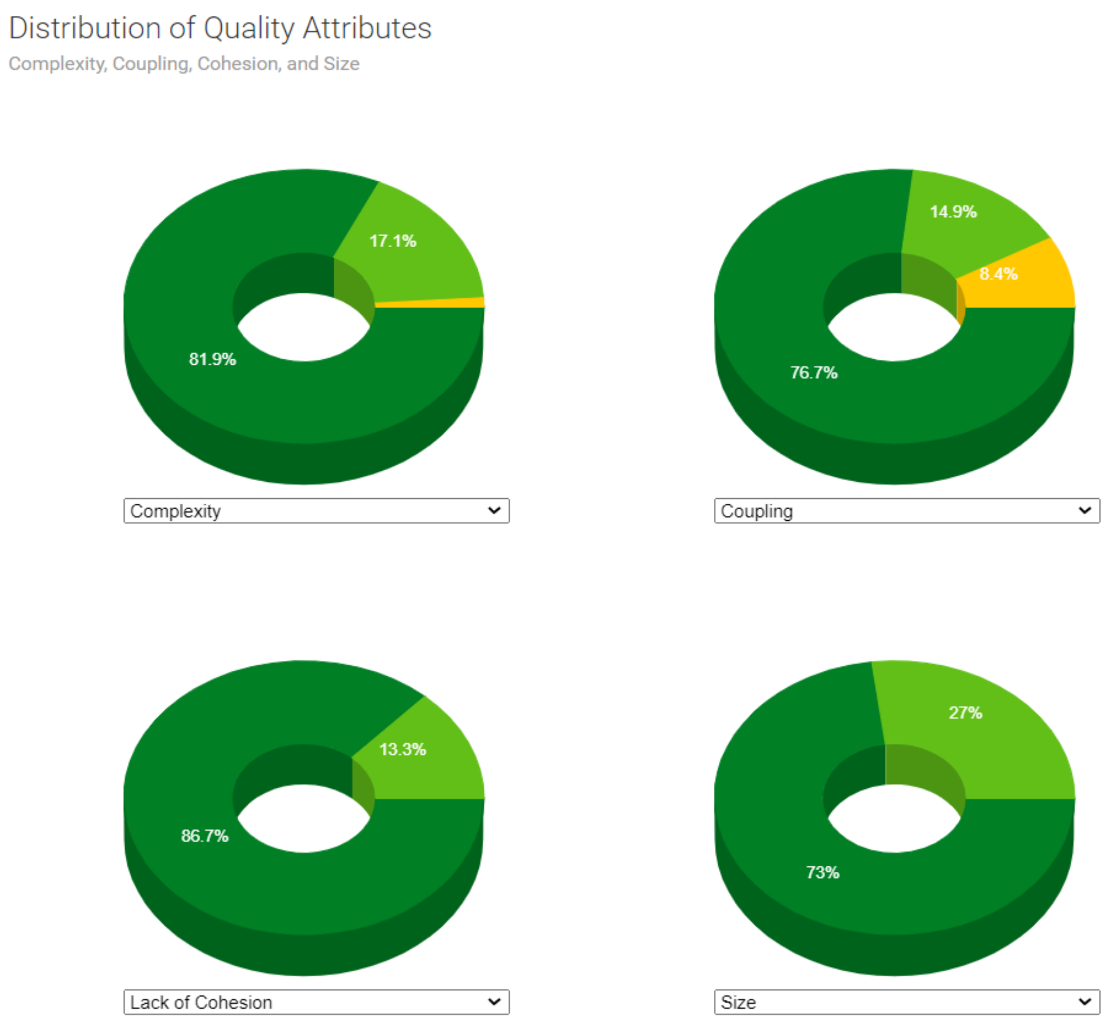
Group 26b

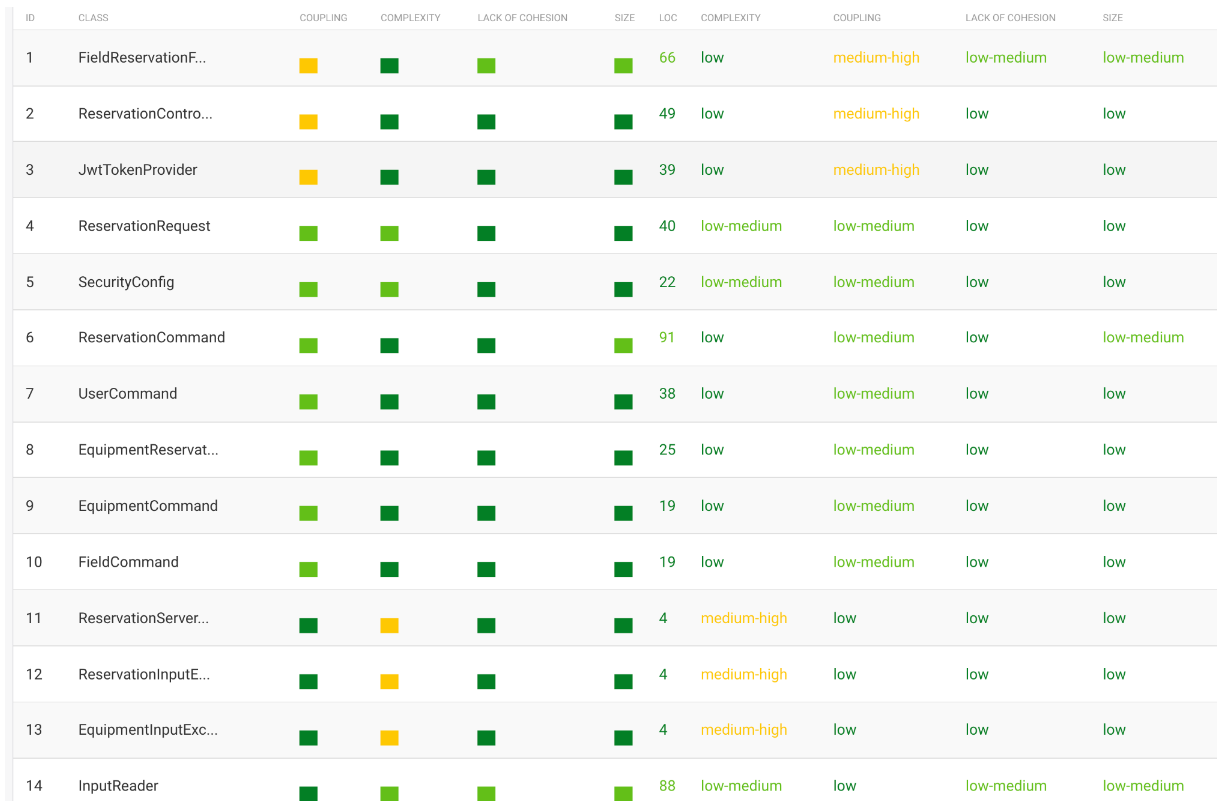
Assignment 2

1. INTRODUCTION

In order to find the methods and classes that need refactoring, we used CodeMR. We then first computed the general metrics in order to decide on the classes and methods that need refactoring the most.



After analyzing the general distribution of quality attributes shown here, we considered Complexity, Coupling, Lack of Cohesion and Size as our main metrics and decided to improve the classes in yellow and light green, since those were the most problematic, as discovered in the code analysis performed by codeMR.



ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	FieldReservationF...	🟨	🟩	🟩	🟩	66	low	medium-high	low-medium	low-medium
2	ReservationContro...	🟨	🟩	🟩	🟩	49	low	medium-high	low	low
3	JwtTokenProvider	🟨	🟩	🟩	🟩	39	low	medium-high	low	low
4	ReservationRequest	🟩	🟩	🟩	🟩	40	low-medium	low-medium	low	low
5	SecurityConfig	🟩	🟩	🟩	🟩	22	low-medium	low-medium	low	low
6	ReservationCommand	🟩	🟩	🟩	🟩	91	low	low-medium	low	low-medium
7	UserCommand	🟩	🟩	🟩	🟩	38	low	low-medium	low	low
8	EquipmentReservat...	🟩	🟩	🟩	🟩	25	low	low-medium	low	low
9	EquipmentCommand	🟩	🟩	🟩	🟩	19	low	low-medium	low	low
10	FieldCommand	🟩	🟩	🟩	🟩	19	low	low-medium	low	low
11	ReservationServer...	🟩	🟨	🟩	🟩	4	medium-high	low	low	low
12	ReservationInputE...	🟩	🟨	🟩	🟩	4	medium-high	low	low	low
13	EquipmentInputExc...	🟩	🟨	🟩	🟩	4	medium-high	low	low	low
14	InputReader	🟩	🟩	🟩	🟩	88	low-medium	low	low-medium	low-medium

The image above exhibits the metrics list of the classes which need refactoring the most. Out of these classes we have chosen the FieldReservationFactory, ReservationController, PromptColor, ReservationService and User entity to improve. Unfortunately a lot of the classes containing yellow refactoring options were based on super classes which we could not change. Therefore not every class was an option for refactoring.

2. CLASS REFACTORING

- FieldReservationFactory

  Problem: Medium-high coupling
  How we improved it: This class had medium-high coupling, which meant it was dependent on too many other classes. We decided to extract the createReservation method logic to a new class called Mapper, since the EquipmentReservationFactory also had a method createReservation with very similar logic. After the refactoring the class went from medium-high coupling to low-medium coupling with the CBO dropping from 11 to 8.

- User entity

  Problem: Size
  How we improved it: This class had low-medium size, which meant that judging by the lines of code, the class was too big and hard to manage. We refactored this class by auto generating the getters and setters using lombok and deleting all the unnecessary getters and setters. After the refactoring, the class went from low-medium size to low size.

- Factories and ReservationService

  Problem: Lack of cohesion and size
  How we improved it: These classes were fixed before the assignment and can be found in commit d304da09. These classes used to have their communication with other microservices coded in between the business logic. We decided to increase manageability by extracting the communication logic and methods into a new class ServiceCommunication. After refactoring the lines of code of the field, the reservation factory decreased from 88 to 71 lines and the lack of cohesion went from medium-high to low-medium.

- ReservationService

  Problem: Lack of cohesion and size
  How we improved it: This class was fixed before the assignment and can be found in commits e957c5c4 (first fix) and 33d1a91d (complete fix). The reservation service class used to have all the methods for validating and creating equipment / fieldreservations. This meant that the class was not manageable or maintainable. We decided to improve these aspects by implementing the abstract factory design pattern. This resulted in the general business logic still being in the reservation service but the specific logic is housed in a factory. After the refactoring the lack of cohesion went from medium-high to low-medium, complexity and coupling went from low-medium to low and lines of code went from 94 to 35

- CLI command classes

  Problem: Class lines of code and size
  How we improved it: There were a few duplicate lines in the majority of the command folder in the CLI microservice. In order to tackle this problem we made a super class containing the duplicate lines and made the rest of those classes inherit from the newly made super class. This solution not only reduces the lines of code in the class as well as the size per method in the class.

3. METHOD REFACTORING

- ReservationController

  Problem: Medium-high coupling
  How we improved it: In this class there were 2 methods which had duplicate code (makeFieldreservation and makeEquipmentReservation). We removed the duplicate code by making an extra method makeReservation which reduced the number of lines of the code 49 to 46, but LOC doesn't say much about the refactoring. We then looked at the effective lines of code which is reduced from 15 to 13 after refactoring. This also reduced the number of calls to external methods and classes which reduced the coupling. The class still has medium-high coupling but there is now nothing more to improve in the class, everything that is there is essential for the working of our program

- FieldReservationFactory -> validateTeam

  Problem: Long parameter list
  How we improved it: This method had five parameters in its list which made the method too complicated. We therefore removed one of the parameters which was not used in the method. After this refactoring, the method had only four parameters in its list.

- ReservationService -> save

  Problem: Long parameter list
  How we improved it: This method was fixed before the assignment and can be found in commits 776568a2 (first fix) and ecd1685c (complete fix). The save method used to have a parameter list with the 11 parameters for an equipment or field reservation. During the development process we already decided that this was too long. So instead of having a request parameter for each parameter of a type of reservation we decided to use a single request body and then map that to an object. After the refactoring the method went from 11 to only 2 parameters in its list.

- FieldReservationFactory -> validate

  Problem: Number of lines and complexity
  How we improved it: This method was fixed before the assignment and can be found in commits 7761fb58 and 33d1a91d. The validate method used to be a blob method with a lot of lines and if-statements for all of the edge cases of validating a reservation. We extracted logic into separate private methods to increase manageability and maintainability. After refactoring the complexity and size went from low-medium to low.

- Reservation Controller Methods

  Problem: Lack of cohesion
  How we improved it: Prior to the changes made in commit 33d1a91d, there existed an abundance of controller methods that returned null values. We solved this code smell by returning a ResponseEntity with an error message. This allows those methods to be tested easier and furthermore increase cohesion within the class.

4. METRICS AFTER REFACTORING

Since most of our refactoring was done before the start of the assignment, the biggest improvements and metric changes cannot be seen in the most recent metrics computed. Only one major change can be observed in the coupling of the FieldReservationfactory, which went from medium-high to low-medium coupling. However, we improved the remaining classes that needed refactoring the most and mentioned the other refactoring operations done before the start of the assignment.