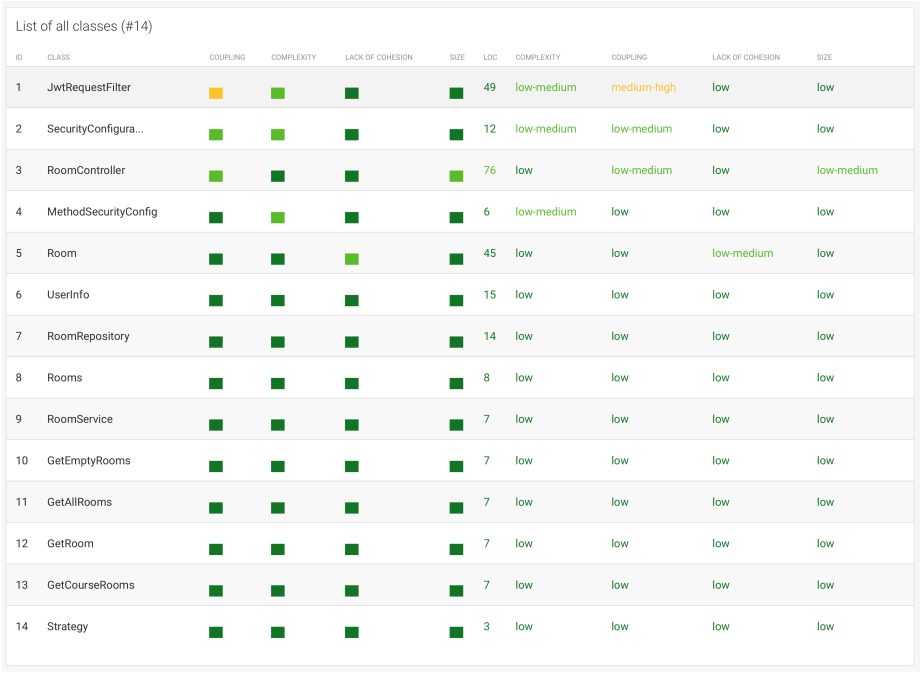


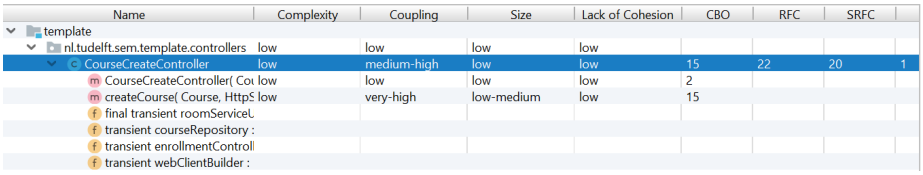
Figure 6: Class-level metric from `room-service` package before refactoring.



List of all classes (#14)

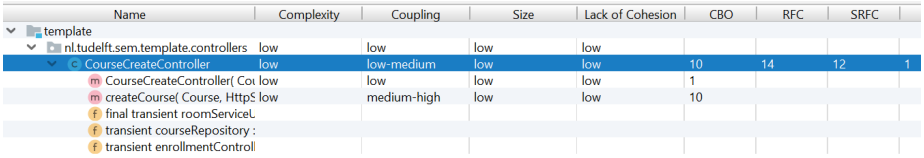
ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	JwtRequestFilter					49	low-medium	medium-high	low	low
2	SecurityConfigura...					12	low-medium	low-medium	low	low
3	RoomController					76	low	low-medium	low	low-medium
4	MethodSecurityConfig					6	low-medium	low	low	low
5	Room					45	low	low	low-medium	low
6	UserInfo					15	low	low	low	low
7	RoomRepository					14	low	low	low	low
8	Rooms					8	low	low	low	low
9	RoomService					7	low	low	low	low
10	GetEmptyRooms					7	low	low	low	low
11	GetAllRooms					7	low	low	low	low
12	GetRoom					7	low	low	low	low
13	GetCourseRooms					7	low	low	low	low
14	Strategy					3	low	low	low	low

Figure 7: List of the classes from `room-service` package before refactoring.



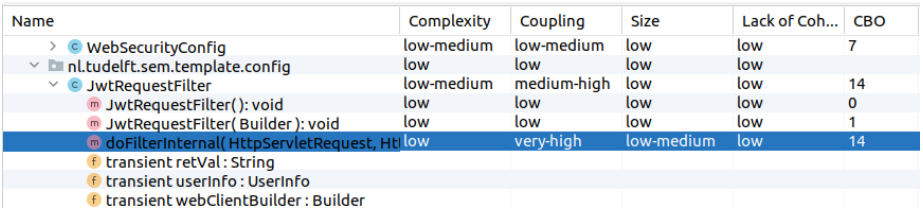
Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	
template								
nl.tudelft.sem.template.controllers	low	low	low	low				
CourseCreateController	low	medium-high	low	low	15	22	20	1
CourseCreateController(Co	low	low	low	low	2			
createCourse(Course, HttpS	low	very-high	low-medium	low	15			
final transient roomServiceL								
transient courseRepository :								
transient enrollmentControll								
transient webClientBuilder :								

Figure 8: Method-level metric from `CourseCreateController` before refactoring.



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	RFC	SRFC	
template								
nl.tudelft.sem.template.controllers	low	low	low	low				
CourseCreateController	low	low-medium	low	low	10	14	12	1
CourseCreateController(Co	low	low	low	low	1			
createCourse(Course, HttpS	low	medium-high	low	low	10			
final transient roomServiceL								
transient courseRepository :								
transient enrollmentControll								

Figure 9: Method-level metric from `CourseCreateController` after refactoring.



Name	Complexity	Coupling	Size	Lack of Coh...	CBO
WebSecurityConfig	low-medium	low-medium	low	low	7
nl.tudelft.sem.template.config	low	low	low	low	
JwtRequestFilter	low-medium	medium-high	low	low	14
JwtRequestFilter( ): void	low	low	low	low	0
JwtRequestFilter(Builder): void	low	low	low	low	1
doFilterInternal(HttpServletRequest, Ht	low	very-high	low-medium	low	14
transient retVal : String					
transient userInfo : UserInfo					
transient webClientBuilder : Builder					

Figure 10: Method-level metric from `JwtRequestFilter` before refactoring.

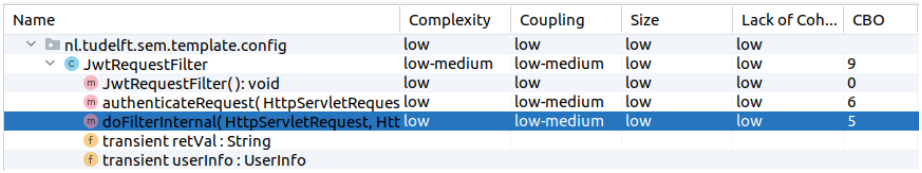


Figure 11: Method-level metric from `JwtRequestFilter` after refactoring.

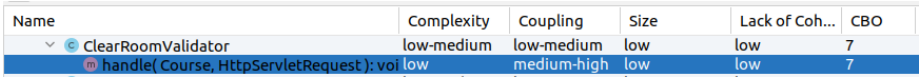


Figure 12: Method-level metric from `ClearRoomValidator` before refactoring.

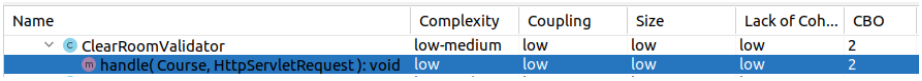


Figure 13: Method-level metric from `ClearRoomValidator` after refactoring.

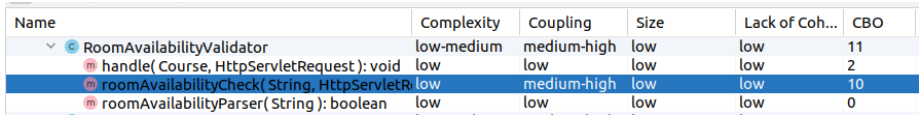


Figure 14: Method-level metric from `RoomAvailabilityValidator` before refactoring.

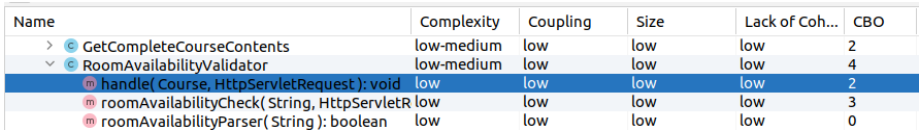


Figure 15: Method-level metric from `RoomAvailabilityValidator` after refactoring.

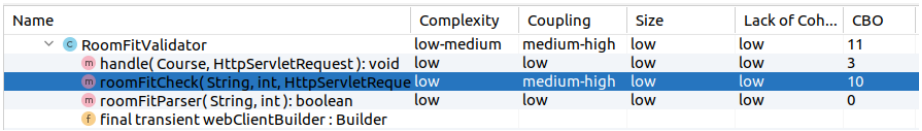


Figure 16: Method-level metric from `RoomFitValidator` before refactoring.

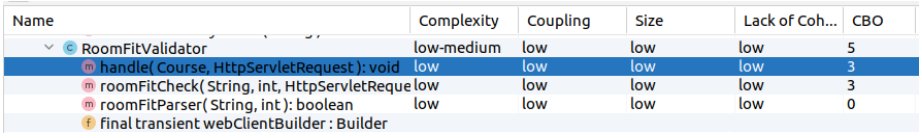
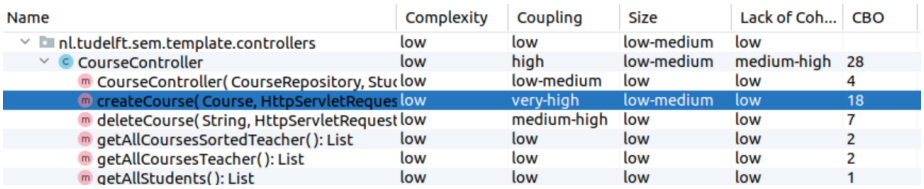
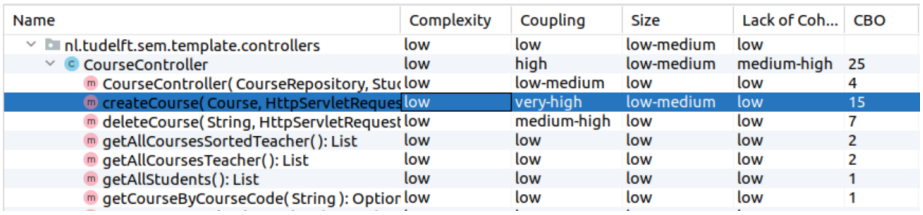


Figure 17: Method-level metric from `RoomFitValidator` after refactoring.



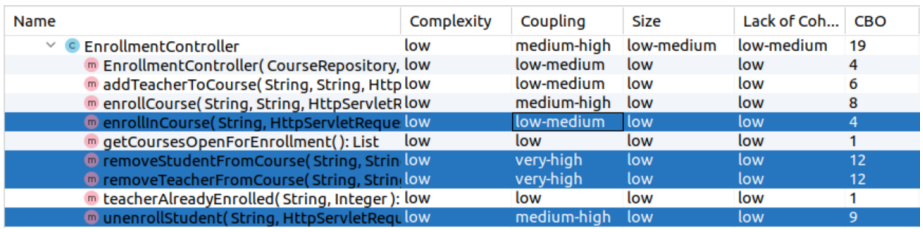
Name	Complexity	Coupling	Size	Lack of Coh...	CBO
nl.tudelft.sem.template.controllers	low	low	low-medium	low	
CourseController	low	high	low-medium	medium-high	28
CourseController(CourseRepository, Stuc	low	low-medium	low	low	4
createCourse(Course, HttpServletReques	low	very-high	low-medium	low	18
deleteCourse(String, HttpServletRequest	low	medium-high	low	low	7
getAllCoursesSortedTeacher( ): List	low	low	low	low	2
getAllCoursesTeacher( ): List	low	low	low	low	2
getAllStudents( ): List	low	low	low	low	1

Figure 18: Method-level metric from `CourseController` before refactoring.



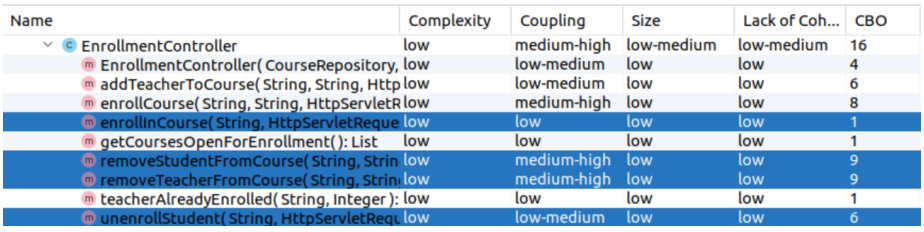
Name	Complexity	Coupling	Size	Lack of Coh...	CBO
nl.tudelft.sem.template.controllers	low	low	low-medium	low	
CourseController	low	high	low-medium	medium-high	25
CourseController(CourseRepository, Stuc	low	low-medium	low	low	4
createCourse(Course, HttpServletReques	low	very-high	low-medium	low	15
deleteCourse(String, HttpServletRequest	low	medium-high	low	low	7
getAllCoursesSortedTeacher( ): List	low	low	low	low	2
getAllCoursesTeacher( ): List	low	low	low	low	2
getAllStudents( ): List	low	low	low	low	1
getCourseByCourseCode(String): Optior	low	low	low	low	1

Figure 19: Method-level metric from `CourseController` after refactoring.



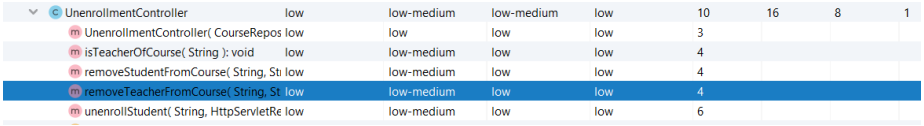
Name	Complexity	Coupling	Size	Lack of Coh...	CBO
EnrollmentController	low	medium-high	low-medium	low-medium	19
EnrollmentController(CourseRepository,	low	low-medium	low	low	4
addTeacherToCourse(String, String, Http	low	low-medium	low	low	6
enrollCourse(String, String, HttpServletR	low	medium-high	low	low	8
enrollInCourse(String, HttpServletReque	low	low-medium	low	low	4
getCoursesOpenForEnrollment( ): List	low	low	low	low	1
removeStudentFromCourse(String, Strin	low	very-high	low	low	12
removeTeacherFromCourse(String, Strin	low	very-high	low	low	12
teacherAlreadyEnrolled(String, Integer):	low	low	low	low	1
unenrollStudent(String, HttpServletReq	low	medium-high	low	low	9

Figure 20: Method-level metric from `EnrollmentController` before refactoring.



Name	Complexity	Coupling	Size	Lack of Coh...	CBO
EnrollmentController	low	medium-high	low-medium	low-medium	16
EnrollmentController(CourseRepository,	low	low-medium	low	low	4
addTeacherToCourse(String, String, Http	low	low-medium	low	low	6
enrollCourse(String, String, HttpServletR	low	medium-high	low	low	8
enrollInCourse(String, HttpServletReque	low	low	low	low	1
getCoursesOpenForEnrollment( ): List	low	low	low	low	1
removeStudentFromCourse(String, Strin	low	medium-high	low	low	9
removeTeacherFromCourse(String, Strin	low	medium-high	low	low	9
teacherAlreadyEnrolled(String, Integer):	low	low	low	low	1
unenrollStudent(String, HttpServletReq	low	low-medium	low	low	6

Figure 21: Method-level metric from `EnrollmentController` after refactoring.



UnenrollmentController	low	low-medium	low-medium	low	10	16	8	1
UnenrollmentController(CourseRepos	low	low	low	low	3			
isTeacherOfCourse(String): void	low	low-medium	low	low	4			
removeStudentFromCourse(String, St	low	low-medium	low	low	4			
removeTeacherFromCourse(String, St	low	low-medium	low	low	4			
unenrollStudent(String, HttpServletRe	low	low-medium	low	low	6			

Figure 22: Method-level metric from `UnenrollmentController` after refactoring.

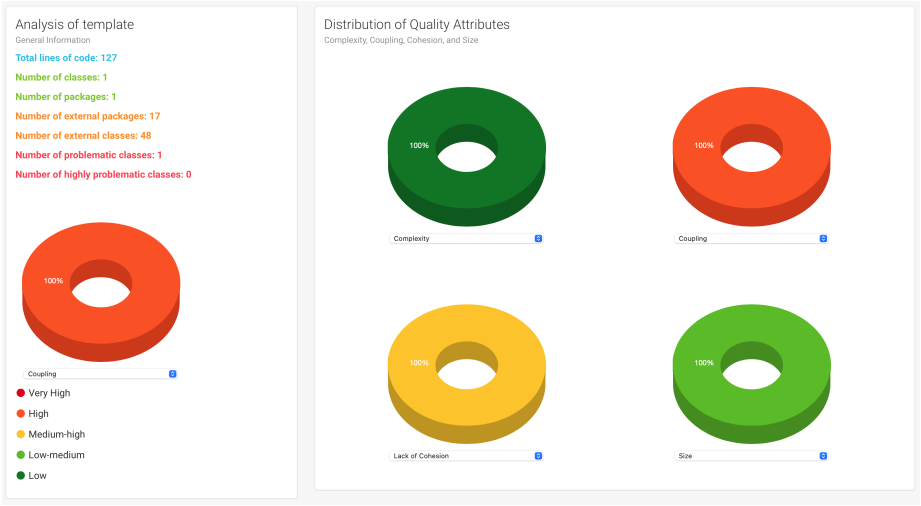


Figure 23: Class-level metric from `CourseController`.

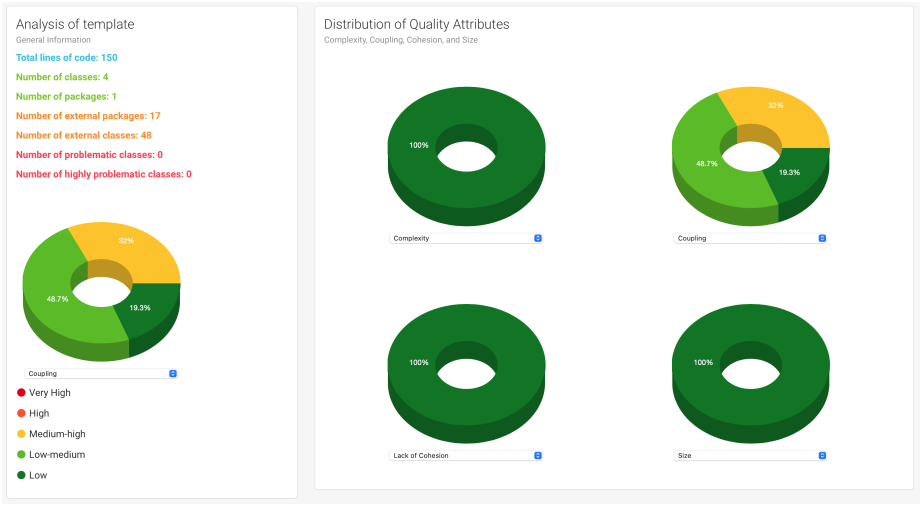


Figure 24: Class-level metric from `CourseCreateController`, `CourseDeleteController`, `CourseUpdateController` and `CourseOverviewController`.

Name	Complexity	Coupling	Size	Lack of Coh...	CBO	RFC	SRFC	DIT
MethodSecurityConfig	low-medium	low	low	low	0	0	0	2
SecurityConfiguration	low-medium	low-medium	low	low	7	10	9	2
nl.tudelft.sem.template.controllers	low	low-medium	low-medium	low				
CourseCreateController	low	medium-high	low	low	15	30	18	1
CourseDeleteController	low	low-medium	low	low	7	9	7	1
CourseOverviewController	low	low-medium	low	low	8	10	4	1
CourseUpdateController	low	low	low	low	5	9	7	1
EnrollmentController	low	low-medium	low	low	8	18	7	1
RoomController	low	low-medium	low-medium	low	9	18	8	1
UnenrollmentController	low	low-medium	low-medium	low	10	16	8	1

Figure 25: Class-level metric (RFC) from `CourseCreateController` before refactoring.

Name	Complexity	Coupling	Size	Lack of Coh...	CBO	RFC	SRFC	DIT
MethodSecurityConfig	low-medium	low	low	low	0	0	0	2
SecurityConfiguration	low-medium	low-medium	low	low	7	10	9	2
nl.tudelft.sem.template.controllers	low	low-medium	low-medium	low				
CourseCreateController	low	medium-high	low	low	15	27	18	1
CourseDeleteController	low	low-medium	low	low	7	9	7	1
CourseOverviewController	low	low-medium	low	low	8	10	4	1
CourseUpdateController	low	low	low	low	5	9	7	1
EnrollmentController	low	medium-high	low-medium	low	12	15	7	1
RoomController	low	low-medium	low-medium	low	9	18	8	1

Figure 26: Class-level metric (RFC) from `CourseCreateController` after refactoring.

Name	Complexity	Coupling	Size	Lack of Coh...	CBO	RFC	SRFC	DIT
nl.tudelft.sem.template.controllers	low	low	low-medium	low				
CourseController	low	high	low-medium	medium-high	28	44	32	1
EnrollmentController	low	medium-high	low-medium	low-medium	19	21	14	1
RoomController	low	low-medium	low-medium	low	9	18	8	1
UserController	low	low	low	low	1	2	1	1
nl.tudelft.sem.template.entities	low	low	low-medium	low				
AuthenticationRequest	low	low	low	low	0	6	0	1
AuthenticationResponse	low	low	low	low	0	2	0	1
Course	low-medium	low	low-medium	medium-high	0	24	4	1
CourseView	low	low	low	low	0	4	0	1

Figure 27: Class-level metric from `EnrollmentConrtoller` before refactoring.

Name	Complexity	Coupling	Size	Lack of Coh...	CBO	RFC	SRFC	DIT
CourseUpdateController	low	low	low	low	5	9	7	1
EnrollmentController	low	low-medium	low	low	8	18	7	1
RoomController	low	low-medium	low-medium	low	9	18	8	1
UnenrollmentController	low	low-medium	low-medium	low	10	16	8	1
UserController	low	low	low	low	1	2	1	1
UserInfoHelper	low	low	low	low	4	7	5	1
nl.tudelft.sem.template.entities	low	low	low-medium	low				
AuthenticationRequest	low	low	low	low	0	6	0	1

Figure 28: Class-level metric from `EnrollmentConrtoller` after refactoring.

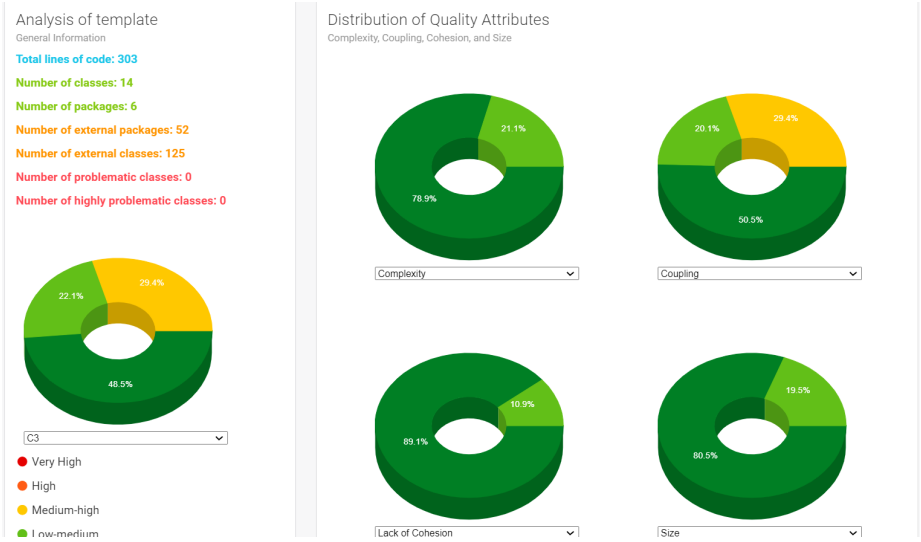


Figure 29: Class-level metric from `authentication-service` package after refactoring.

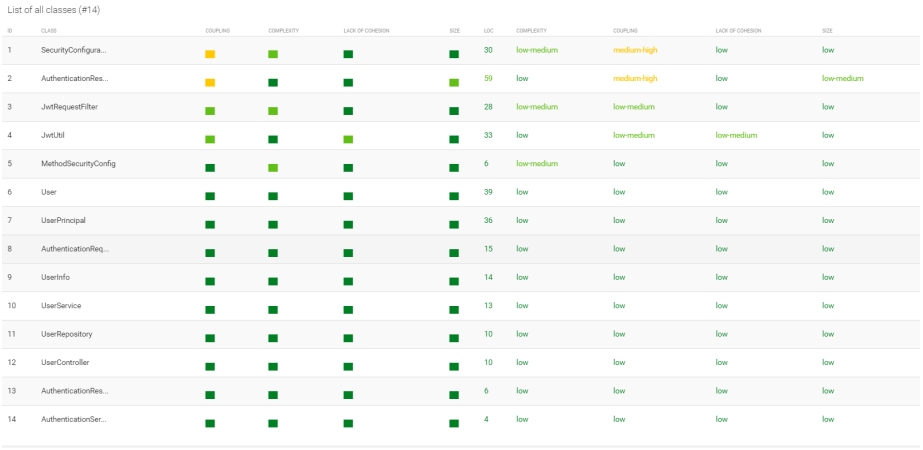
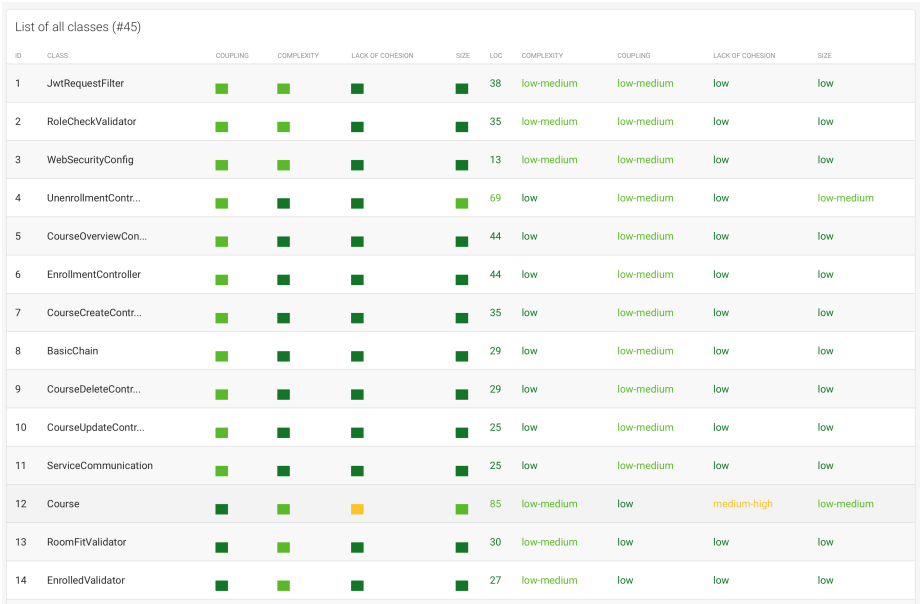


Figure 30: List of the classes from `authentication-service` package after refactoring.

Figure 31: Class-level metric from `course-service` package after refactoring.



List of all classes (#45)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	JwtRequestFilter					38	low-medium	low-medium	low	low
2	RoleCheckValidator					35	low-medium	low-medium	low	low
3	WebSecurityConfig					13	low-medium	low-medium	low	low
4	UnenrollmentContr...					69	low	low-medium	low	low-medium
5	CourseOverviewCon...					44	low	low-medium	low	low
6	EnrollmentController					44	low	low-medium	low	low
7	CourseCreateContr...					35	low	low-medium	low	low
8	BasicChain					29	low	low-medium	low	low
9	CourseDeleteContr...					29	low	low-medium	low	low
10	CourseUpdateContr...					25	low	low-medium	low	low
11	ServiceCommunication					25	low	low-medium	low	low
12	Course					85	low-medium	low	medium-high	low-medium
13	RoomFitValidator					30	low-medium	low	low	low
14	EnrolledValidator					27	low-medium	low	low	low

Figure 32: List of the classes from `course-service` package after refactoring.

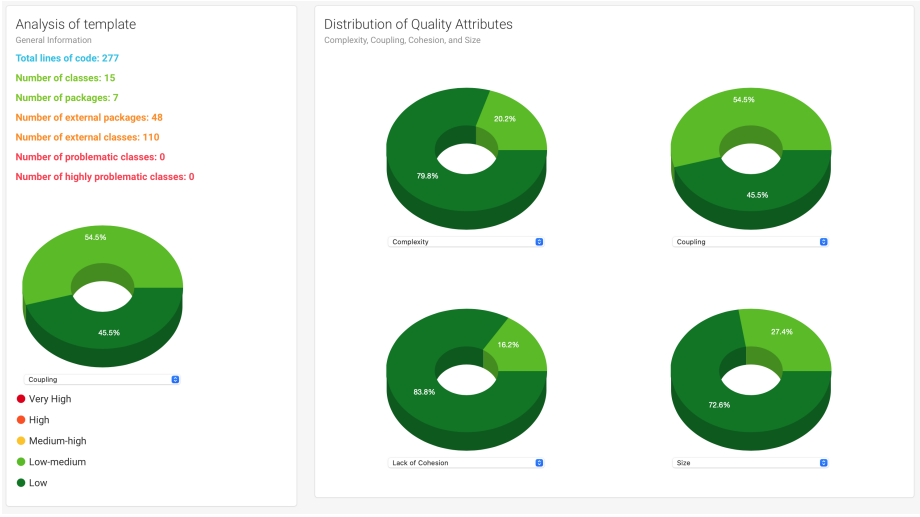


Figure 33: Class-level metric from `room-service` package after refactoring.

List of all classes (#15)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	JwtRequestFilter					38	low-medium	low-medium	low	low
2	SecurityConfigura...					12	low-medium	low-medium	low	low
3	RoomController					76	low	low-medium	low	low-medium
4	ServiceCommunication					25	low	low-medium	low	low
5	MethodSecurityConfig					6	low-medium	low	low	low
6	Room					45	low	low	low-medium	low
7	UserInfo					15	low	low	low	low
8	RoomRepository					14	low	low	low	low
9	Rooms					8	low	low	low	low
10	RoomService					7	low	low	low	low
11	GetEmptyRooms					7	low	low	low	low
12	GetAllRooms					7	low	low	low	low
13	GetRoom					7	low	low	low	low
14	GetCourseRooms					7	low	low	low	low

Figure 34: List of the classes from `room-service` package after refactoring.