Software Engineering Methods

Assignment 2 - Scenario Homeowner's Association

Student name	**Student number**
Rafael Petouris	4776968
Jelt Jongsma	5496594
Vladimir Petkov	5447194
Bram Snelten	5519365
Alex Brown	5171709
Roland Bockholt	5534283

Figure 1: Image of a house model titled Homeowner's Association [1].

CSE2115

Group 19a

Delft University of Technology, The Netherlands

16 January 2023

Contents

1. Identification

Before refactoring our code base, we need to determine what parts of the code required refactoring. To this end, we use the CodeMR IntelliJ plugin to find the most problematic classes. We begin by running the tool on our application. The result is shown in fig 1.1. The output also indicates what classes and methods are problematic. This result is shown in fig 1.4. We created the threshold that all metrics must at least be yellow (medium score in codeMR), but all yellow classes were also inspected for smells and possibly refactored. Our aim is to have only green metrics.

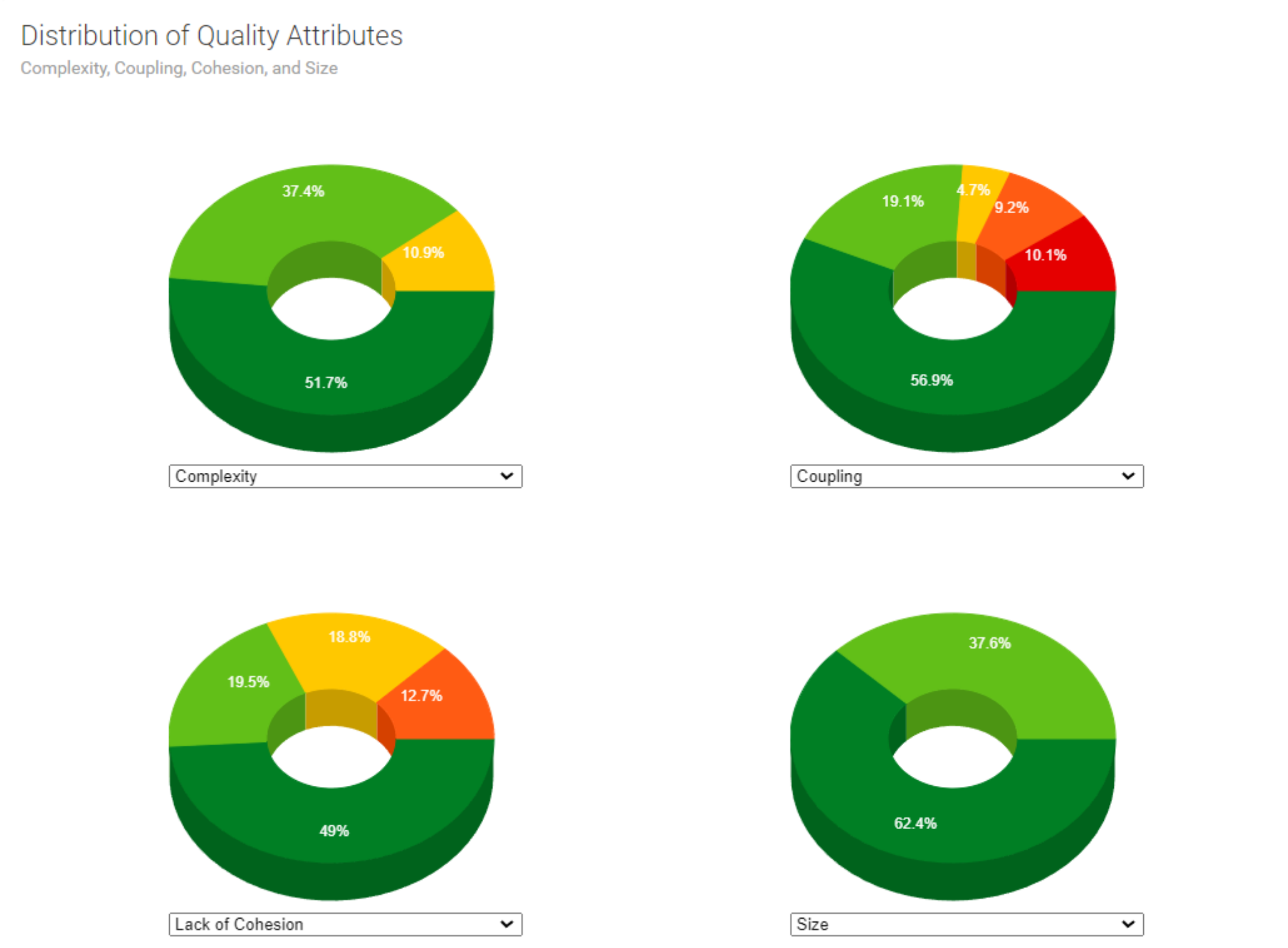
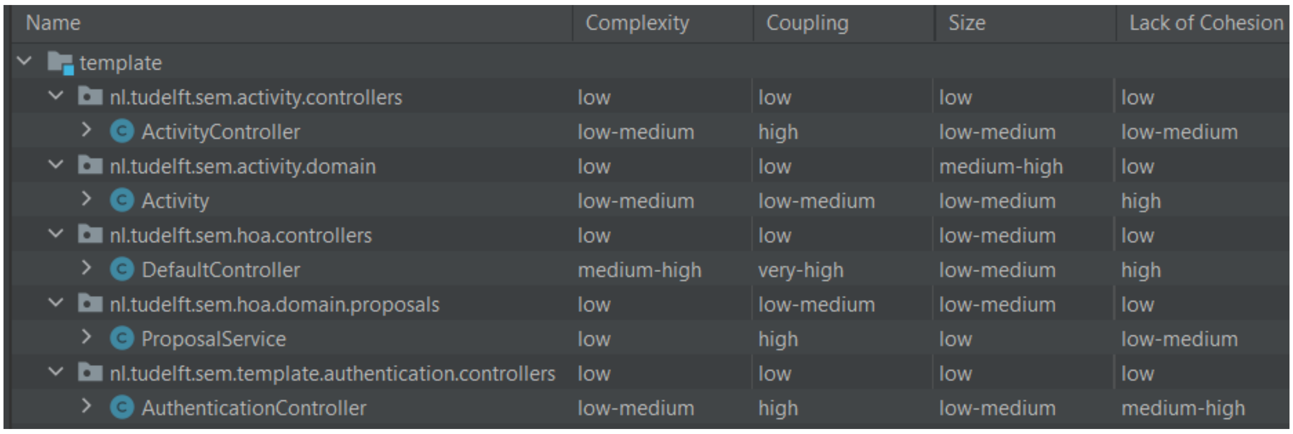


Figure 1.1: Distribution of metrics before



Name	Complexity	Coupling	Size	Lack of Cohesion
template				
nl.tudelft.sem.activity.controllers	low	low	low	low
ActivityController	low-medium	high	low-medium	low-medium
nl.tudelft.sem.activity.domain	low	low	medium-high	low
Activity	low-medium	low-medium	low-medium	high
nl.tudelft.sem.hoa.controllers	low	low	low-medium	low
DefaultController	medium-high	very-high	low-medium	high
nl.tudelft.sem.hoa.domain.proposals	low	low-medium	low-medium	low
ProposalService	low	high	low	low-medium
nl.tudelft.sem.template.authentication.controllers	low	low	low	low
AuthenticationController	low-medium	high	low-medium	medium-high

Figure 1.2: Most problematic classes according to codeMR

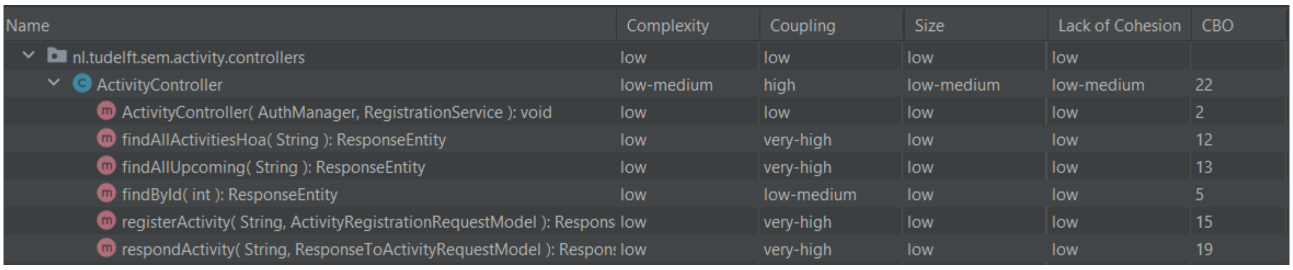
The Metrics we are most interested in are coupling and lack of cohesion since those are the metrics that have the worst scores. The classes that are highly coupled are DefaultController, ElectionService, ActivityController, AuthenticationController, and ProposalService. The classes with a high lack of cohesion are Activity, Hoa, and AuthenticationController.
The classes that will be refactored are:

1. DefaultController
2. ActivityController
3. AuthenticationController
4. ProposalService
5. ElectionService
6. Activity

The reason we have chosen not to refactor HOA over these classes is that most of the non-cohesive methods are getters and setters. Splitting the class or extracting the methods does not make sense since it will only mean that the data belonging to one activity needs to come from two sources or objects. It will add complexity instead of reducing it. In other words, there is no clear smell indicating a need for refactoring.

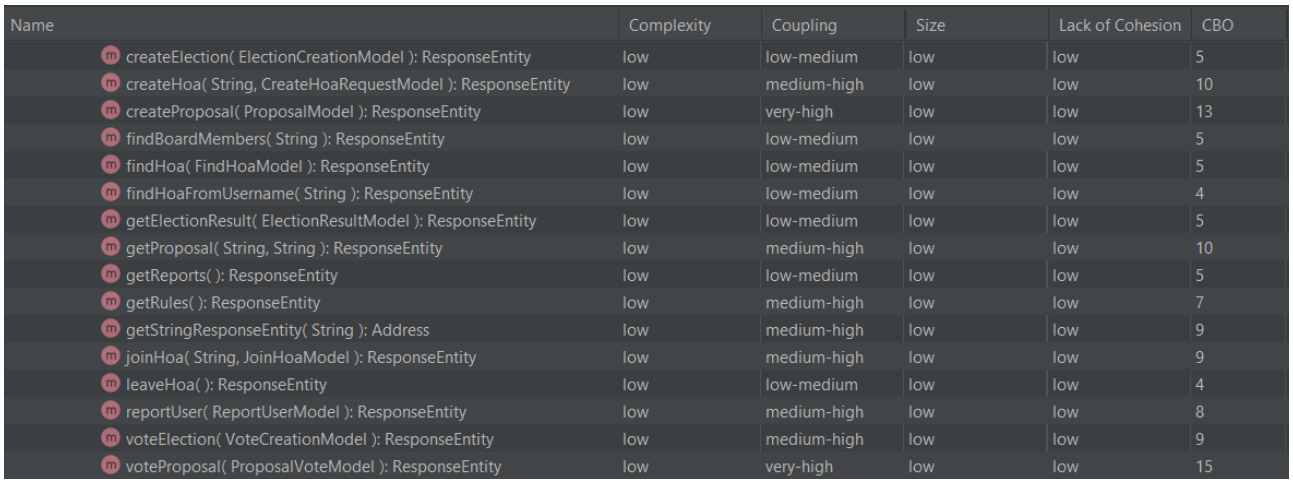
On the method level, we are mostly interested in coupling and size, since LOC is defined on the file or class level. The methods that we have identified to be the worst on these metrics are:

1. Respond Activity - ActivityController
2. Activity constructor - Activity
3. CreateProposalModel constructor - CreateProposalModel
4. Vote Proposal - DefaultController
5. getStringResponseEntity - HoaController



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
nl.tudelft.sem.activity.controllers	low	low	low	low	
ActivityController	low-medium	high	low-medium	low-medium	22
ActivityController(AuthManager, RegistrationService): void	low	low	low	low	2
findAllActivitiesHoa(String): ResponseEntity	low	very-high	low	low	12
findAllUpcoming(String): ResponseEntity	low	very-high	low	low	13
findById(int): ResponseEntity	low	low-medium	low	low	5
registerActivity(String, ActivityRegistrationRequestModel): Respons	low	very-high	low	low	15
respondActivity(String, ResponseToActivityRequestModel): Respon	low	very-high	low	low	19

Figure 1.3: Metrics of the Methods of the ActivityController Class



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO
createElection(ElectionCreationModel): ResponseEntity	low	low-medium	low	low	5
createHoa(String, CreateHoaRequestModel): ResponseEntity	low	medium-high	low	low	10
createProposal(ProposalModel): ResponseEntity	low	very-high	low	low	13
findBoardMembers(String): ResponseEntity	low	low-medium	low	low	5
findHoa(FindHoaModel): ResponseEntity	low	low-medium	low	low	5
findHoaFromUsername(String): ResponseEntity	low	low-medium	low	low	4
getElectionResult(ElectionResultModel): ResponseEntity	low	low-medium	low	low	5
getProposal(String, String): ResponseEntity	low	medium-high	low	low	10
getReports( ): ResponseEntity	low	low-medium	low	low	5
getRules( ): ResponseEntity	low	medium-high	low	low	7
getStringResponseEntity(String): Address	low	medium-high	low	low	9
joinHoa(String, JoinHoaModel): ResponseEntity	low	medium-high	low	low	9
leaveHoa( ): ResponseEntity	low	low-medium	low	low	4
reportUser(ReportUserModel): ResponseEntity	low	medium-high	low	low	8
voteElection(VoteCreationModel): ResponseEntity	low	medium-high	low	low	9
voteProposal(ProposalVoteModel): ResponseEntity	low	very-high	low	low	15

Figure 1.4: Metrics of the Methods of the DefaultController Class

2. Refactoring

2.1 Class Refactoring

2.1.1 ActivityController

To refactor the ActivityController the extract class refactoring was applied. The new method findHoa was extracted to make the class less coupled and more cohesive. A new class HoaConnection took over the responsibility of the HTTP connection with the possibility for future extensions to include more connections to endpoints of the HOA microservice. The findHoa method calls the connectFindHoaByMember method in the new class.

2.1.2 Activity

The Activity class had a high degree of lack of cohesion as its features were predominantly getters and setters and existed in their own logical context, independent from one another. However, since very few of those were valuable for the corresponding microservice's core logic, we decided to remove most of them from the class and refactor the tests to rather rely on the equals method to test similarity than on the getters. That way only the methods corresponding to an activity's id, list of responses, and the getters of the activity's date and HOA id were preserved in addition to the generic overridden methods. Hence, the class was simplified and made more readable.

2.1.3 DefaultController

In order to decrease coupling in this class we decided to use an extract class refactoring. First, we split the controller into 2 distinct controllers: HoaController and DefaultController. HoaController does all the work regarding the creation, finding, and leaving of Hoa's. But Defaultcontroller still has a lot of different functionalities regarding Proposals and Elections. So we extracted another class called ProposalController. It handles the creation and getting of proposals, and also the voting on proposals.

2.1.4 AuthenticationController

The Authentication Controller took care of all requests to the Authentication Microservice. This meant that all authentication, general user, and address methods were in the same class, leading to high coupling. In order to decrease coupling in this class we decided to split the controller into 3 distinct controllers, all with their own tasks. These controllers are now: AuthenticationController, UserController, and AddressController. AuthenticationController takes care of authenticating the user, UserController takes care of registration and password management, and AddressController takes care of address management.

By splitting this class we have significantly lowered its coupling, leading to better code quality.

2.1.5 ProposalService

The proposal service has also a relatively high coupling. This is in part due to the many different Exceptions it throws. To reduce the coupling, a class ProposalServiceException is created. This class contains methods that can throw every Exception related to the ProposalService. Instead of throwing the exception in the ProposalService, a method in the ProposalServiceException is called that then throws the exception. This significantly lowered the coupling of the ProposalService class.

2.1.6 ElectionService

The ElectionService class has high coupling as well. Part of the problem is also the many different exceptions it throws. The throwing of the exceptions is extracted and moved to a class ElectionServiceExceptions. Instead of throwing the exception in the ElectionService, a method in the ElectionServiceException is called that then throws the exception.

2.2 Method Refactoring

2.2.1 Respond Activity

The respondActivity, registerActivity, findAllUpcoming, and findAll methods needed an HTTP connection with the Hoa microservice, and this code was duplicated in various places. To fix this the extract method refactoring was applied. A new method was created, called findHoa, the code was moved to the new function, and in every method that used that code a function call to the new method was added. This made the methods shorter and more readable and allowed for further refactoring.

2.2.2 Create Proposal - Vote Proposal

In DefaultController there were two methods called voteProposal and createProposal. These methods throw a lot of exceptions, leading to high coupling. In order to decrease coupling, extract method refactoring was used. Three new methods were created by the names: throwUnauthorized, throwBadRequest, and throwNotFound. These methods throw exceptions based on an input string and exception.

2.2.3 Create Proposal model

This class has a constructor with five arguments, this is indicative of the too-many-argument-list code smell. This was reduced by overloading the constructor (the original is needed for the JSON deserialization) and passing the proposal object instead of all its properties. This reduces the number of arguments and the technical debt of the code.

2.2.4 Activity

Due to the extended logic inside of the constructor of the Activity class itself that was responsible for marking the organizer's response as going automatically. This logic was extracted to another method setOrganizerResponseToGoing() that is now called in the RegistrationService of the ActivityController right after the construction of one. Thus, the constructor is now not responsible for any functional logic.

2.2.5 getStringResponseEntity

A section of this method was extracted into a new method with the aim of reducing its size. The logic responsible for communicating with other microservices was moved into the new connectToAuthentication method as part of the extract method refactoring.

3. Conclusion

The codeMR report after refactoring is shown in 3.3 and no problematic classes are left after refactoring. When comparing these results to the report created before refactoring, it is clear that the refactoring had an impact on the code metrics of our code. Mostly the coupling and Lack of cohesion have been reduced, which makes sense since those were the ones we focused on. Almost all classes have become green with just a few yellow classes where we did not identify any code smells and the high metrics are inherent to the logic in those classes. The methods in which we wanted to reduce the coupling were refactored and the coupling was reduced in each of them. However not all methods could be refactored to be below the threshold of coupling (CBO) in order to be considered not highly coupled. Now our technical debt has been significantly reduced and maintaining or adjusting the application will become a lot easier.

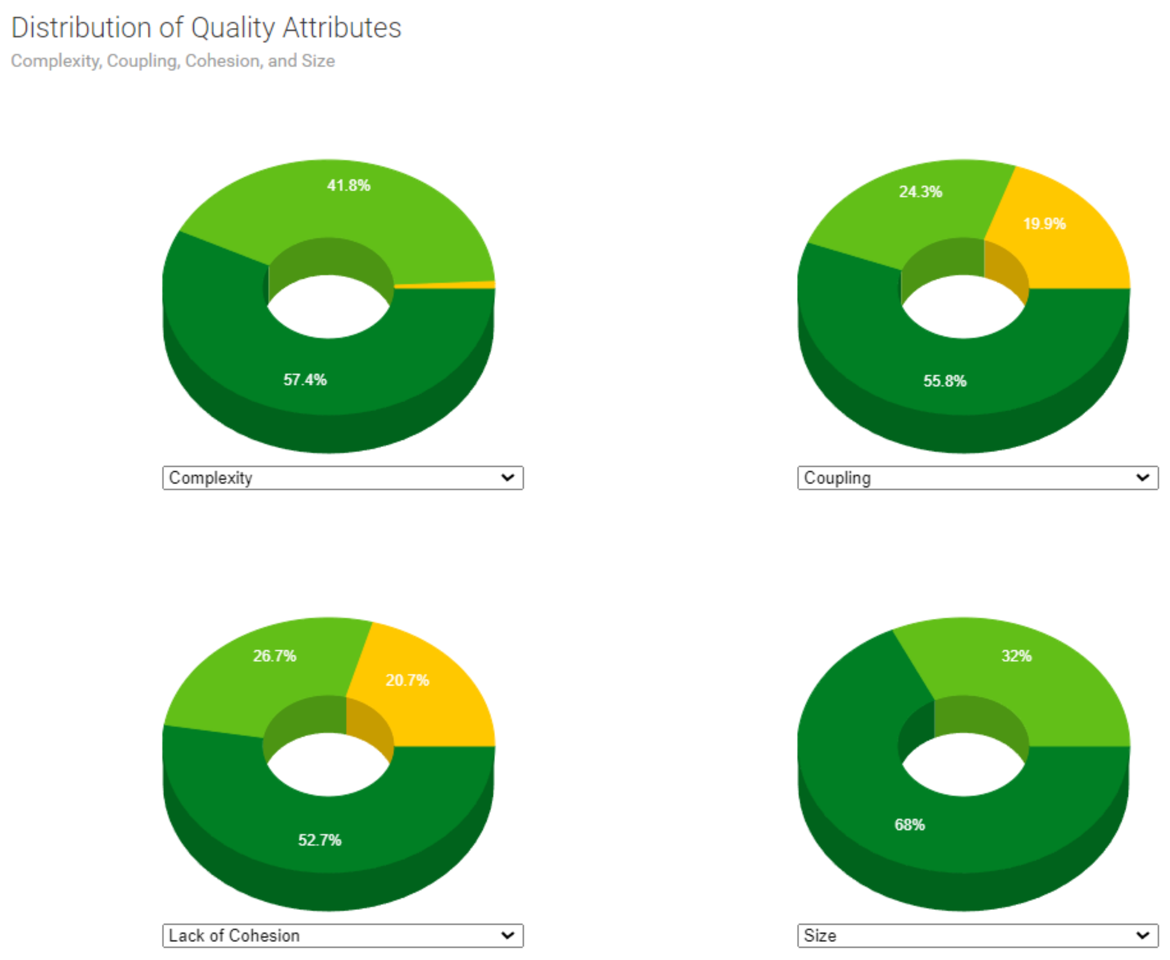
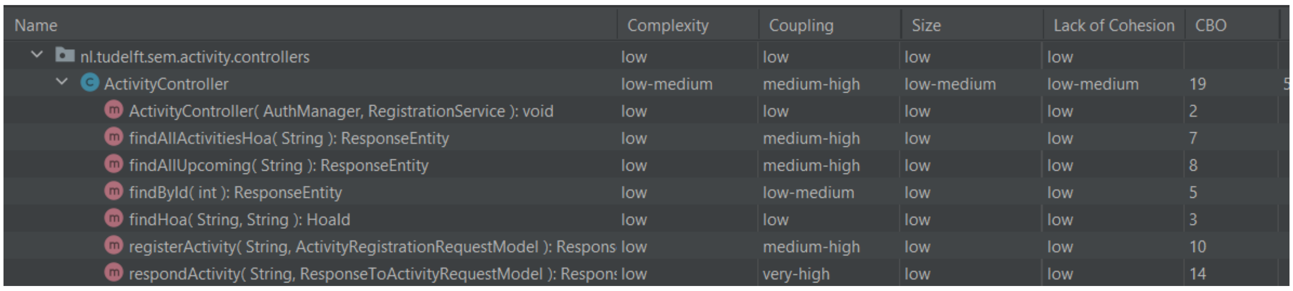
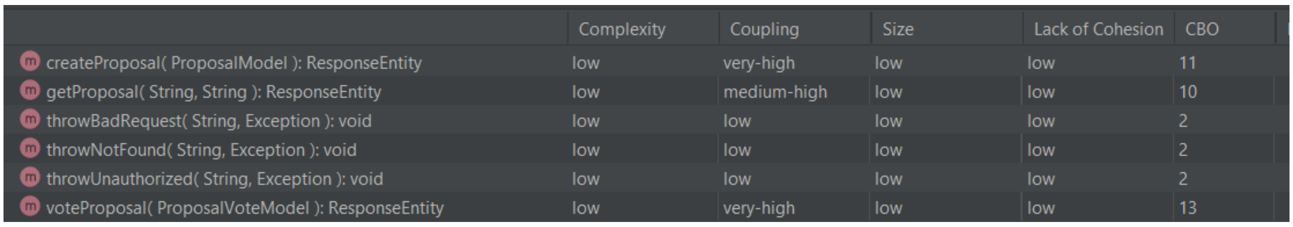


Figure 3.1: Distribution of metrics after refactoring



Name	Complexity	Coupling	Size	Lack of Cohesion	CBO	
nl.tudelft.sem.activity.controllers	low	low	low	low		
ActivityController	low-medium	medium-high	low-medium	low-medium	19	5
ActivityController(AuthManager, RegistrationService): void	low	low	low	low	2	
findAllActivitiesHoa(String): ResponseEntity	low	medium-high	low	low	7	
findAllUpcoming(String): ResponseEntity	low	medium-high	low	low	8	
findById(int): ResponseEntity	low	low-medium	low	low	5	
findHoa(String, String): HoaId	low	low	low	low	3	
registerActivity(String, ActivityRegistrationRequestModel): Respons	low	medium-high	low	low	10	
respondActivity(String, ResponseToActivityRequestModel): Respon	low	very-high	low	low	14	

Figure 3.2: Metrics of Activity controller methods after refactoring



	Complexity	Coupling	Size	Lack of Cohesion	CBO
createProposal(ProposalModel): ResponseEntity	low	very-high	low	low	11
getProposal(String, String): ResponseEntity	low	medium-high	low	low	10
throwBadRequest(String, Exception): void	low	low	low	low	2
throwNotFound(String, Exception): void	low	low	low	low	2
throwUnauthorized(String, Exception): void	low	low	low	low	2
voteProposal(ProposalVoteModel): ResponseEntity	low	very-high	low	low	13

Figure 3.3: Metrics of methods in the newly created Proposal controller after refactoring

Bibliography

[1] Nick Youngson. *Homeowners association*. Retrieved: 27 November 2022. URL: `https://pix4free.org/assets/library/2021-10-24/originals/homeowners-association.jpg`.