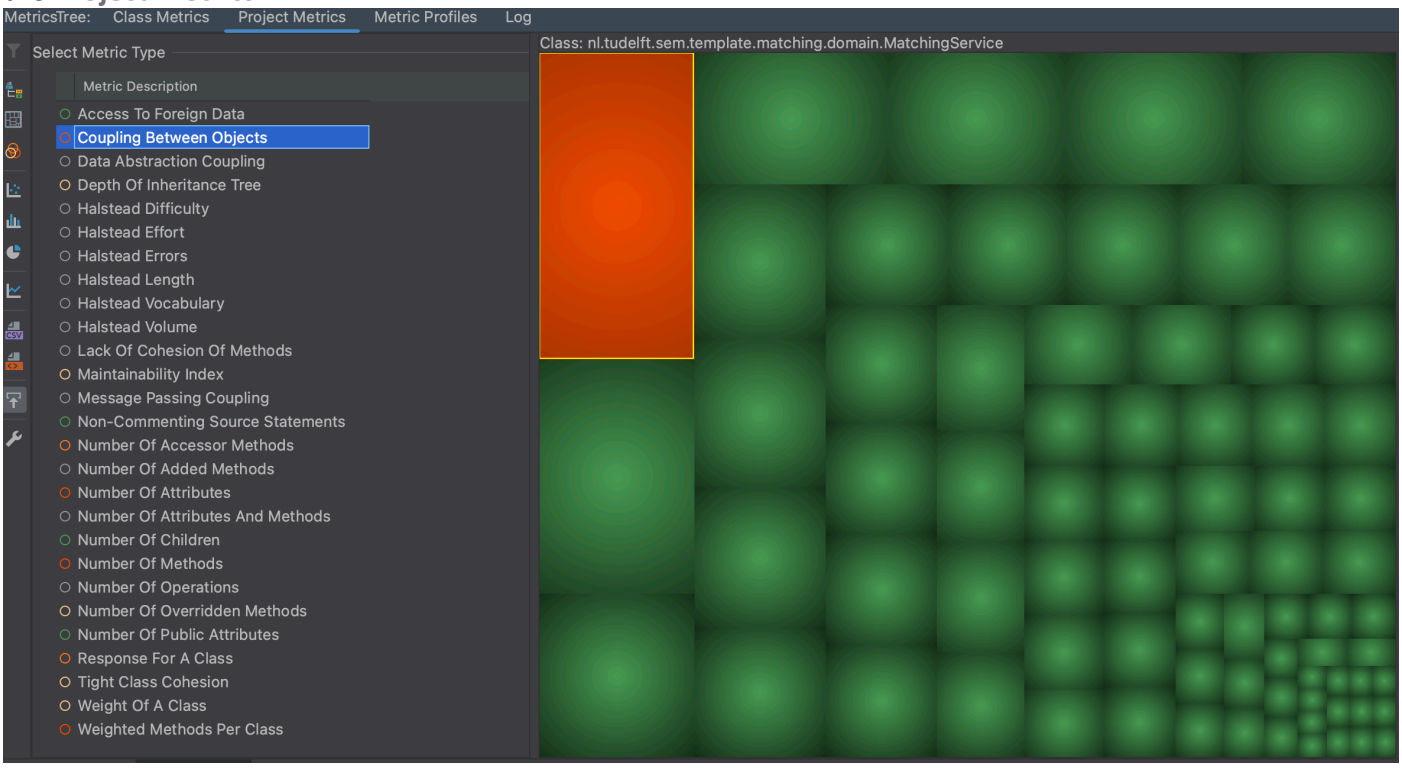
Assignment 2 – Refactoring

Task 1

For computing the code metrics of our project, we decided to use the ***"MetricsTree" IntelliJ plugin*** to identify the important points of refactoring at the class as well as method levels.

We first started by trying to capture an overall image of the problems our software had, thus we looked at the **Project Metrics**

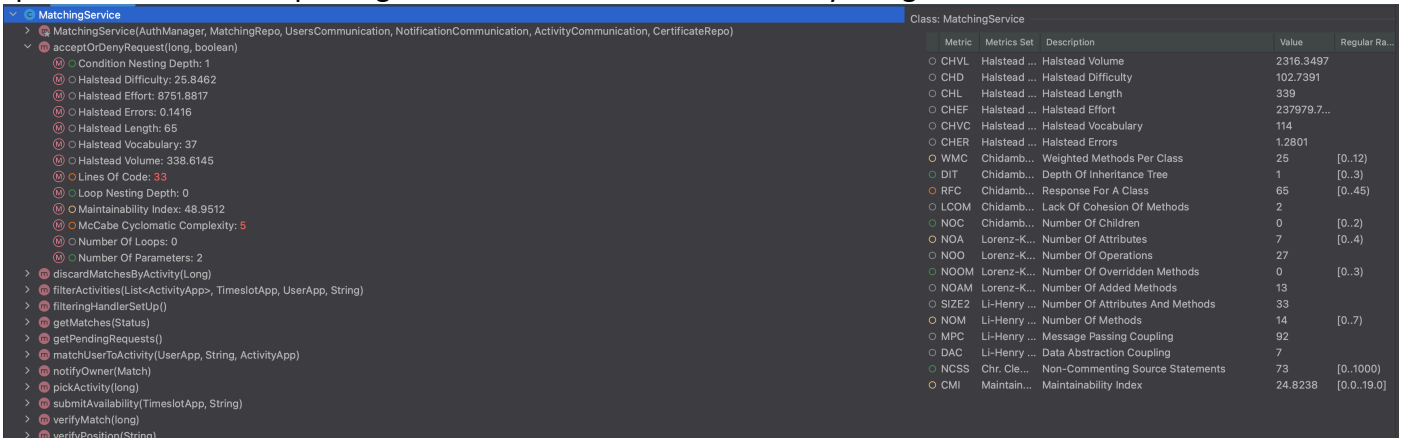


As seen above, the ***"MetricsTree" plugin*** associates each metric to a specific colour marking the gravity of the problem:

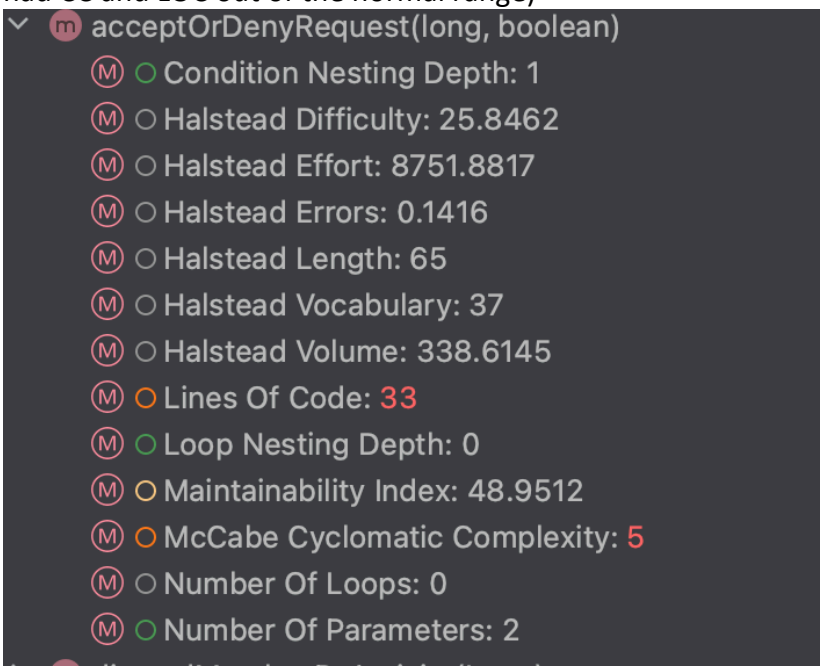
- ➔ **Green** – in the expected range
- ➔ **Yellow** – high above the expected range
- ➔ **Orange** – very high
- ➔ **Red** – extreme values

After considerations, we decided to prioritize refactoring for the classes yielding **red** and then **orange** metrics and leave out the **yellow** and **green** ones since they were in the accepted threshold.

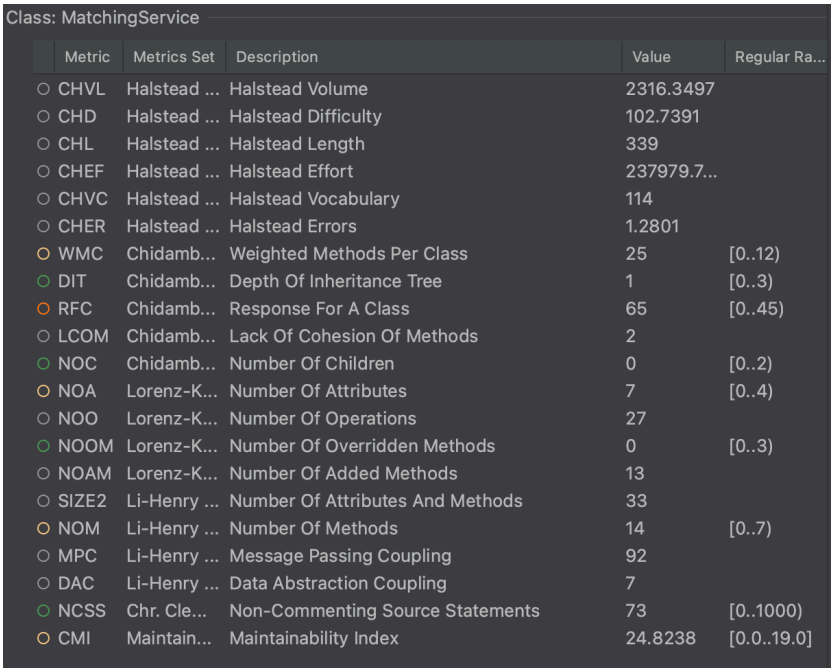
The next step was looking over all the metrics highlighted in **red** and **orange** and carefully analysing the class specified above for inspecting the methods and other metrics by using the **Class Metrics** window



In the left part we went through all the methods of the class and looked for metrics highlighted in red so that we could identify the **method smells** (for example the method **acceptOrDenyRequest** in **MatchingService** had CC and LOC out of the normal range)



In the right part we also kept track of the **class smells** highlighted by the plugin and concentrated strictly on the class under discussion.



Metric	Metrics Set	Description	Value	Regular Ra...
CHVL	Halstead ...	Halstead Volume	2316.3497	
CHD	Halstead ...	Halstead Difficulty	102.7391	
CHL	Halstead ...	Halstead Length	339	
CHEF	Halstead ...	Halstead Effort	237979.7...	
CHVC	Halstead ...	Halstead Vocabulary	114	
CHER	Halstead ...	Halstead Errors	1.2801	
WMC	Chidamb...	Weighted Methods Per Class	25	[0..12)
DIT	Chidamb...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamb...	Response For A Class	65	[0..45)
LCOM	Chidamb...	Lack Of Cohesion Of Methods	2	
NOC	Chidamb...	Number Of Children	0	[0..2)
NOA	Lorenz-K...	Number Of Attributes	7	[0..4)
NOO	Lorenz-K...	Number Of Operations	27	
NOOM	Lorenz-K...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-K...	Number Of Added Methods	13	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	33	
NOM	Li-Henry ...	Number Of Methods	14	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	92	
DAC	Li-Henry ...	Data Abstraction Coupling	7	
NCSS	Chr. Cle...	Non-Commenting Source Statements	73	[0..1000)
CMI	Maintain...	Maintainability Index	24.8238	[0.0..19.0]

In short, the process of identifying the main problems of our software can be summoned up in the following steps:

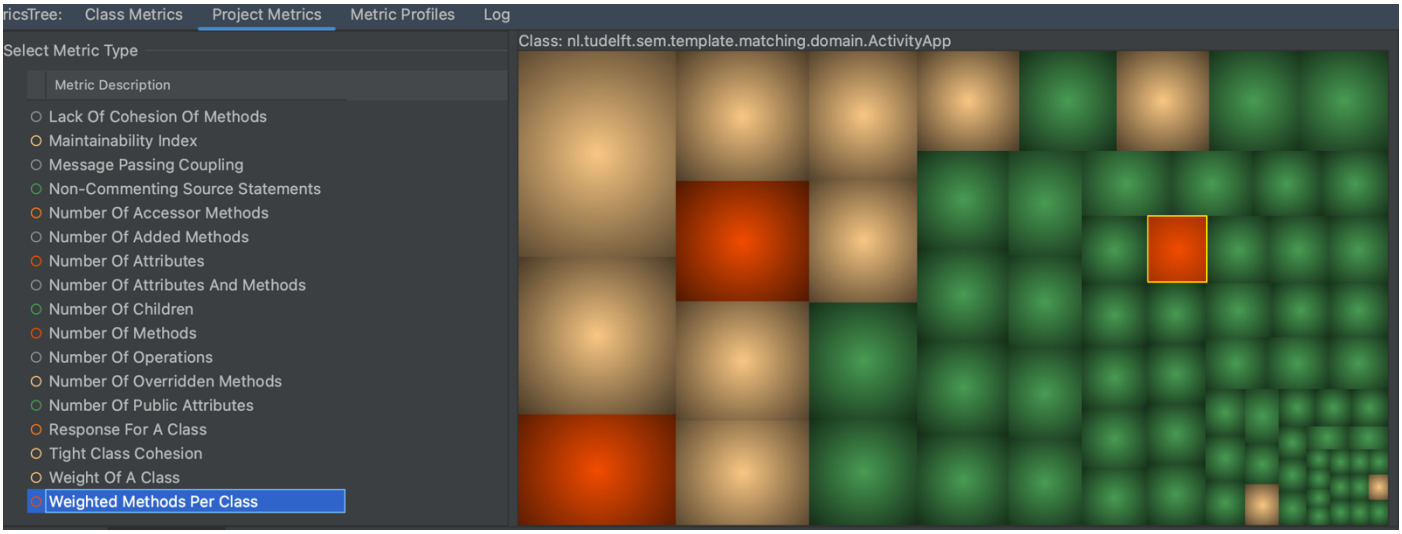
1. Checking the **Project Metrics** for all metrics highlighted in **red** and **orange**
2. For each class pointed out in the step before, we investigated **Class Metrics** and analyzed each method and the class smells
3. We made a list of all these problems and hence identified the weaknesses of the implemented software (methods and classes that would benefit from a refactoring process)

Note that from this final list we prioritized the refactoring that had a considerable impact on the software's scalability, readability and matched the scope and resources of the project.

As a consequence, our refactoring process is going to include the following classes and methods:

CLASSES:

1. **ActivityApp in matching microservice**



We first identified this class by navigating the **Project Metrics** tree over the classes in **red** and **orange.** Afterwards, we checked the **Class Metrics** for this specific one and saw all the problems that it pointed out:

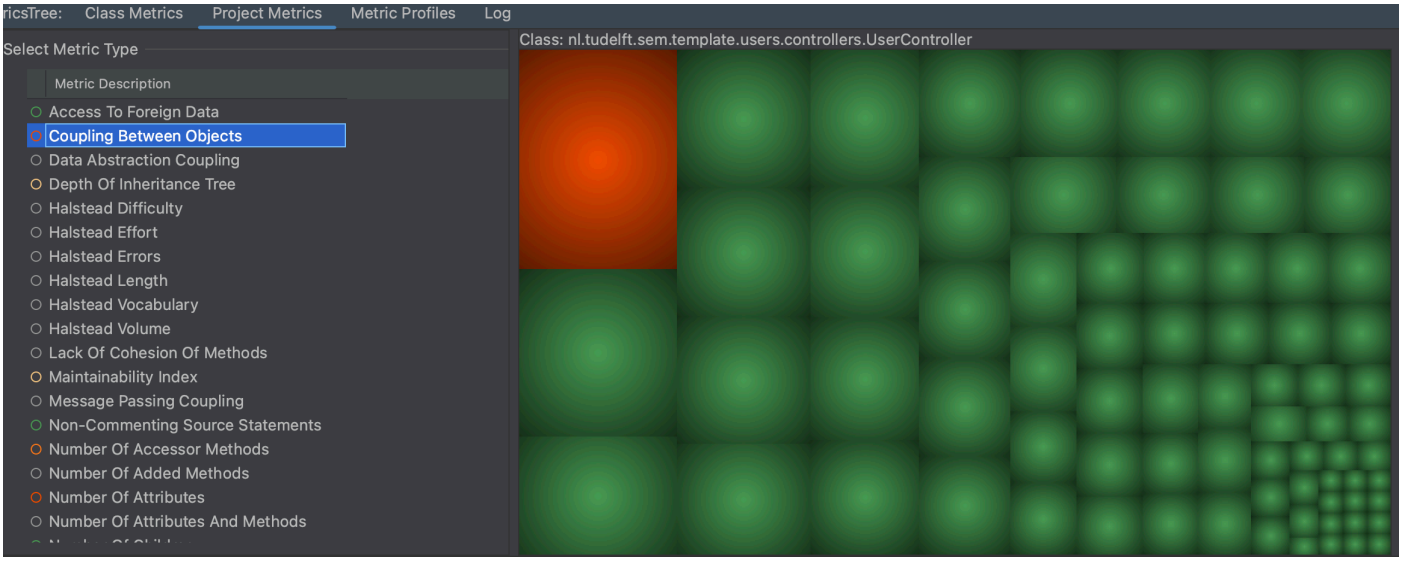
Class: ActivityApp

Metric	Metrics Set	Description	Value	Regular Ra...
CHVL	Halstead ...	Halstead Volume	166.797	
CHD	Halstead ...	Halstead Difficulty	17.5	
CHL	Halstead ...	Halstead Length	40	
CHEF	Halstead ...	Halstead Effort	2918.9475	
CHVC	Halstead ...	Halstead Vocabulary	18	
CHER	Halstead ...	Halstead Errors	0.0681	
WMC	Chidamb...	Weighted Methods Per Class	56	[0..12)
DIT	Chidamb...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamb...	Response For A Class	25	[0..45)
LCOM	Chidamb...	Lack Of Cohesion Of Methods	8	
NOC	Chidamb...	Number Of Children	0	[0..2)
NOA	Lorenz-K...	Number Of Attributes	9	[0..4)
NOO	Lorenz-K...	Number Of Operations	38	
NOOM	Lorenz-K...	Number Of Overridden Methods	3	[0..3)
NOAM	Lorenz-K...	Number Of Added Methods	20	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	46	
NOM	Li-Henry ...	Number Of Methods	25	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	6	
DAC	Li-Henry ...	Data Abstraction Coupling	4	
NCSS	Chr. Cle...	Non-Commenting Source Statements	9	[0..1000)
CMI	Maintain...	Maintainability Index	40.6722	[0.0..19.0]

In order of gravity these are the metrics that need to be improved:

- **Weighted Methods Per Class –** as pointed out the normal threshold is **lower than 12** and our **current metric is 56**; this metric is calculated as the ***weighted sum of methods within the class*** so most probably the *@Data annotation* together with the increased number of methods and attributes is the main concern
- **Number of Attributes** – normal number is **between 0 and 4**, however our **current one is 9,** due to the encapsulation of a considerable amount of information regarding trainings/competitions
- **Number of Methods** – normal number is **between 0 and 7**, but we unfortunately have **25** so the refactoring should include erasing or extracting in another class of some of the methods implemented

2. MatchingService in matching subsystem



After checking the **Project Metrics** and identifying various problems in this class we also investigated the **Class Metrics** to get a grasp of the overall problem:

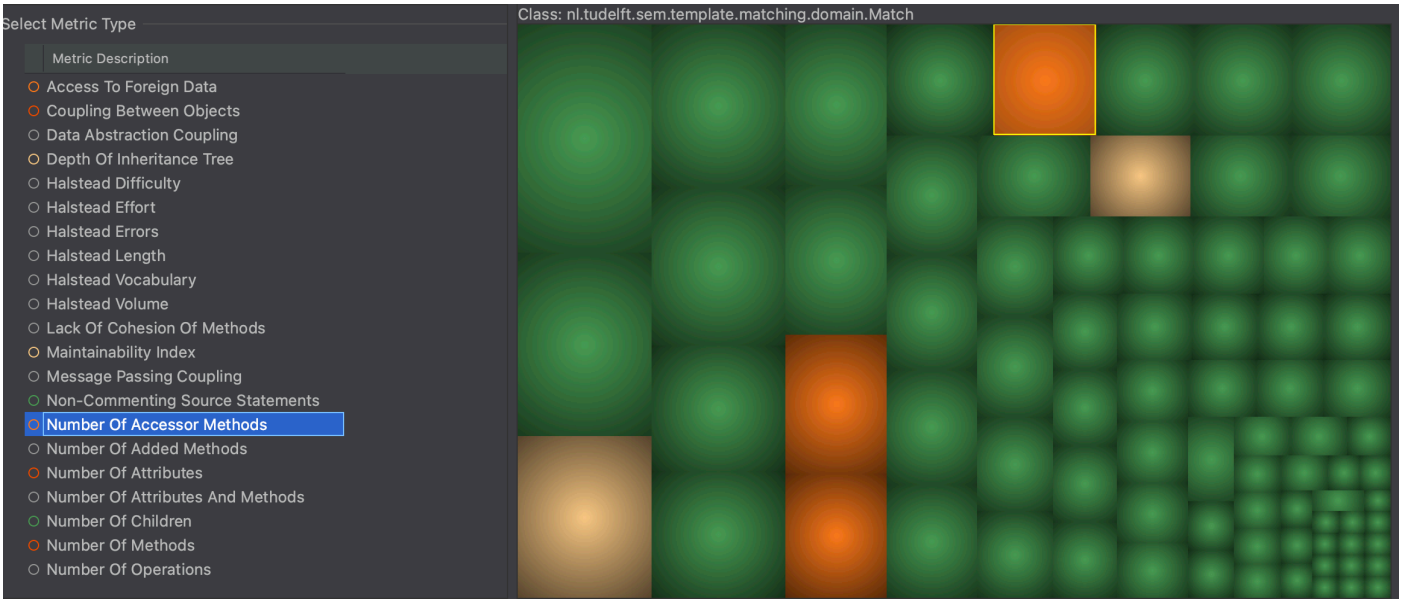
Class: MatchingService

Metric	Metrics Set	Description	Value	Regular Ra...
CHVL	Halstead ...	Halstead Volume	2316.3497	
CHD	Halstead ...	Halstead Difficulty	102.7391	
CHL	Halstead ...	Halstead Length	339	
CHEF	Halstead ...	Halstead Effort	237979.7...	
CHVC	Halstead ...	Halstead Vocabulary	114	
CHER	Halstead ...	Halstead Errors	1.2801	
WMC	Chidamb...	Weighted Methods Per Class	25	[0..12)
DIT	Chidamb...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamb...	Response For A Class	65	[0..45)
LCOM	Chidamb...	Lack Of Cohesion Of Methods	2	
NOC	Chidamb...	Number Of Children	0	[0..2)
NOA	Lorenz-K...	Number Of Attributes	7	[0..4)
NOO	Lorenz-K...	Number Of Operations	27	
NOOM	Lorenz-K...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-K...	Number Of Added Methods	13	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	33	
NOM	Li-Henry ...	Number Of Methods	14	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	92	
DAC	Li-Henry ...	Data Abstraction Coupling	7	
NCSS	Chr. Cle...	Non-Commenting Source Statements	73	[0..1000)
CMI	Maintain...	Maintainability Index	24.8238	[0.0..19.0]

In order of gravity these are the metrics that need to be improved:

- ➔ **Coupling between objects –** as pointed out the normal threshold is **lower than 13** and our **current metric is 27**; this metric is calculated as the ***number of classes coupled with this one*** as the MatchingService class contains the main functionality of the subsystem it is understandable, but the goal is to lower this metric as much as possible
- ➔ **Response for a class –** normal number is **between 0 and 45**, however our **current one is 65**; this is clearly a result of the class calling various other methods (the consequence of the fact that it handles the main functionality required by the software) as the metric represents ***the sum of the methods of the class, and all distinct methods that are invoked directly within the class methods***

3. Match in matching subsystem



The **Project Metrics** clearly show a problem with the "Number of Accessor methods" in this class, however we also checked the **Class Metrics** to get a bigger picture of the overall issue:

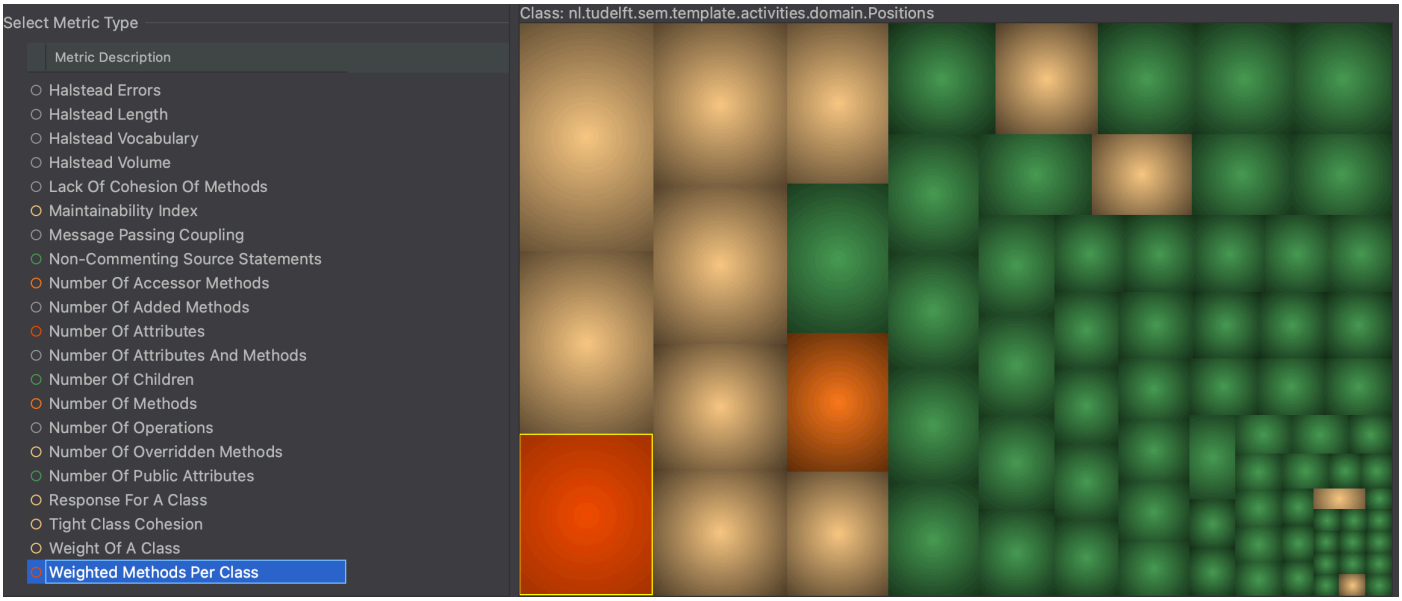
Class: Match

Metric	Metrics Set	Description	Value	Regular Ra...
CHVL	Halstead ...	Halstead Volume	486.257	
CHD	Halstead ...	Halstead Difficulty	11.76	
CHL	Halstead ...	Halstead Length	92	
CHEF	Halstead ...	Halstead Effort	5718.3824	
CHVC	Halstead ...	Halstead Vocabulary	39	
CHER	Halstead ...	Halstead Errors	0.1066	
WMC	Chidamb...	Weighted Methods Per Class	13	[0..12)
DIT	Chidamb...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamb...	Response For A Class	12	[0..45)
LCOM	Chidamb...	Lack Of Cohesion Of Methods	6	
NOC	Chidamb...	Number Of Children	0	[0..2)
NOA	Lorenz-K...	Number Of Attributes	6	[0..4)
NOO	Lorenz-K...	Number Of Operations	24	
NOOM	Lorenz-K...	Number Of Overridden Methods	2	[0..3)
NOAM	Lorenz-K...	Number Of Added Methods	7	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	29	
NOM	Li-Henry ...	Number Of Methods	11	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	1	
DAC	Li-Henry ...	Data Abstraction Coupling	2	
NCSS	Chr. Cle...	Non-Commenting Source Statements	21	[0..1000)
CMI	Maintain...	Maintainability Index	37.3104	[0.0..19.0]

As seen above all the other metrics are within the threshold, leaving only the following one for refactoring:

- ➔ **Number of accessor methods** – normal number is **between 0 and 3**, however our **current one is 7**; this is a result of the class having too many attributes as the metric represents ***the sum of the getters and setters within that certain class*** meaning that the class needs to be analyzed for further reductions in the number of attributes (since all of them are currently in use in other class, thus requiring the getters and setters written)

4. Positions in activity subsystem



After checking the **Project Metrics** and identifying various problems of this class we also investigated the **Class Metrics** to have an idea of the whole problem:

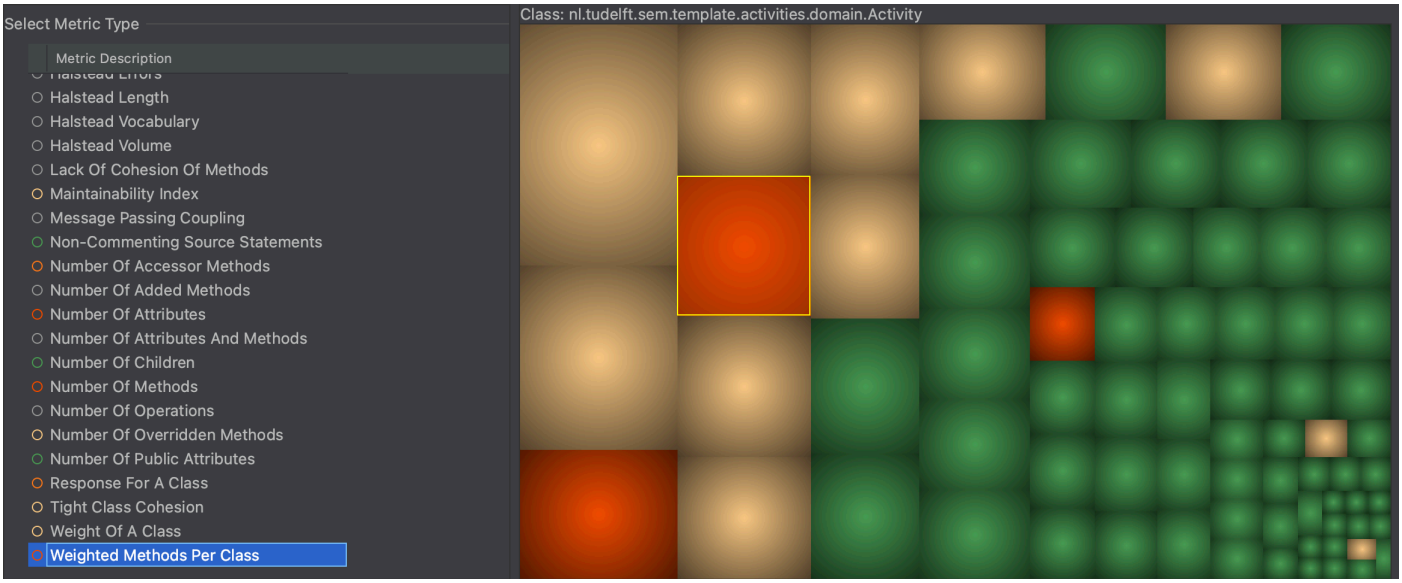
Class: Positions

Metric	Metrics Set	Description	Value	Regular Ra..
CHVL	Halstead ...	Halstead Volume	735.9083	
CHD	Halstead ...	Halstead Difficulty	26.8571	
CHL	Halstead ...	Halstead Length	134	
CHEF	Halstead ...	Halstead Effort	19764.39...	
CHVC	Halstead ...	Halstead Vocabulary	45	
CHER	Halstead ...	Halstead Errors	0.2437	
WMC	Chidamb...	Weighted Methods Per Class	50	[0..12)
DIT	Chidamb...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamb...	Response For A Class	17	[0..45)
LCOM	Chidamb...	Lack Of Cohesion Of Methods	2	
NOC	Chidamb...	Number Of Children	0	[0..2)
NOA	Lorenz-K...	Number Of Attributes	5	[0..4)
NOO	Lorenz-K...	Number Of Operations	30	
NOOM	Lorenz-K...	Number Of Overridden Methods	2	[0..3)
NOAM	Lorenz-K...	Number Of Added Methods	13	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	34	
NOM	Li-Henry ...	Number Of Methods	17	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	15	
DAC	Li-Henry ...	Data Abstraction Coupling	1	
NCSS	Chr. Cle...	Non-Commenting Source Statements	40	[0..1000)
CMI	Maintain...	Maintainability Index	34.5348	[0.0..19.0]

In order of gravity these are the metrics that need to be improved:

- **Weighted Methods Per Class –** as pointed out the normal threshold is **lower than 12** and our **current metric is 50**; this metric is calculated as the ***weighted sum of methods within the class*** so most probably the *annotations* together with the increased number of methods and attributes are the main concerns
- **Number of Methods –** normal number is **between 0 and 7**, but we unfortunately have **17** so the refactoring should include erasing or extracting in another class of some of the methods implemented

5. Activity in activity subsystem



After checking the **Project Metrics** and identifying various problems of this class we also investigated the **Class Metrics** to get a grasp of the overall problem:

Class: Activity

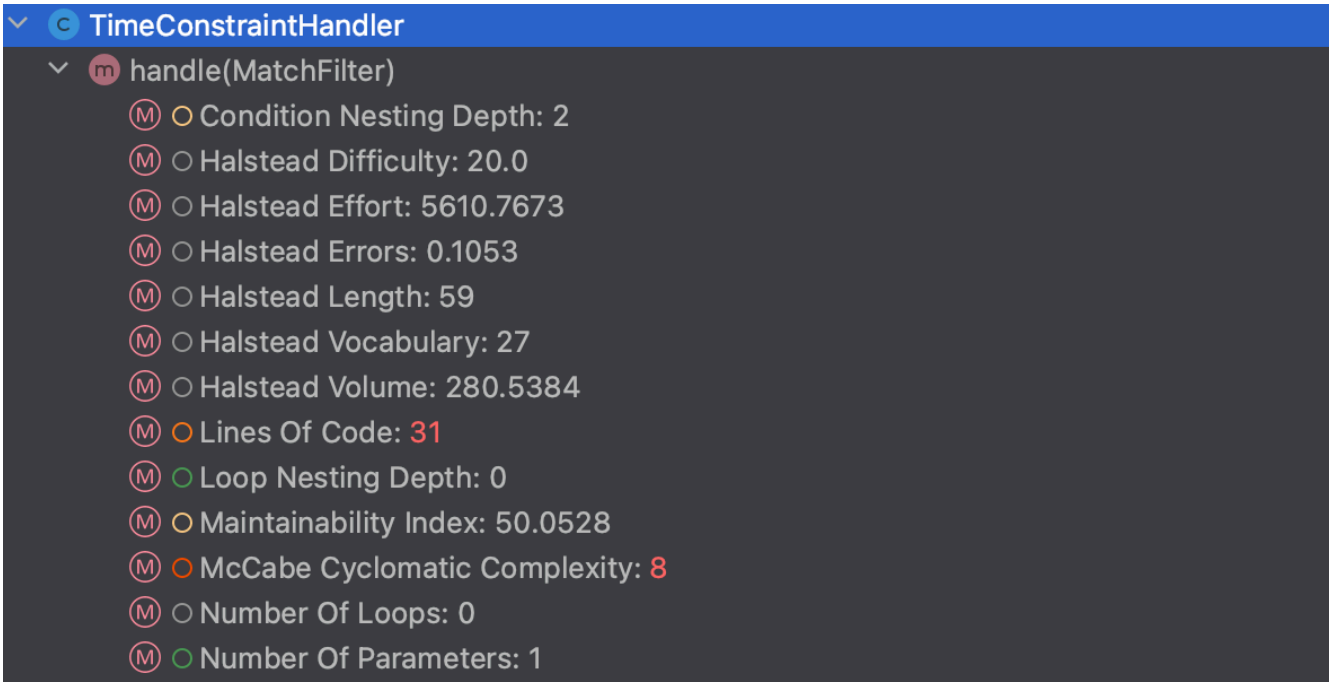
Metric	Metrics Set	Description	Value	Regular Ra...
CHVL	Halstead ...	Halstead Volume	1246.5019	
CHD	Halstead ...	Halstead Difficulty	41.9535	
CHL	Halstead ...	Halstead Length	195	
CHEF	Halstead ...	Halstead Effort	52295.10...	
CHVC	Halstead ...	Halstead Vocabulary	84	
CHER	Halstead ...	Halstead Errors	0.4661	
WMC	Chidamber-Kemerer Metrics Set	er Class	68	[0..12)
DIT	Chidamb...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamb...	Response For A Class	39	[0..45)
LCOM	Chidamb...	Lack Of Cohesion Of Methods	3	
NOC	Chidamb...	Number Of Children	0	[0..2)
NOA	Lorenz-K...	Number Of Attributes	10	[0..4)
NOO	Lorenz-K...	Number Of Operations	36	
NOOM	Lorenz-K...	Number Of Overridden Methods	2	[0..3)
NOAM	Lorenz-K...	Number Of Added Methods	19	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	45	
NOM	Li-Henry ...	Number Of Methods	23	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	33	
DAC	Li-Henry ...	Data Abstraction Coupling	5	
NCSS	Chr. Cle...	Non-Commenting Source Statements	36	[0..1000)
CMI	Maintain...	Maintainability Index	30.942	[0.0..19.0]

In order of gravity these are the metrics that need to be improved:

- **Weighted Methods Per Class –** as pointed out the normal threshold is **lower than 12** and our **current metric is 36**; this metric is calculated as the ***weighted sum of methods within the class*** so most probably the increased number of methods and attributes are the main concerns (aspect also pointed out by the other increased metrics
- **Number of Attributes –** normal number is **between 0 and 4**, however our **current one is 10**
- **Number of Methods –** normal number is **between 0 and 7**, but we unfortunately have **22** so the refactoring should include erasing or extracting in another class of some of the methods implemented

METHODS:

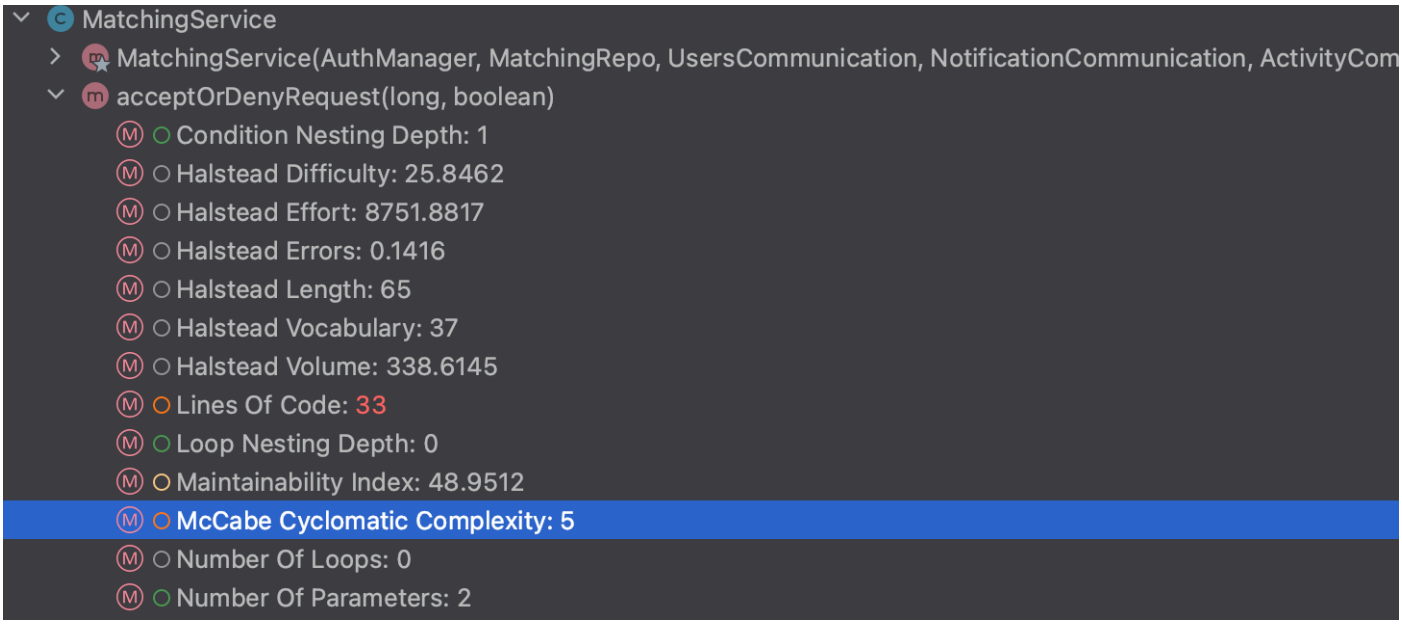
1. **handle in TimeConstraintHandler of matching subsystem**



By checking the **Class Metrics** for the TimeConstraintHandler we identified the following problems in the **handle method** of this class:

- ➔ **Lines of code** – normal number is **between 0 and 11**, however our **current one is 31**, which means the method should probably be simplified by extracting code or refactoring the logic
- ➔ **McCabe Cyclomatic Complexity** – ***computed as the number of linearly independent paths*** the threshold is **between 0 and 3**, but we unfortunately have **8** so the refactoring should include getting rid of the switch and other if statements that naturally increase this number

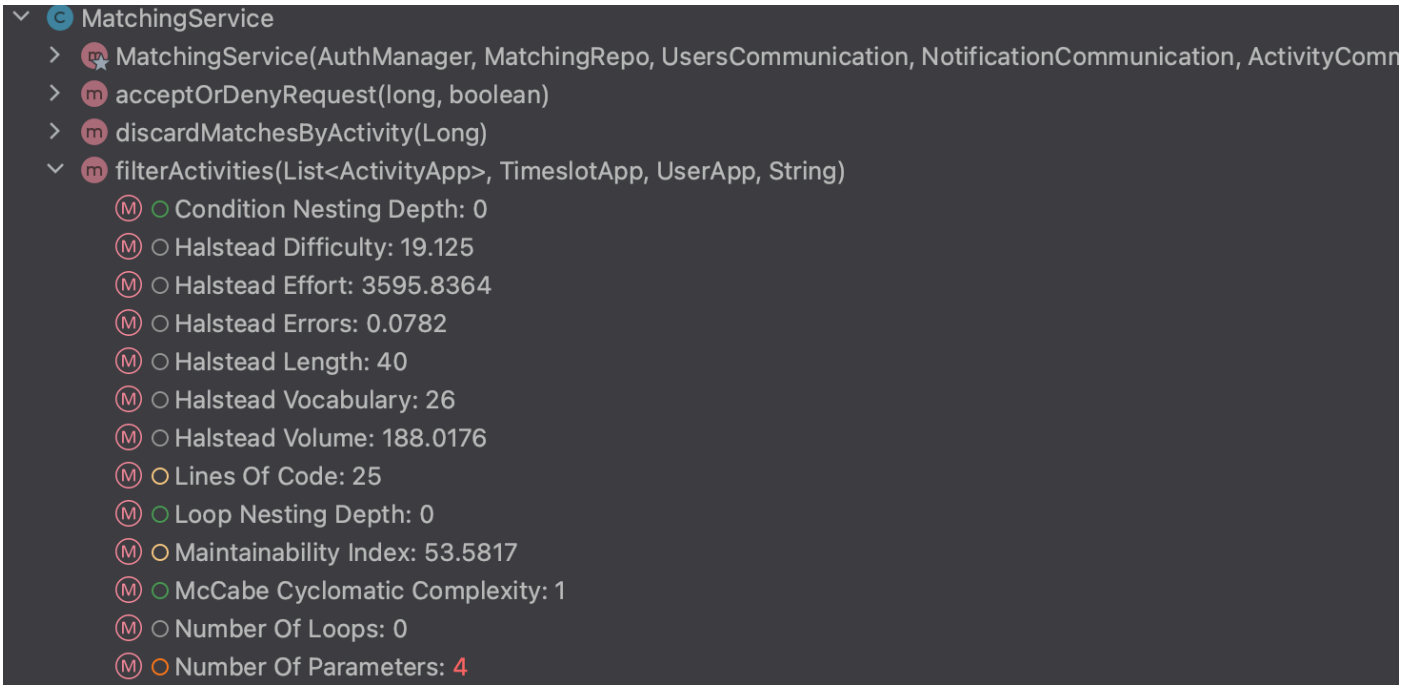
2. **acceptOrDenyRequest in MatchingService of matching subsystem**



By checking the **Class Metrics** for the MatchingService we identified the following problems in the **acceptOrDenyRequest method** of this class:

- **Lines of code** – normal number is **between 0 and 11**, however our **current one is 33**, which means the method should probably be simplified by extracting code or refactoring the logic
- **McCabe Cyclomatic Complexity** – ***computed as the number of linearly independent paths*** the threshold is **between 0 and 3**, but we unfortunately have **5** so the refactoring should include decreasing the number of if statements and/or extracting part of the sanitization done to another method

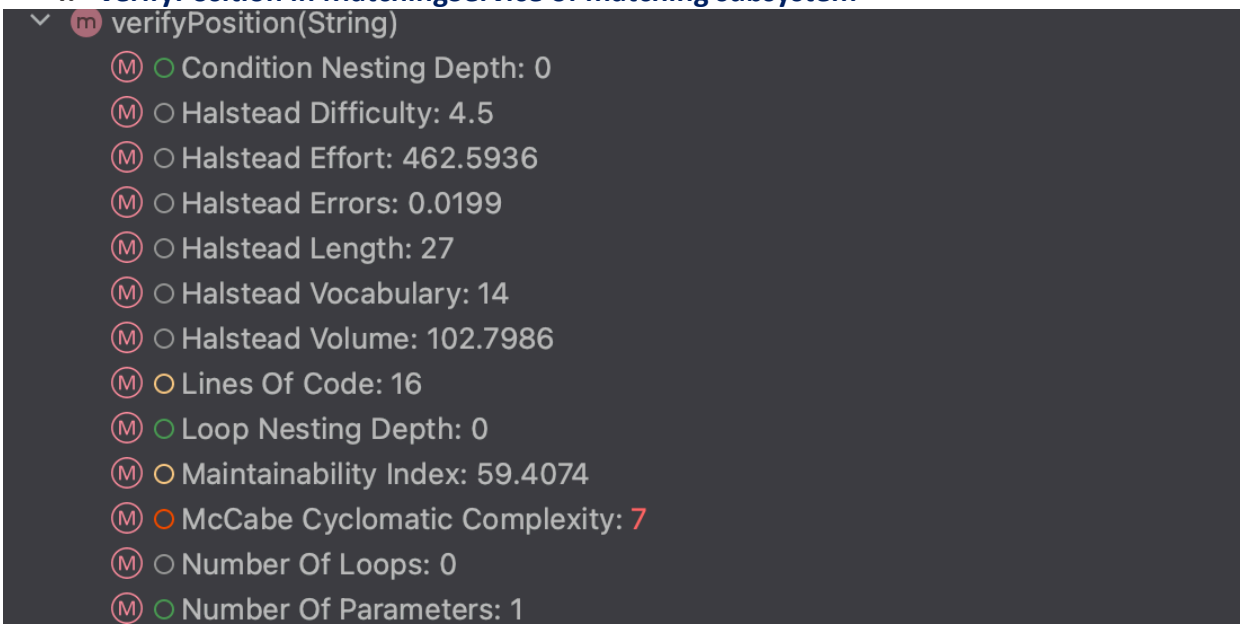
3. filterActivities in MatchingService of matching subsystem



The **Class Metrics** for the MatchingService class point out the following problem in the **filterActivities method** of this class:

- **Number of parameters** – normal number is **between 0 and 3**, but we **currently have 4**, which means the method parameters should probably be grouped together in another entity to decrease the metric and also simplify the readability and maintainability of the code

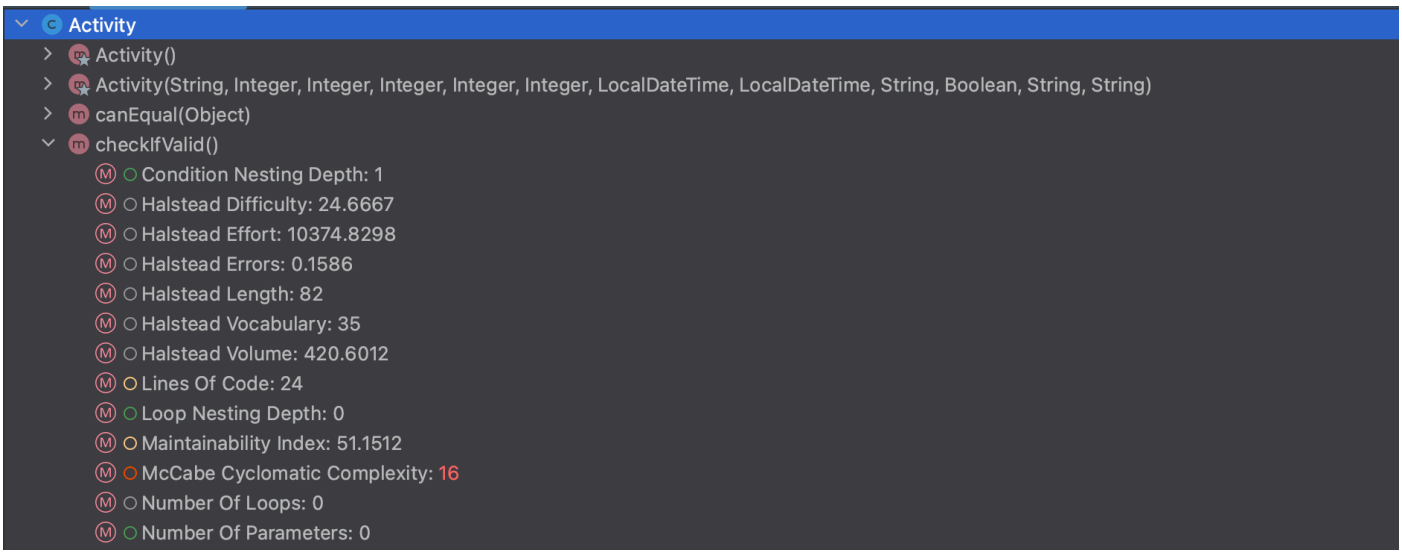
4. verifyPosition in MatchingService of matching subsystem



The **Class Metrics** for the MatchingService class show the following problem in the **verifyPosition method** of this class:

➔ **McCabe Cyclomatic Complexity** – ***computed as the number of linearly independent paths*** the threshold is **between 0 and 3**, but we unfortunately have **7** so the refactoring should probably refer to the switch statement used in this function by trying to avoid its use

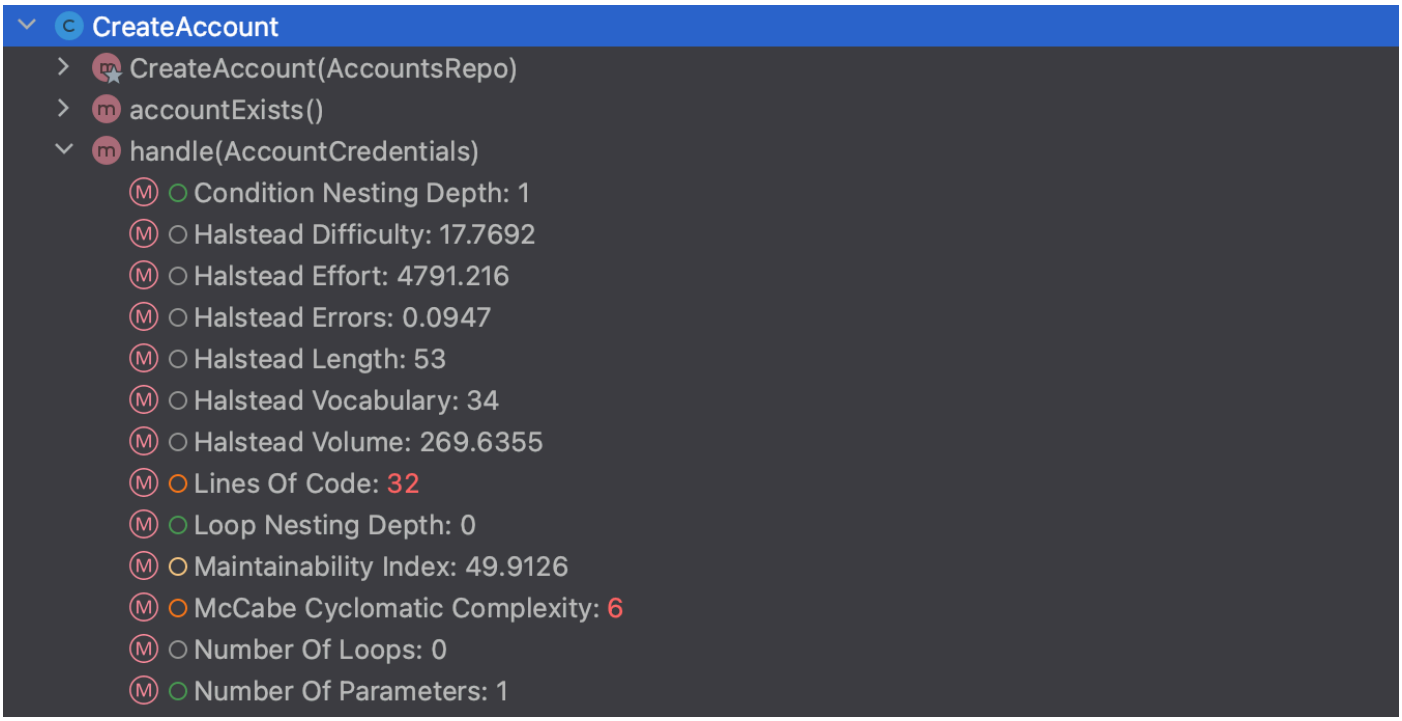
5. **checkIfValid in Activity of activity microservice**



After checking the **Project Metrics** for the Activity class, we decided to look closely onto the method it contains by analyzing the **Class Metrics** and as a consequence we identified the following problem in the **checkIfValid method** of this class:

➔ **McCabe Cyclomatic Complexity** – ***computed as the number of linearly independent paths*** the threshold is **between 0 and 3**, but we unfortunately have **16** so the refactoring should probably include simplifying the logic behind the sanitization process and maybe extracting some part of the code

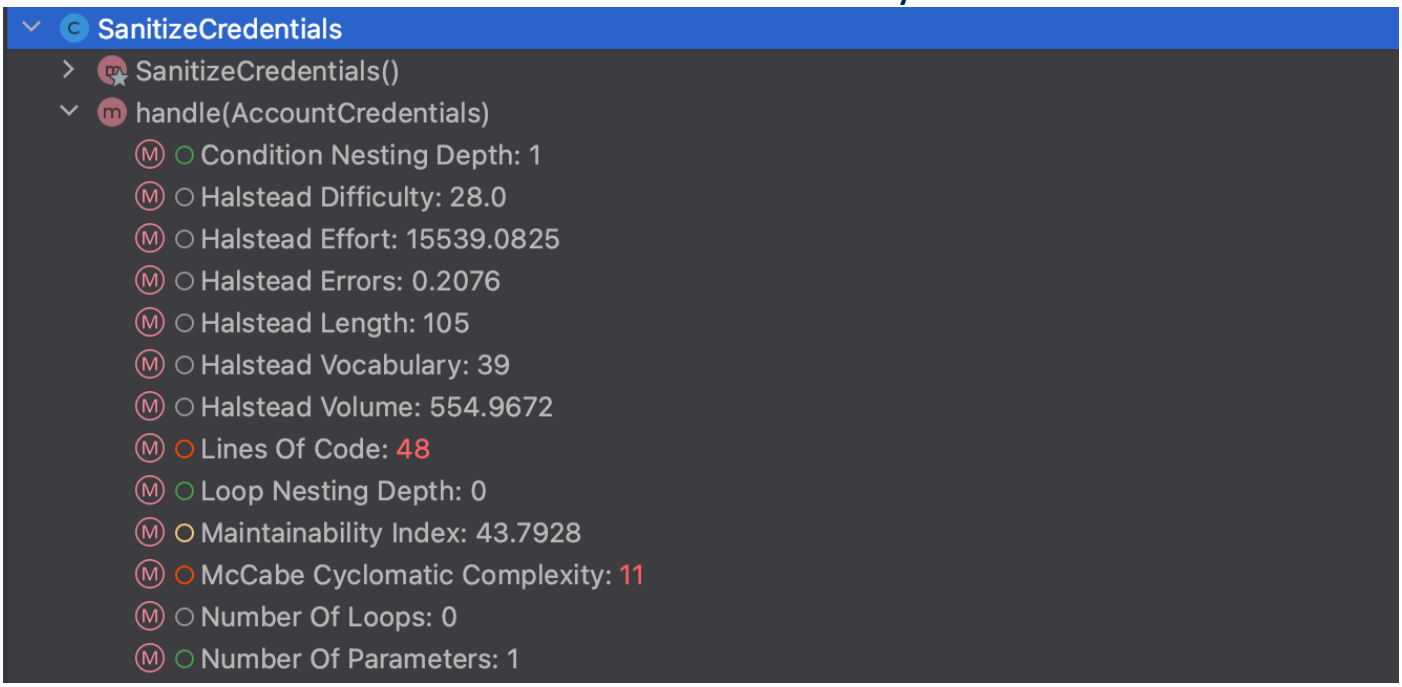
6. **handle in CreateAccount handler in authentication subsystem**



By checking the **Project Metrics** for the Authentication microservice we ended up analyzing the **Class Metrics** of the classes included in this package and as a consequence we identified the following problem in the **handle method of the CreateAccount class**:

- **Lines of code** – normal number is **between 0 and 11**, however our **current one is 32**, which means the method should probably be simplified by extracting code or refactoring the logic
- **McCabe Cyclomatic Complexity** – ***computed as the number of linearly independent paths*** the threshold is **between 0 and 3**, but we unfortunately have **6** so the refactoring should include decreasing the number of if statements and/or extracting part of the checks done to another method

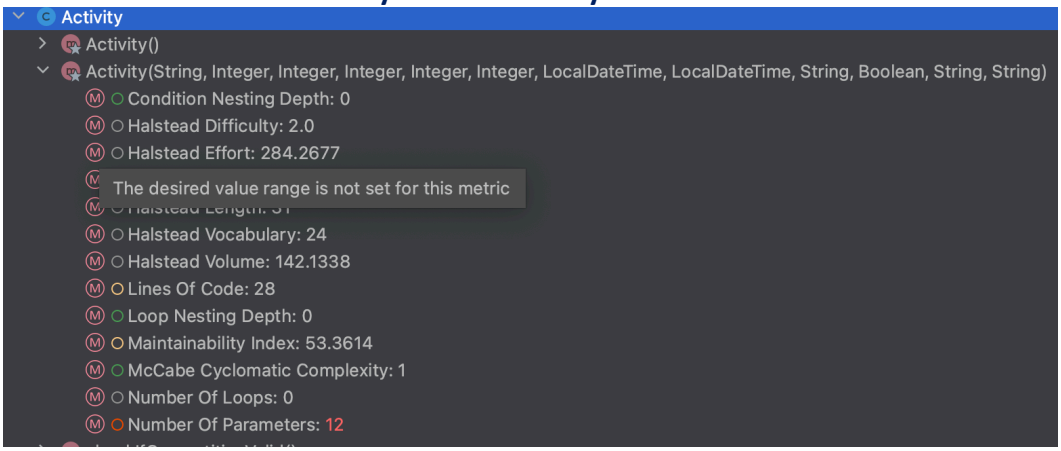
7. handle in SanitizeCredentials handler in authentication subsystem



Continuing with our search for **red** highlighted metrics in the **Authentication microservice** we identified the following problem

- **Lines of code** – normal number is **between 0 and 11**, however our **current one is 48**, which clearly shows a considerable amount of lines of code which should be refactored
- **McCabe Cyclomatic Complexity** – ***computed as the number of linearly independent paths*** the threshold is **between 0 and 3**, but we unfortunately have **11** and points out to the number of if statements used for sanitizations checks enforced, thus those should probably be included in separate methods for decluttering the code and decreasing the impact of the metric

8. constructor in Activity class of activity microservice



The **Class Metrics** for the Activity class point out the following problem in the **constructor** of this class:

- **Number of parameters –** normal number is **between 0 and 3**, but we **currently have 12**, which means the class has overall an increased number of attributes and reducing this number should be a priority for the refactoring of this class

In summary, these are the metrics that we are going to improve during the refactoring assignment:

CLASSES:

- **Weighted Methods Per Class – weighted sum of methods implemented within a class should be lower than 12** and an increased value of this metric points out to reduced maintainability and re-usability, thus it's critical to decrease the value of this metric as much as possible

- **Coupling between Objects – number of other classes that a class is coupled to should be lower than 14** and an increased value of this metric points out to reduced maintainability and re-usability and nonetheless coupling is undesirable in the context of a microservice based software architecture

- **Response for a class – total number of methods that can potentially be executed in response to a message received by an object of a class should be lower than 45** and an increased value of this metric calls for decreased adaptability of the software (consequently we decided to bring down its value in all the problematic classes)

- **Number of accessor methods – total number of getters and setters within a class should be lower than 3** and given the fact that the desired number of attributes within a class is 4, we aim to target refactoring for classes that have an orange or red metric (as all the classes that encapsulate critical information have also an increased number of attributes and, thus become problematic)

- **Number of attributes – total number of attributes within a class should be around 4** and an increased value of this metric is often linked to a cluttered class, thus decreasing the values of this metric in the classes that show irregularities would benefit maintainability and the overall evolution of the software

- **Number of methods – total number of methods within a class should be lower than 7**; the values of this metric should be reduced until the reasonable threshold as the portability and maintainability of the software are directly connected to this metric

METHODS:

- **McCabe Cyclomatic Complexity – computed as the number of linearly independent paths should be lower than 3** and it also represents **a measure of the control structure complexity of software,** thus we considered critical trying to declutter the methods with increased values for this metrics in the refactoring process as this would significantly increase the readability and maintainability of the methods

- **Lines of code – number of lines within a method should be lower than 11** and an increased value of this metric points out to reduced maintainability and efficiency of the methods, therefore we will definitely try to reduce as much as possible this metric for the methods that show irregularities

- **Number of parameters – total number of parameters within a method call should be lower than 3** and this metric makes sense in the context of a reusable and portable software which we aim to have, thus making this metric one of the refactoring metrics we want to decrease for this assignment

To wrap up, by following the beforementioned process we came up with the method and class smells detailed above that need refactoring as the metrics point out to extreme irregularities compared to the normal thresholds.

Task 2

CLASSES:

1. ActivityApp in matching microservice

These were the metrics before the refactoring:

Class: ActivityApp

Metric	Metrics Set	Description	Value	Regular Ra..
WMC	Chidamber...	Weighted Methods Per Class	56	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
CBO	Chidamber...	Coupling Between Objects	12	[0..14)
RFC	Chidamber...	Response For A Class	25	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	8	
NOC	Chidamber...	Number Of Children	0	[0..2)
NOA	Lorenz-Kid...	Number Of Attributes	9	[0..4)
NOO	Lorenz-Kid...	Number Of Operations	38	
NOOM	Lorenz-Kid...	Number Of Overridden Methods	3	[0..3)
NOAM	Lorenz-Kid...	Number Of Added Methods	20	
SIZE2	Li-Henry M...	Number Of Attributes And Methods	46	
NOM	Li-Henry M...	Number Of Methods	25	[0..7)
MPC	Li-Henry M...	Message Passing Coupling	6	
DAC	Li-Henry M...	Data Abstraction Coupling	4	
ATFD	Lanza-Mari...	Access To Foreign Data	1	[0..6)
NOPA	Lanza-Mari...	Number Of Public Attributes	0	[0..3)
NOAC	Lanza-Mari...	Number Of Accessor Methods	9	[0..4)
WOC	Lanza-Mari...	Weight Of A Class	0.55	[0.5..1.0]
TCC	Bieman-Ka...	Tight Class Cohesion	0.0474	[0.33..1.0]
NCSS	Chr. Cleme...	Non-Commenting Source Statements	9	[0..1000)

We can immediately see that we have **high Number of Methods**, which is also related to the **high WMC and NOAC**. We can also see that there is a **high Number Of Attributes**, which is also influences the other mentioned metrics.

Firstly, we noticed that the *annotation @Data* included a lot of methods that were not used, so we removed this annotation and added *@Getter* one as we only needed getters. This reduced the NOM, WMC and NOAC.

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class ActivityApp {
    private long id;
    private String ownerId;
    private TimeslotApp timeslot;
    private String gender;
    private String organisation;
    private HashMap<String, Integer> positions;
    private boolean competition;
    private TypeOfActivity type;
    private String certificate;
```

We can also identify that this is a **Long Class smell**, as not all the attributes belong here. To further improve the metrics we also decided to introduce an **Object Parameter** containing 4 of the 9 attributes.

```java
@Getter
@Setter
@AllArgsConstructor
public class ActivityProperties {
    private String gender;
    private String organisation;
    private boolean competition;
    private String certificate;
}
```

This further reduced the NOM, WMC and NOAC for the initial ActivityApp class, as there are fewer getters.

```java
@Getter
@NoArgsConstructor
public class ActivityApp {
    private long id;
    private String ownerId;
    private TimeslotApp timeslot;
    private HashMap<String, Integer> positions;
    @Setter
    private TypeOfActivity type;
    private ActivityProperties properties;
```

Finally, these are the metrics after the class refactoring. As we can see, all the metrics that were not in the required threshold have now been fixed.

Class: ActivityApp

Metric	Metrics Set	Description	Value	Regular Ra...
CHVL	Halstead M...	Halstead Volume	403.9979	
CHD	Halstead M...	Halstead Difficulty	10.0	
CHL	Halstead M...	Halstead Length	74	
CHEF	Halstead M...	Halstead Effort	4039.9794	
CHVC	Halstead M...	Halstead Vocabulary	44	
CHER	Halstead M...	Halstead Errors	0.0846	
WMC	Chidamber...	Weighted Methods Per Class	14	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
CBO	Chidamber...	Coupling Between Objects	13	[0..14)
RFC	Chidamber...	Response For A Class	13	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	5	
NOC	Chidamber...	Number Of Children	0	[0..2)
NOA	Lorenz-Kid...	Number Of Attributes	6	[0..4)
NOO	Lorenz-Kid...	Number Of Operations	23	
NOOM	Lorenz-Kid...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Kid...	Number Of Added Methods	8	
SIZE2	Li-Henry M...	Number Of Attributes And Methods	28	
NOM	Li-Henry M...	Number Of Methods	10	[0..7)
MPC	Li-Henry M...	Message Passing Coupling	6	
DAC	Li-Henry M...	Data Abstraction Coupling	5	
ATFD	Lanza-Mari...	Access To Foreign Data	2	[0..6)
NOPA	Lanza-Mari...	Number Of Public Attributes	0	[0..3)
NOAC	Lanza-Mari...	Number Of Accessor Methods	6	[0..4)
WOC	Lanza-Mari...	Weight Of A Class	0.25	[0.5..1.0]
TCC	Bieman-Ka...	Tight Class Cohesion	0.0714	[0.33..1.0]
NCSS	Chr. Cleme...	Non-Commenting Source Statements	15	[0..1000)
CMI	Maintainab...	Maintainability Index	43.7894	[0.0..19.0]

2. MatchingService in matching subsystem

These were the metrics before refactoring:

Class: MatchingService

Metric	Value	Excess
Access To Foreign Data	6	+1
Coupling Between Objects	28	+15
Data Abstraction Coupling	7	-
Depth Of Inheritance Tree	1	
Halstead Difficulty	90.5213	-
Halstead Effort	19668...	-
Halstead Errors	1.1274	-
Halstead Length	318	-
Halstead Vocabulary	114	-
Halstead Volume	2172.8...	-
Lack Of Cohesion Of Methods	2	-
Maintainability Index	25.448	+6.448
Message Passing Coupling	96	-
Non-Commenting Source Statements	62	
Number Of Accessor Methods	0	
Number Of Added Methods	13	-
Number Of Attributes	7	+4
Number Of Attributes And Methods	33	-
Number Of Children	0	
Number Of Methods	14	+8
Number Of Operations	27	-
Number Of Overridden Methods	0	
Number Of Public Attributes	1	
Response For A Class	69	+25
Tight Class Cohesion	0.5769	
Weight Of A Class	1.0	
Weighted Methods Per Class	19	+8

As we can see we have a **high** **RFC** and **CBO**. However it makes sense that this class is highly coupled as it acts as the main component of the matching microservice and it requires functionality from almost all other classes. To reduce **RFC**, we tried to move some functionality to external classes as to decrease the amount of method calls.

```
public final void filteringHandlerSetUp() {
    this.filteringHandler = new PositionHandler();
    FilteringHandler certificateHandler = new CertificateHandler(certificateRepo);
    this.filteringHandler.setNext(certificateHandler);
    FilteringHandler timeConstraintHandler = new TimeConstraintHandler();
    certificateHandler.setNext(timeConstraintHandler);
    FilteringHandler typeOfActivityHandler = new TypeOfActivityHandler();
    timeConstraintHandler.setNext(typeOfActivityHandler);
    FilteringHandler organisationHandler = new OrganisationHandler();
    typeOfActivityHandler.setNext(organisationHandler);
    FilteringHandler genderHandler = new GenderHandler();
    organisationHandler.setNext(genderHandler);
    FilteringHandler competitivenessHandler = new CompetitivenessHandler();
    genderHandler.setNext(competitivenessHandler);
}
```

We have moved the **Chain of Responsability** setup in a separate class, thus reducing a significant amount of method calls and coupling with each handler.

To further reduce **RFC**, we have changed the functional programming implementation to a for loop with conditional statements. However this increased the cyclomatic complexity of the methods themselves, but we accepted the trade-off since **RFC** had an extreme value.

Note that this is the second refactor of method filterActivities().

```
public List<ActivityReponse> filterActivities(List<ActivityApp> activities,
                                              UserPreferences userPreferences) {
    return activities
            .stream() Stream<ActivityApp>
            .filter(Objects::nonNull)
            .distinct()
            .map(ActivityApp::setTypeOfActivity)
            .filter(Objects::nonNull)
            .filter(a -> this.filteringHandler.handle(new MatchFilter(a, userPreferences)))
            .filter(a -> matchingRepo.getMatchesByActivityIdAndParticipantId(a.getId(), userPreferences.getUser()
                    .getEmail()).isEmpty())
            .map(a -> matchUserToActivity(userPreferences.getUser(), userPreferences.getPosition(), a)) Stream<ActivityReponse>
            .collect(Collectors.toList());
}
```

```
public List<ActivityResponse> filterActivities(List<ActivityApp> activities,
                                               UserPreferences userPreferences) {
    List<ActivityResponse> responses = new ArrayList<>();

    for (ActivityApp activity : activities) {
        if (activity == null
                || activity.setTypeOfActivity() == null
                || !this.filteringHandler.handle(new MatchFilter(activity, userPreferences))
                || !matchingRepo.getMatchByActivityInformation_ActivityIdAndParticipantId(activity.getId(),
                userPreferences.getUser().getEmail()).isEmpty()) {
            continue;
        }
        responses.add(matchUserToActivity(userPreferences.getUser(), userPreferences.getPosition(), activity));
    }

    return responses;
}
```

Same refactoring was applied to discardMatchesByActivity().

```
public void discardMatchesByActivity(Long activityId) {
    List<Match> matchesModifiedByActivityChange = matchingRepo.getMatchesByActivityId(activityId);
    matchesModifiedByActivityChange
            .stream()
            .filter(match -> match.getStatus() == Status.ACCEPTED)
            .forEach(match ->
                    notificationCommunication
                            .activityModifiedNotification(new NotificationActivityModified(match.getParticipantId(),
                                    activityId, activityCommunication.getActivityTimeslotById(activityId))));
    matchesModifiedByActivityChange
            .stream()
            .forEach(match -> matchingRepo.deleteById(match.getMatchId()));
}
```

```
public void discardMatchesByActivity(Long activityId) {
    List<Match> matchesModifiedByActivityChange = matchingRepo.getMatchesByActivityInformation_ActivityId(activityId);
    for (Match match : matchesModifiedByActivityChange) {
        if (match.getStatus() == Status.ACCEPTED) {
            communication.getNotificationCommunication()
                    .activityModifiedNotification(new NotificationActivityModified(match.getParticipantId(),
                            activityId, communication.getActivityCommunication().getActivityTimeslotById(activityId)));
        }
        matchingRepo.deleteById(match.getMatchId());
    }
}
```

Next, to reduce coupling, we have extracted methods and attributes to other objects.

```
public class MatchingService {

    private final transient AuthManager auth;
    private final transient MatchingRepo matchingRepo;
    private final transient UsersCommunication usersCommunication;
    private final transient NotificationCommunication notificationCommunication;
    private final transient ActivityCommunication activityCommunication;
    public transient FilteringHandler filteringHandler;
    private final transient CertificateRepo certificateRepo;
```

```
private final transient AuthManager auth;
private final transient MatchingRepo matchingRepo;
private final transient Communication communication;
public transient FilteringHandler filteringHandler;
```

The 3 communication classes were exchanged with an **Object Parameter**, which allows for a more readable code. This also reduced both **CBO** and **RFC**. **“certificateRepo”** was also extracted along with other methods to a **Sanitization class**, which handles the verification of input.

```
private boolean verifyUser(long matchId) {
    return auth.getUserId().equals(matchingRepo.getMatchByMatchId(matchId).get().getParticipantId());
}
public boolean verifyMatch(long matchId) {
    return matchingRepo.getMatchByMatchId(matchId).isPresent() && verifyUser(matchId);
}
public boolean verifyPosition(String position) {
    List<String> validPositions = List.of("cox", "port", "coach", "starboard", "sculling");
    return validPositions.contains(position);
}
public List<Match> getPendingRequests() {
    return matchingRepo.getMatchesByOwnerIdAndStatus(auth.getUserId(), Status.PENDING);
}
```

```java
public List<Match> getMatches(Status status) {
    String userId = auth.getUserId();
    return matchingRepo.getMatchesByParticipantIdAndStatus(userId, status);
}
```

We considered that these methods should not be related to the **MatchingService class** we applied **Extract Method Refactoring**.

```java
public class Sanitization {

    private final transient AuthManager auth;
    private final transient MatchingRepo matchingRepo;
```

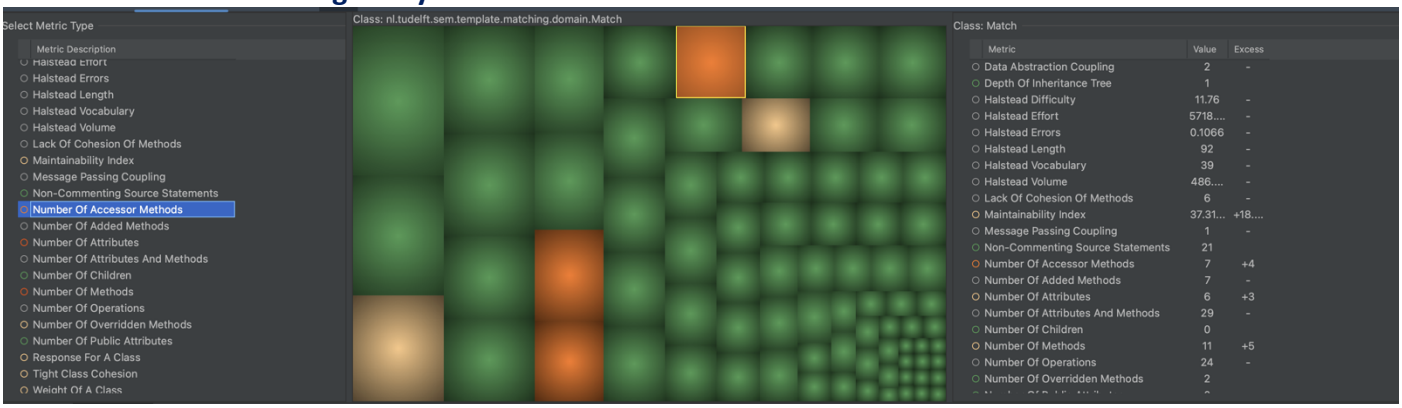
Finally, these are the metrics after the refactoring

Class: MatchingService

Metric	Value	Excess
Access To Foreign Data	7	+2
Coupling Between Objects	23	+10
Data Abstraction Coupling	4	-
Depth Of Inheritance Tree	1	
Halstead Difficulty	95.6207	-
Halstead Effort	14329...	-
Halstead Errors	0.9128	-
Halstead Length	232	-
Halstead Vocabulary	88	-
Halstead Volume	1498.5...	-
Lack Of Cohesion Of Methods	1	-
Maintainability Index	30.6213	+11.62...
Message Passing Coupling	72	-
Non-Commenting Source Statements	50	
Number Of Accessor Methods	0	
Number Of Added Methods	7	-
Number Of Attributes	4	+1
Number Of Attributes And Methods	24	-
Number Of Children	0	
Number Of Methods	8	+2
Number Of Operations	21	-
Number Of Overridden Methods	0	
Number Of Public Attributes	1	
Response For A Class	56	+12
Tight Class Cohesion	0.7143	
Weight Of A Class	1.0	
Weighted Methods Per Class	19	+8

As you can see, the **RFC** is now in the accepted threshold we have discussed, but even though **CBO** has been reduced by **5**, it still over the range. However, since this class depends on multiple other objects and it handles most of the functionality of **the Matching microservice** which is difficult or maybe impossible to extract further, we consider that it cannot be reduced more due to the complexity of this microservice.

3. Match in matching subsystem



As we can see, this class has a **high** **Number Of Accessor Methods**, which is also related to the **high** **Number Of Attributes**.

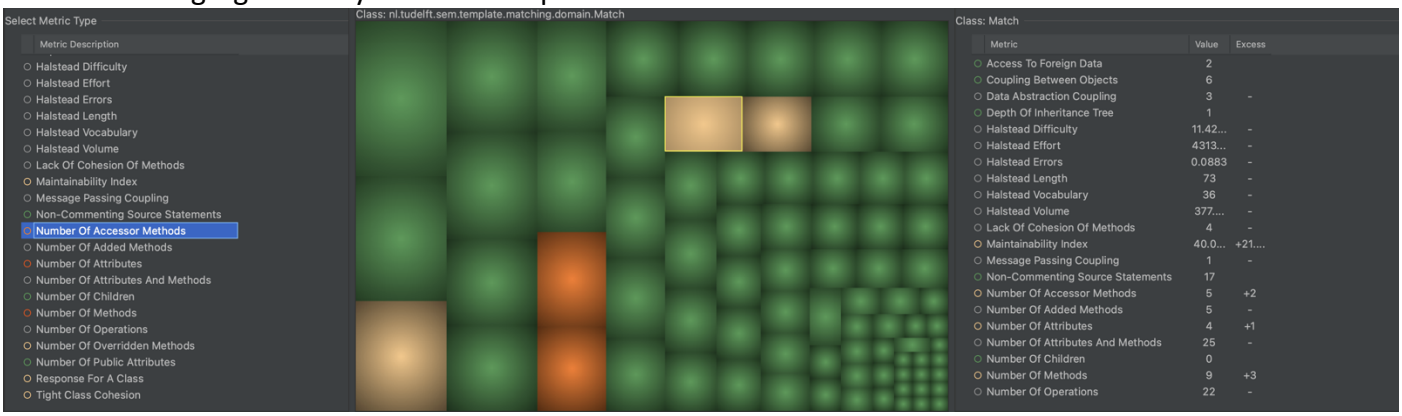
```java
public class Match {

    /**
     * Entity that depicts the match being made by the system
     * between an user with an availability and a certain activity.
     */
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", unique = true, nullable = false)
    private long matchId;

    @Column(name = "participant_id", nullable = false)
    private String participantId;

    @Column(name = "acivity_id", nullable = false)
    private long activityId;

    @Column(name = "owner_id", nullable = false)
    private String ownerId;

    @Column(name = "position", nullable = false)
    private String position;

    @Column(name = "status", nullable = false)
    private Status status;
```

This represents a **Long Class smell**, so we decided to include an **Object Parameter** which encapsulates attributes that are closely related to each other, namely *activityId, ownerId and position.*

```java
public class ActivityInformation {

    @Column(name = "acivity_id", nullable = false)
    private long activityId;

    @Column(name = "owner_id", nullable = false)
    private String ownerId;

    @Column(name = "position", nullable = false)
    private String position;
```

```java
public class Match {

    /**
     * Entity that depicts the match being made by the system
     * between an user with an availability and a certain activity.
     */
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", unique = true, nullable = false)
    private long matchId;

    @Column(name = "participant_id", nullable = false)
    private String participantId;

    @Embedded
    private ActivityInformation activityInformation;

    @Column(name = "status", nullable = false)
    private Status status;
```

This refactoring significantly reduced the problematic metrics:



4. Positions in activity subsystem

The Positions class contains information about the types of positions available for a training or competition, as well as their number. Initially, the **MetricsTree** showed that the **Weighted Methods per Class and the Number of Methods metrics** surpassed the suggested limit.

Class: Positions

Metric	Metrics Set	Description	Value	Regular Ra...
CHVL	Halstead...	Halstead Volume	735.9083	
CHD	Halstead...	Halstead Difficulty	26.8571	
CHL	Halstead...	Halstead Length	134	
CHEF	Halstead...	Halstead Effort	19764.3...	
CHVC	Halstead...	Halstead Vocabulary	45	
CHER	Halstead...	Halstead Errors	0.2437	
WMC	Chidamb...	Weighted Methods Per Class	50	[0..12)
DIT	Chidamb...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamb...	Response For A Class	17	[0..45)
LCOM	Chidamb...	Lack Of Cohesion Of Methods	2	
NOC	Chidamb...	Number Of Children	0	[0..2)
NOA	Lorenz-...	Number Of Attributes	5	[0..4)
NOO	Lorenz-...	Number Of Operations	30	
NOOM	Lorenz-...	Number Of Overridden Methods	2	[0..3)
NOAM	Lorenz-...	Number Of Added Methods	13	
SIZE2	Li-Henry...	Number Of Attributes And Methods	34	
NOM	Li-Henry...	Number Of Methods	17	[0..7)
MPC	Li-Henry...	Message Passing Coupling	15	
DAC	Li-Henry...	Data Abstraction Coupling	1	
NCSS	Chr. Cle...	Non-Commenting Source Statements	40	[0..1000)
CMI	Maintain...	Maintainability Index	34.5348	[0.0..19.0]

During the development phase of this app, we generally used the *'@EqualsAndHashCode' annotation* from *Lombok* to generate the implementation of these boilerplate methods in most of our classes. We found out that the **'equals' and the 'hashCode' methods** corresponding to the Positions class, which were included through that annotation, were not used anywhere in the project. Therefore we decided to remove them and to alter the tests accordingly. This change improved the **WMC** by **21 points (50 -> 29)** and the **NOM** by **3 points (17 -> 14).**

Class: Positions

Metric	Metrics Set	Description	Value	Regular Ra...
CHVL	Halstead...	Halstead Volume	735.9083	
CHD	Halstead...	Halstead Difficulty	26.8571	
CHL	Halstead...	Halstead Length	134	
CHEF	Halstead...	Halstead Effort	19764.3...	
CHVC	Halstead...	Halstead Vocabulary	45	
CHER	Halstead...	Halstead Errors	0.2437	
WMC	Chidamb...	Weighted Methods Per Class	29	[0..12)
DIT	Chidamb...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamb...	Response For A Class	14	[0..45)
LCOM	Chidamb...	Lack Of Cohesion Of Methods	1	
NOC	Chidamb...	Number Of Children	0	[0..2)
NOA	Lorenz-...	Number Of Attributes	5	[0..4)
NOO	Lorenz-...	Number Of Operations	27	
NOOM	Lorenz-...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-...	Number Of Added Methods	12	
SIZE2	Li-Henry...	Number Of Attributes And Methods	31	
NOM	Li-Henry...	Number Of Methods	14	[0..7)
MPC	Li-Henry...	Message Passing Coupling	15	
DAC	Li-Henry...	Data Abstraction Coupling	1	
NCSS	Chr. Cle...	Non-Commenting Source Statements	40	[0..1000)
CMI	Maintain...	Maintainability Index	38.2032	[0.0..19.0]

We thought about improving the **Number of Attributes metric** as well, even if it is just above the limit (5 instead of 4), but integrating all types of positions into a list-type attribute would have had a major impact on the logic of the project, would have created problems in the serialization process and would have decreased the efficiency of lookups, so we did not implement that idea in the end.

5. Activity in activity subsystem

The Activity is is in charge of encapsulating all the information about competitions and trainings. Initially, the **MetricsTree** pointed out the following irregularities:

Class: Activity

Metric	Metrics Set	Description	Value	Regular Ra...
CHVL	Halstead ...	Halstead Volume	1246.5019	
CHD	Halstead ...	Halstead Difficulty	41.9535	
CHL	Halstead ...	Halstead Length	195	
CHEF	Halstead ...	Halstead Effort	52295.10...	
CHVC	Halstead ...	Halstead Vocabulary	84	
CHER	Halstead ...	Halstead Errors	0.4661	
WMC	Chidamber-Kemerer Metrics Set	er Class	68	[0..12)
DIT	Chidamb...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamb...	Response For A Class	39	[0..45)
LCOM	Chidamb...	Lack Of Cohesion Of Methods	3	
NOC	Chidamb...	Number Of Children	0	[0..2)
NOA	Lorenz-K...	Number Of Attributes	10	[0..4)
NOO	Lorenz-K...	Number Of Operations	36	
NOOM	Lorenz-K...	Number Of Overridden Methods	2	[0..3)
NOAM	Lorenz-K...	Number Of Added Methods	19	
SIZE2	Li-Henry ...	Number Of Attributes And Methods	45	
NOM	Li-Henry ...	Number Of Methods	23	[0..7)
MPC	Li-Henry ...	Message Passing Coupling	33	
DAC	Li-Henry ...	Data Abstraction Coupling	5	
NCSS	Chr. Cle...	Non-Commenting Source Statements	36	[0..1000)
CMI	Maintain...	Maintainability Index	30.942	[0.0..19.0]

Since **WMC** is the only metric with an extreme value, it was the main priority. To reduce it we deleted the unused *equals and hashCode methods*, which largely accounted for the combined complexity. The final **WMC** measures at **36**.

Regarding the **Number of Attributes** (metric which is naturally connected with the further discussed one), the Activity class already has two embedded classes, namely, Positions and Timeslot that encapsulate some related attributes. Creating a new one would mean grouping less related fields together. Furthermore, it would mean significant changes in the business logic, which would complicate the refactor and cause serialization problems. As a consequence, we decided that 10 attributes is adequate in our situation. Especially given that two of them are static lists.

Note *that in the beginning the Activity class did not have the Positions and Timeslot embedded classes and had all the attributes as a list, part of the big class. However, we did not approve the merge request, we refactored that version and decided to improve the implementation by encapsulating some of the attributes in separate classes for maintainability and testability. (Naturally this also gave us a better start for the refactoring stage – see this merge request for reference* https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM30b/-/merge_requests/12 *)*

Since there quite a few attributes, which we need to store an activity, their autogenerated accessor methods inflate this number and also, there are a few methods that are essential for this class and we decided to keep them where they belong. That being said, the final **Number of Methods** measures at **22**. The final class metrics were also impacted by the refactor of the checkIfValid method, which reduced the overall cyclomatic complexity but added two simple methods.

Class: Activity

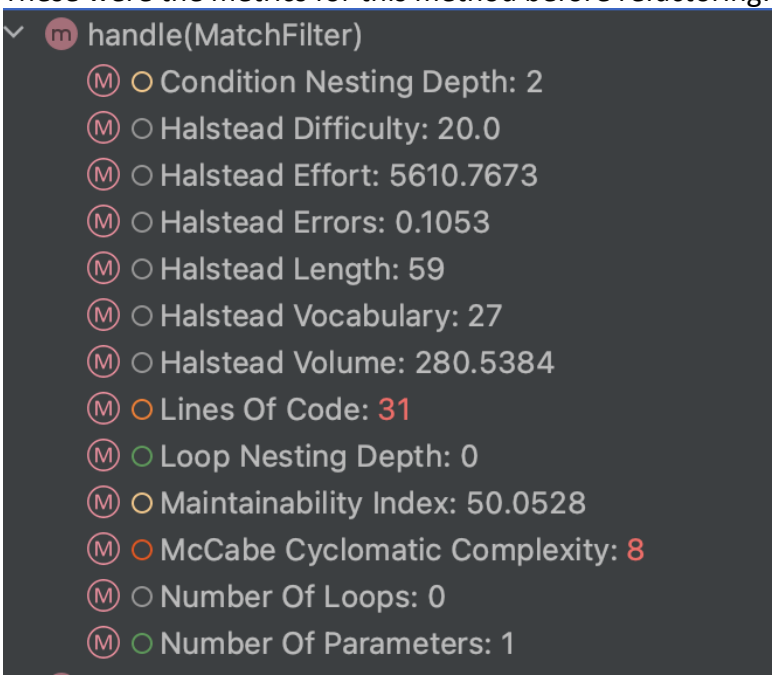
Metric	Metrics Set	Description	Value	Regular R
CHVL	Halstead M...	Halstead Volume	909.9134	
CHD	Halstead M...	Halstead Difficulty	32.9697	
CHL	Halstead M...	Halstead Length	150	
CHEF	Halstead M...	Halstead Effort	29999.5684	
CHVC	Halstead M...	Halstead Vocabulary	67	
CHER	Halstead M...	Halstead Errors	0.3218	
WMC	Chidamber...	Weighted Methods Per Class	36	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	28	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	2	
NOC	Chidamber...	Number Of Children	0	[0..2)
NOA	Lorenz-Kid...	Number Of Attributes	10	[0..4)
NOO	Lorenz-Kid...	Number Of Operations	35	
NOOM	Lorenz-Kid...	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Kid...	Number Of Added Methods	20	
SIZE2	Li-Henry M...	Number Of Attributes And Methods	44	
NOM	Li-Henry M...	Number Of Methods	22	[0..7)
MPC	Li-Henry M...	Message Passing Coupling	24	
DAC	Li-Henry M...	Data Abstraction Coupling	5	
NCSS	Chr. Cleme...	Non-Commenting Source Statements	26	[0..1000)
CMI	Maintainab...	Maintainability Index	36.7138	[0.0..19.0]

While refactoring this class we made sure that we acknowledge the information given by the metrics, but also tried to watch out for the thin line of overengineering the class.

METHODS:

1. **handle in TimeConstraintHandler of matching subsystem**

These were the metrics for this method before refactoring:



This method represents the **Long Method code smell**, having too many lines of code and we can also see that it has a **high cyclomatic complexity** as it contains a switch statement with multiple if statements.

```
public boolean handle(MatchFilter matchFilter) {
    switch (matchFilter.getActivityApp().getType()) {
        case TRAINING: {
            if (LocalDateTime.now().plusMinutes(30)
                    .isBefore(matchFilter.getActivityApp().getTimeslot().getStartTime())) {
                if (next != null) {
                    return next.handle(matchFilter);
                } else {
                    return true;
                }
            } else {
                return false;
            }
        }
        case COMPETITION: {
            if (LocalDateTime.now().plusDays(1)
                    .isBefore(matchFilter.getActivityApp().getTimeslot().getStartTime())) {
                if (next != null) {
                    return next.handle(matchFilter);
                } else {
                    return true;
                }
            } else {
                return false;
            }
        }
        default:
            return false;
    }
}
```

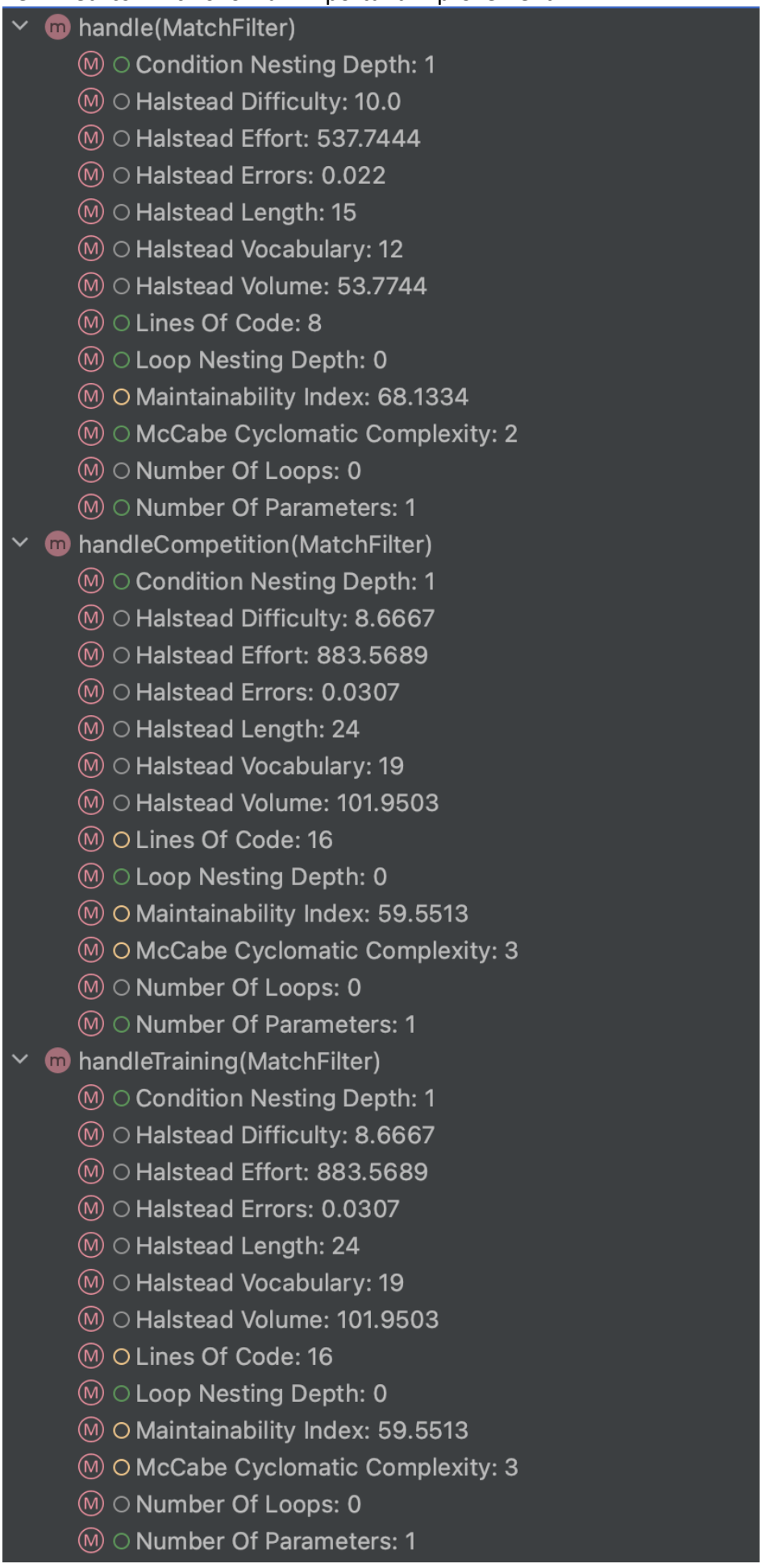
Thus we have applied **Extract Method Refactoring** and created 2 new methods, each handling a different type of activity.

```java
public boolean handleTraining(MatchFilter matchFilter) {
    if (!LocalDateTime.now().plusMinutes(30)
            .isBefore(matchFilter.getActivityApp().getTimeslot().getStartTime())) {
        return false;
    }

    if (next != null) {
        return next.handle(matchFilter);
    }

    return true;
}

/**
 * Method for handling the competition.
 *
 * @param matchFilter the MatchFilter entity containing info to be filtered on constraints
 * @return TRUE if match filter complies with the constraints, FALSE otherwise
 */
Micloiu Diana +1
public boolean handleCompetition(MatchFilter matchFilter) {
    if (!LocalDateTime.now().plusDays(1)
            .isBefore(matchFilter.getActivityApp().getTimeslot().getStartTime())) {
        return false;
    }
    if (next != null) {
        return next.handle(matchFilter);
    }

    return true;
}
```

Using these new methods we can create a more simple handle() method:

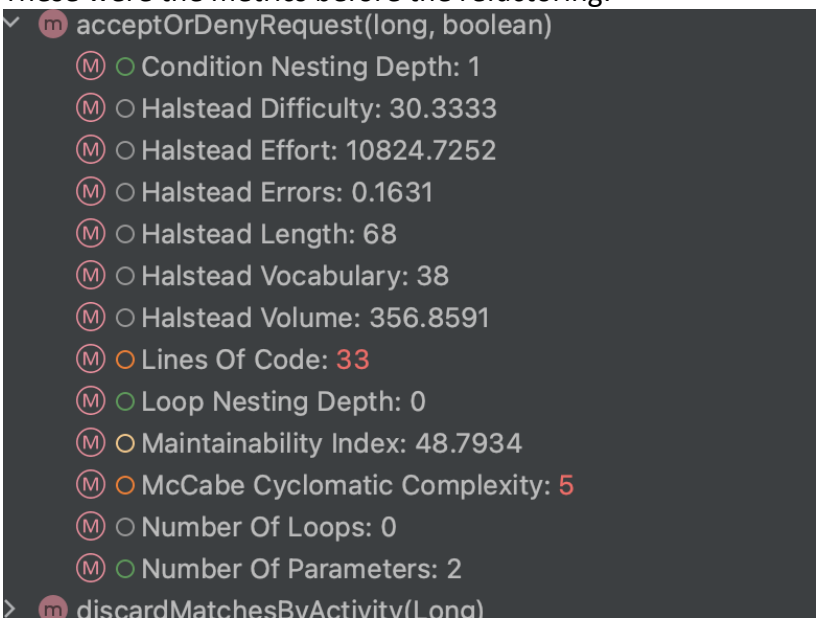
```java
public boolean handle(MatchFilter matchFilter) {
    if (matchFilter.getActivityApp().getType() == TypeOfActivity.TRAINING) {
        return handleTraining(matchFilter);
    } else {
        return handleCompetition(matchFilter);
    }
}
```

This decreases the cyclomatic complexity significantly and also reduces the lines of code, resulting in these new metrics which show an important improvement:



2. acceptOrDenyRequest in MatchingService class of matching subsystem

These were the metrics before the refactoring:



Here again we can notice the **Long Method smell** with **high LOC** and **CC**.

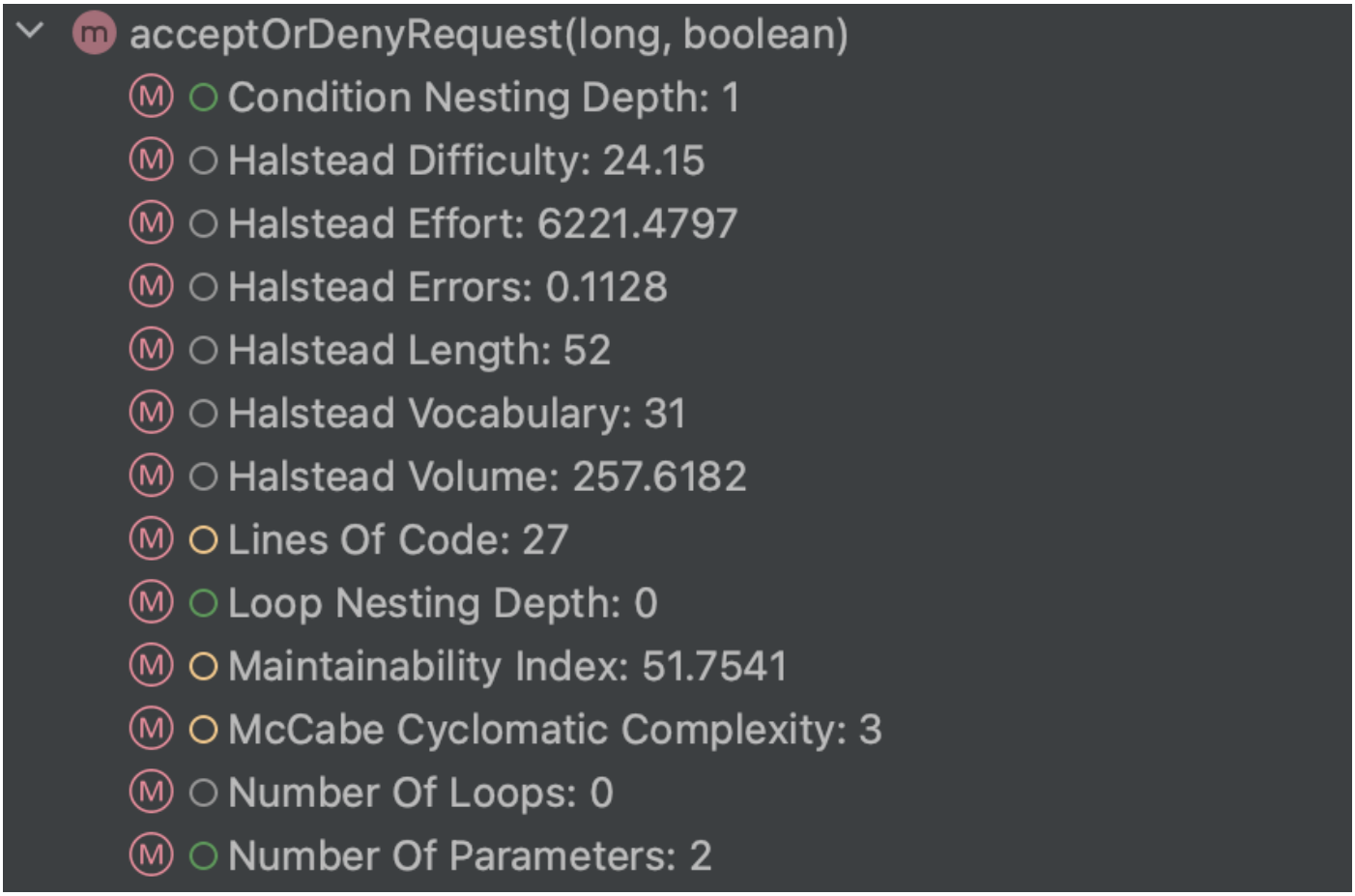
```java
public boolean acceptOrDenyRequest(long matchId, boolean decision) {
    Optional<Match> match = matchingRepo.getMatchByMatchId(matchId);
    if (match.isEmpty()) {
        return false;
    }
    if (!match.get().getOwnerId().equals(auth.getUserId())) {
        return false;
    }
    if (!match.get().getStatus().equals(Status.PENDING)) {
        return false;
    }
    Match newMatch = match.get();
    if (decision) {
        newMatch.setStatus(Status.ACCEPTED);
        activityCommunication.updateActivity(newMatch.getActivityId(), newMatch.getPosition());
    } else {
        newMatch.setStatus(Status.DECLINED);
    }
    matchingRepo.save(newMatch);
    notificationCommunication.sendNotificationToParticipant(
            new NotificationRequestModelParticipant(newMatch.getParticipantId(),
                    newMatch.getActivityId(),
                    activityCommunication.getActivityTimeslotById(newMatch.getActivityId()),
                    decision));
    return true;
}
```

We used too many if statements so we applied the **Extract method refactoring** by removing the first verification conditions in the beginning in a separate method (verifySubmission()) which verifies if the match is valid.

```java
public boolean verifySubmission(Optional<Match> match) {
    return match.isPresent()
            && match.get().getActivityInformation().getOwnerId().equals(auth.getUserId())
            && match.get().getStatus().equals(Status.PENDING);
}
```

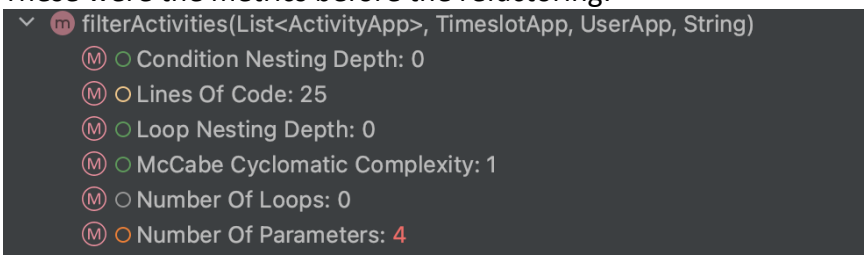
```java
public boolean acceptOrDenyRequest(long matchId, boolean decision) {
    Optional<Match> match = matchingRepo.getMatchByMatchId(matchId);
    if (!verifySubmission(match)) {
        return false;
    }

    Match newMatch = matchingRepo.getMatchByMatchId(matchId).get();

    if (decision) {
        newMatch.setStatus(Status.ACCEPTED);
        communication.getActivityCommunication().updateActivity(newMatch.getActivityInformation().getActivityId(),
                newMatch.getActivityInformation().getPosition());
    } else {
        newMatch.setStatus(Status.DECLINED);
    }
    matchingRepo.save(newMatch);
    communication.getNotificationCommunication().sendNotificationToParticipant(
            new NotificationRequestModelParticipant(newMatch.getParticipantId(),
                    newMatch.getActivityInformation().getActivityId(),
                    communication.getActivityCommunication()
                            .getActivityTimeslotById(newMatch.getActivityInformation().getActivityId()),
                    decision));
    return true;
}
```

These are the metrics after this refactoring, and we can see that the metrics have greatly improved.



3. filterActivities in MatchingService class of matching subsystem

These were the metrics before the refactoring:



This method uses **too many parameters** as we can see from the metrics.

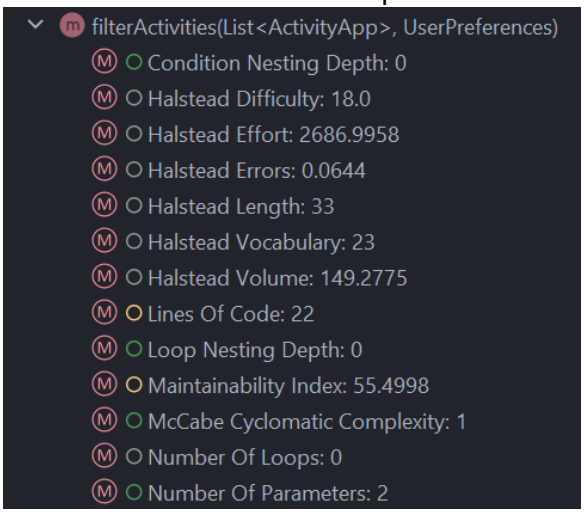
```
public List<ActivityReponse> filterActivities(List<ActivityApp> activities,
                                              TimeslotApp timeslot,
                                              UserApp user,
                                              String position) {
    return activities
            .stream() Stream<ActivityApp>
            .filter(a -> a != null)
            .distinct()
            .map(a -> a.setTypeOfActivity())
            .filter(a -> a != null)
            .filter(a -> this.filteringHandler.handle(new MatchFilter(a, user, position, timeslot)))
            .filter(a -> matchingRepo.getMatchesByActivityIdAndParticipantId(a.getId(), user.getEmail()).isEmpty())
            .map(a -> matchUserToActivity(user, position, a)) Stream<ActivityReponse>
            .collect(Collectors.toList());
}
```

So we decided to group the parameters in an **Object Parameter** containing the *timeslot, user and the position* as it makes sense to group them together in a **class UserPreferences**.

```
public List<ActivityReponse> filterActivities(List<ActivityApp> activities,
                                              UserPreferences userPreferences) {
    return activities
            .stream() Stream<ActivityApp>
            .filter(Objects::nonNull)
            .distinct()
            .map(ActivityApp::setTypeOfActivity)
            .filter(Objects::nonNull)
            .filter(a -> this.filteringHandler.handle(new MatchFilter(a, userPreferences)))
            .filter(a -> matchingRepo.getMatchesByActivityIdAndParticipantId(a.getId(), userPreferences.getUser()
                    .getEmail()).isEmpty())
            .map(a -> matchUserToActivity(userPreferences.getUser(), userPreferences.getPosition(), a)) Stream<ActivityReponse>
            .collect(Collectors.toList());
}
```

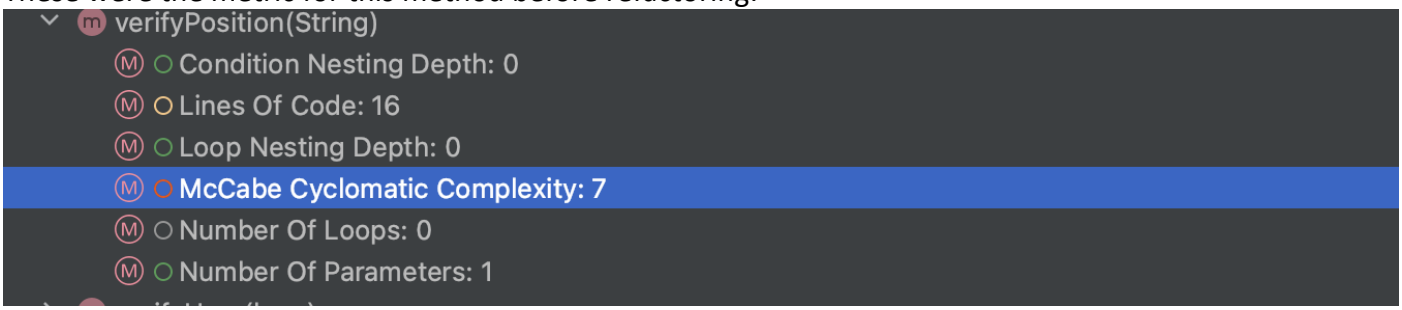
```
public class UserPreferences {

    private TimeslotApp timeslot;
    private UserApp user;
    private String position;
}
```

This results in fewer method parameters and an overall increase in readability.



4. verifyPosition in MatchingService class of matching subsystem

These were the metric for this method before refactoring:



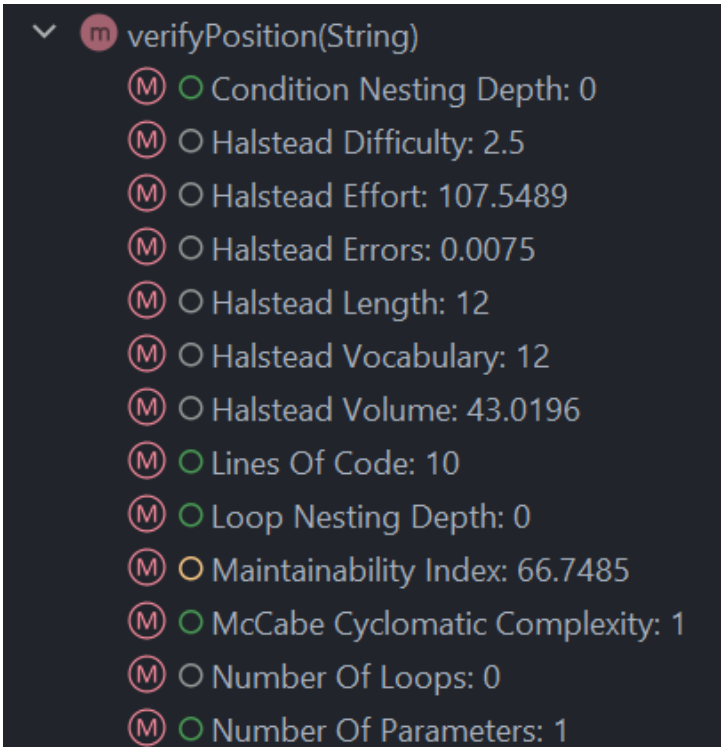
This method was using a **switch case** to check if a position is valid. However this increased the **cyclomatic complexity significantly.**

```
public boolean verifyPosition(String position) {
    switch (position) {
        case "cox" : return true;
        case "starboard" : return true;
        case "coach" : return true;
        case "port" : return true;
        case "sculling" : return true;
        default : return false;
    }
}
```

Thus we have decided to remove the switch case and use the contains() method in the List class.

```
public boolean verifyPosition(String position) {
    List<String> validPositions = List.of("cox", "port", "coach", "starboard", "sculling");
    return validPositions.contains(position);
}
```

This reduced the **cyclomatic complexity** to **1** and every metric for this method is now in the accepted threshold.



5. checkIfValid in Activity of activity microservice

These were the metric and implementation for this method before refactoring:


```java
public boolean checkIfValid() {
    boolean requiresRowers = positions.getCox() != null || positions.getCoach() != null
            || positions.getPort() != null || positions.getStarboard() != null
            || positions.getSculling() != null;
    boolean nonNull = ownerId != null && requiresRowers;
    if (!nonNull) {
        return false;
    }
    if (!CERTIFICATE_TYPES.contains(certificate)) {
        return false;
    }
    if (competition && ((gender == null || !GENDER_TYPES.contains(gender)) || organisation == null)) {
        return false;
    }
    LocalDateTime now = LocalDateTime.now();
    boolean timeslotExists = timeslot != null && timeslot.getStartTime() != null && timeslot.getEndTime() != null;
    return timeslotExists && timeslot.getStartTime().isBefore(timeslot.getEndTime())
            && timeslot.getEndTime().isAfter(now);
}
```

As you can see, the **checkIfValid method** was messy and complicated at first, so it was obvious that it could benefit from refactoring. The idea behind simplifying the method was to **divide it into smaller parts**. First of all, the feature envy for positions and timeslot attributes was removed by moving the related logic in the Positions and Timeslot classes. After that additional two methods were created in the Activity class that were both really concise and easy to test. The results were that the **cyclomatic complexity** was reduced from **16** to **4** as a tradeoff for creating more albeit really short methods. Meanwhile the **Lines of Code** decreased to only **9**.

Position class method extracted

```java
public boolean checkIfValid() {
    return cox != null || coach != null || port != null || starboard != null || sculling != null;
}
```

Timeslot class method extracted

```java
public boolean checkIfValid() {
    LocalDateTime now = LocalDateTime.now();
    return startTime != null && endTime != null && startTime.isBefore(endTime) && endTime.isAfter(now);
}
```

```java
public boolean checkIfValid() {
    boolean valid = ownerId != null && CERTIFICATE_TYPES.contains(certificate);
    return valid && checkIfPositionsAndTimeslotValid() && checkIfCompetitionValid();
}

/**
 * Checks if the positions and timeslot have valid data.
 *
 * @return true if the data is valid and false otherwise
 */
public boolean checkIfPositionsAndTimeslotValid() {
    return positions != null && positions.checkIfValid() && timeslot != null && timeslot.checkIfValid();
}

/**
 * Checks if this activity is a competition that is valid.
 *
 * @return true if this is a valid competition or not a competition at all and false otherwise
 */
public boolean checkIfCompetitionValid() {
    return !competition || ((gender == null || GENDER_TYPES.contains(gender)) && organisation != null);
}
```

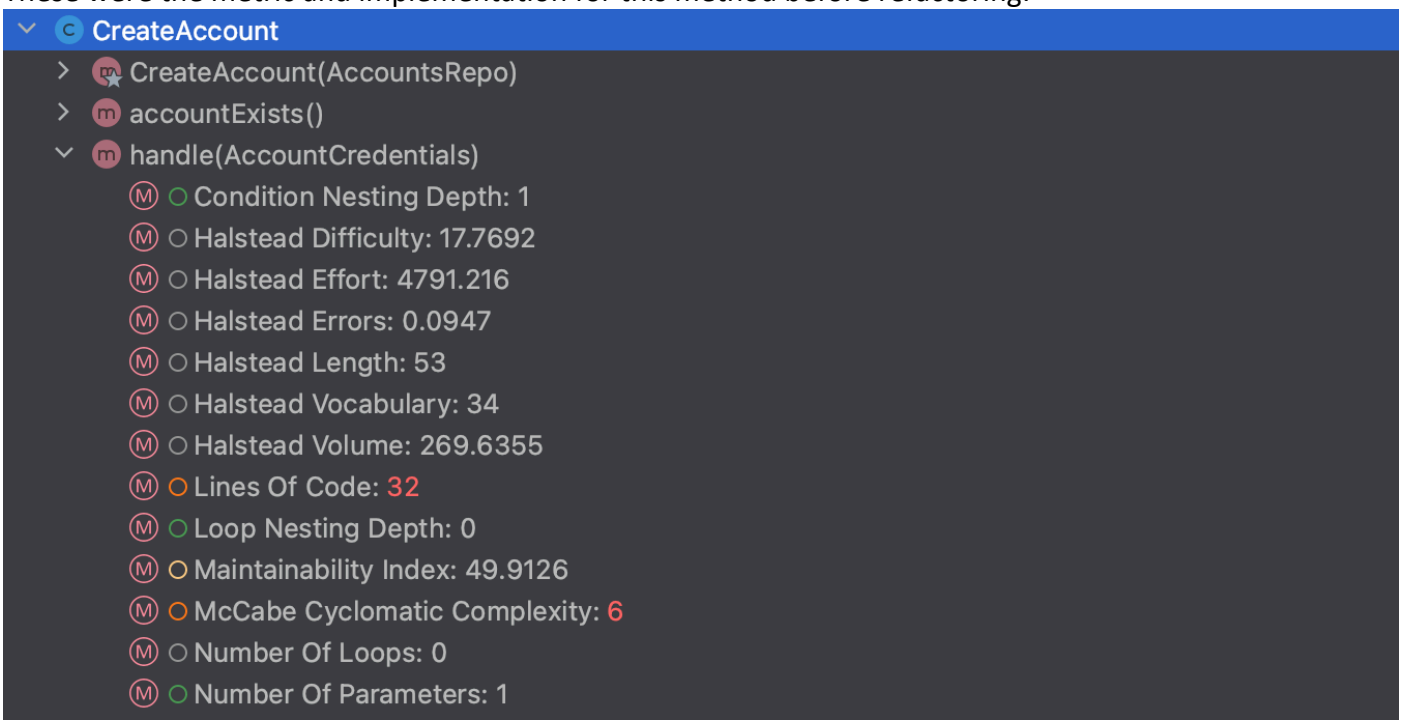
Consequently, the metrics show a real improvement in the end:

Method: checkIfValid()

Metric	Metrics Set	Description	Value	Regular Ra...
CND		Condition Nesting Depth	0	[0..2)
LND		Loop Nesting Depth	0	[0..2)
CC		McCabe Cyclomatic Complexity	4	[0..3)
NOL		Number Of Loops	0	
LOC		Lines Of Code	9	[0..11)
NOPM		Number Of Parameters	0	[0..3)
HVL	Halstead M...	Halstead Volume	51.8062	
HD	Halstead M...	Halstead Difficulty	4.0	
HL	Halstead M...	Halstead Length	14	
HEF	Halstead M...	Halstead Effort	207.2246	
HVC	Halstead M...	Halstead Vocabulary	13	
HER	Halstead M...	Halstead Errors	0.0117	
MMI	Maintainab...	Maintainability Index	67.0413	[0.0..19.0]

6. handle in CreateAccount handler in authentication subsystem

These were the metric and implementation for this method before refactoring:




```java
@Override
public void handle(AccountCredentials credentials) {
    if (next == null || exceptionHandler == null) {
        return;
    }
    this.credentials = credentials;
    try {
        if (accountExists()) {
            exceptionHandler.handleException(new SQLException(), "An account with this user id already exists."
                    + " Please choose a different user id.", 400);
            return;
        }
        String hashedPassword = hashPassword(credentials.getPassword());
        AccountCredentials hashedCredentials = new AccountCredentials(credentials.getUserId(), hashedPassword);
        accountsRepo.save(hashedCredentials);
        if (!verifySavedAccount(credentials.getPassword())) {
            exceptionHandler.handleException(new SQLException(), "There was an error while saving your account."
                    + " Please try again later", 500);
            return;
        }
        next.handle(credentials);
    } catch (Exception e) {
        exceptionHandler.handleException(e);
    }
}
```

The handle method in **CreateAccount** puts a new account in the database and conducts checks to test if this is possible and if the insertion was successful. If one of these checks fail, it sets the **ExceptionHandler** and returns. Because of the many checks, the method is **too long** and has a **cyclomatic complexity** that is **too high.**

To reduce the **number of lines**, all interactions with the ExceptionHandler were moved to the methods that perform the actual checks. These methods were quite short, thus even with the addition of this responsibility, they all still score in the acceptable range for all metrics.

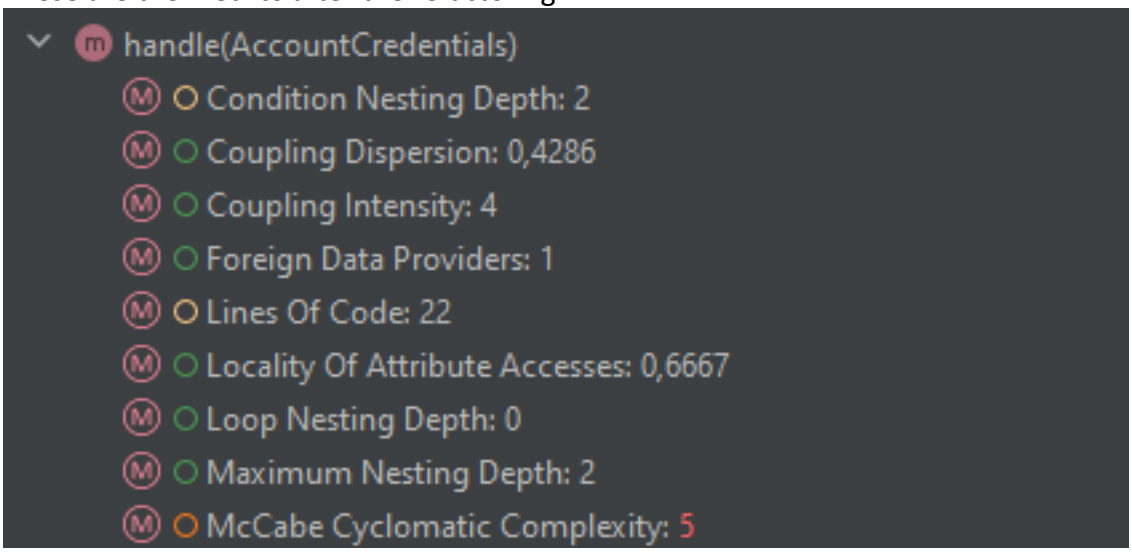
To reduce the **method length** even further, the checks were **changed from 'catch clause form' to a nested form**. This allows for the removal of the return statements.

To reduce the **cyclomatic complexity**, the try-catch block was removed. This was possible because it is highly unlikely that the methods called by handle would throw an exception, and thus the block was somewhat redundant.

After these refactors, the methods looks as follows:

```
@Override
public void handle(AccountCredentials credentials) {
    if (next == null || exceptionHandler == null) {
        return;
    }
    this.credentials = credentials;
    if (!accountExists()) {
        String hashedPassword = hashPassword(credentials.getPassword());
        AccountCredentials hashedCredentials = new AccountCredentials(credentials.getUserId(), hashedPassword);
        accountsRepo.save(hashedCredentials);
        if (verifySavedAccount(credentials.getPassword())) {
            next.handle(credentials);
        }
    }
}
```

These are the metrics after the refactoring:

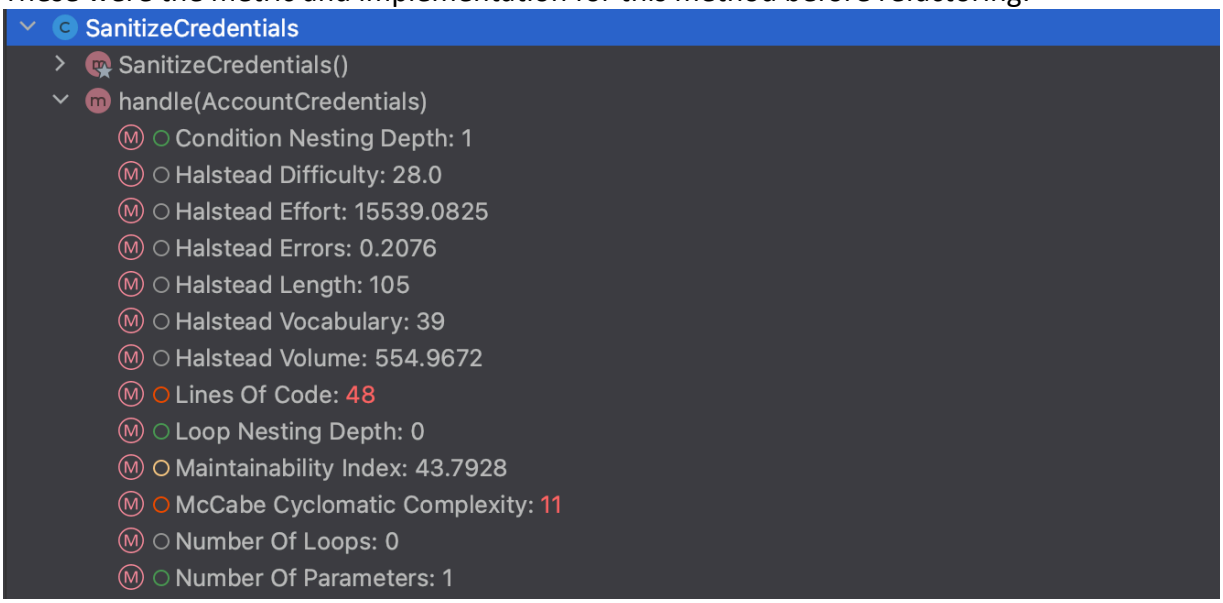


The improvements on the insufficient metrics are **LOC** from **32** to **22** and **CC** from **6** to **5.**

Note that MetricsTree still marks a CC of 5 as orange. Although this is still a bit high, it is felt that any further refactors are inapposite, because they would take away the purpose of the method: To check if the Handler is properly configured and an account can be successfully created.

7. handle in SanitizeCredentials handler in authentication subsystem

These were the metric and implementation for this method before refactoring:




```java
@Override
public void handle(AccountCredentials credentials) {
    if (next == null || exceptionHandler == null) {
        return;
    }
    this.credentials = credentials;
    try {
        if (credentials.getUserId() == null || credentials.getUserId().equals("")) {
            exceptionHandler.handleException(new IllegalArgumentException(), "Please provide a user id.", 400);
            return;
        }
        if (credentials.getPassword() == null || credentials.getPassword().equals("")) {
            exceptionHandler.handleException(new IllegalArgumentException(), "Please provide a password.", 400);
            return;
        }
        sanitizedUserId = sanitize(credentials.getUserId());
        sanitizedPassword = sanitize(credentials.getPassword());
        if (sanitizedUserId.length() < credentials.getUserId().length()) {
            exceptionHandler.handleException(new IllegalArgumentException(), "Your user id contains illegal"
                    + " characters. Please only use letters, numbers and the following characters: "
                    + "!#$%&()*+,-./:;<=>?@^_`{|}~", 400);
            return;
        }
        if (sanitizedPassword.length() < credentials.getPassword().length()) {
            exceptionHandler.handleException(new IllegalArgumentException(), "Your password contains illegal"
                    + " characters. Please only use letters, numbers and the following characters: "
                    + "!#$%&()*+,-./:;<=>?@^_`{|}~", 400);
            return;
        }
        if (!isEmailAddress(sanitizedUserId)) {
            exceptionHandler.handleException(new IllegalArgumentException(), "Your user id must be an email address.",
                    400);
            return;
        }
        AccountCredentials newCredentials = new AccountCredentials(sanitizedUserId, sanitizedPassword);
        next.handle(newCredentials);
    } catch (Exception e) {
        exceptionHandler.handleException(e);
    }
}
```

The **handle method in SanitizeCredentials** checks if the correct credentials are provided and if they are clean enough to be further processed by the system. It performs many different checks and sets the appropriate error message in the **ExceptionHandler** if the credentials are deemed insufficient. Because of the many checks and their accompanying interactions with the **ExceptionHandler**, the method is **extremely long** and has a **high cyclomatic complexity.**

To reduce the **number of lines and the cyclomatic complexity**, the checks were split into two parts: presence and cleanness. For these two parts, two additional methods were created: *credentialsNotEmpty and credentialsClean*. These two methods together contain all the checks that the handler contained before. They return a boolean based on if all the checks pass. Now the handle method only has to check if these two methods return true before continuing.

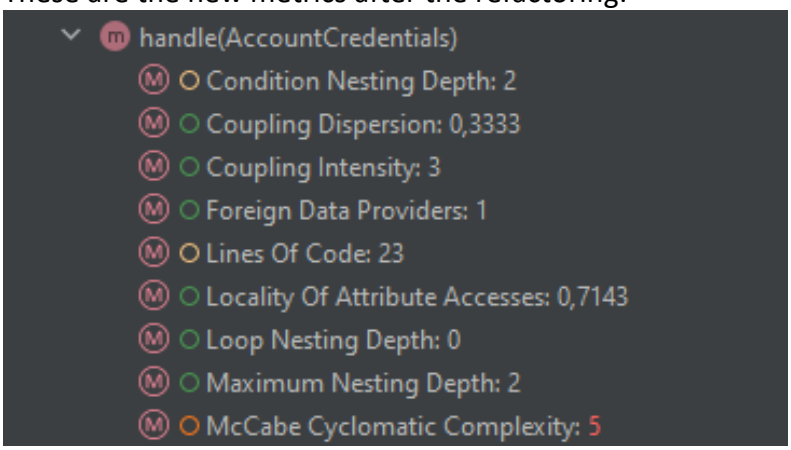
To reduce the **cyclomatic complexity** even further, the try-catch block was removed. This was possible because it is highly unlikely that the methods called by handle would throw an exception, and thus the block was somewhat redundant.

To reduce the **method length** even further, the checks were **changed from 'catch clause form' to a nested form**. This allows for the removal of the return statements.

After these refactors, the methods looks as follows:

```
@Override
public void handle(AccountCredentials credentials) {
    if (next == null || exceptionHandler == null) {
        return;
    }
    this.credentials = credentials;
    if (credentialsNotEmpty()) {
        sanitizedUserId = sanitize(credentials.getUserId());
        sanitizedPassword = sanitize(credentials.getPassword());
        if (credentialsClean()) {
            AccountCredentials newCredentials = new AccountCredentials(sanitizedUserId, sanitizedPassword);
            next.handle(newCredentials);
        }
    }
}
```

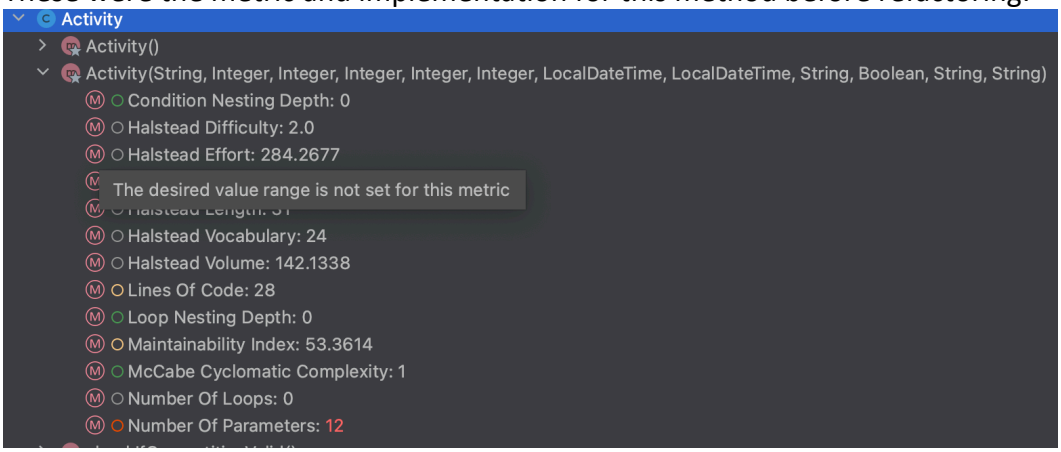
These are the new metrics after the refactoring:



The improvements on the insufficient metrics are **LOC** from **48** to **23** and **CC** from **11** to **6.**
Note that MetricsTree still marks a CC of 5 as orange. Although this is still a bit high, it is felt that any further refactors are inapposite, because they would take away the purpose of the method: To check if the Handler is properly configured and the provided credentials are clean enough to be processed further.

8. constructor in Activity class of activity microservice

These were the metric and implementation for this method before refactoring:




```java
public Activity(String ownerId, Integer coxCount, Integer coachCount, Integer portSideRowerCount,
                Integer starboardSideRowerCount, Integer scullingRowerCount, LocalDateTime startTime,
                LocalDateTime endTime, String certificate, Boolean competition, String gender, String organisation) {
    this.ownerId = ownerId;
    this.positions = new Positions(coxCount, coachCount, portSideRowerCount, starboardSideRowerCount,
            scullingRowerCount);
    this.timeslot = new Timeslot(startTime, endTime);
    this.certificate = certificate;
    this.competition = Objects.requireNonNullElse(competition, defaultObj: false);
    this.gender = gender;
    this.organisation = organisation;
}
```

The constructor for this class was created before the embedded objects (Positions and Timeslot) were introduced. Thus, it had 12 parameters which were used to initiate the class. However, the fact that we had introduced the objects quite early in the development allowed us to refactor the constructor easily without any problems while deserializing the class. The parameters that were used to create the embedded classes were replaced by the objects themselves, this way reducing **the number of parameters** down to **7** and **lines of code** to **21**.

ethod: Activity(String, Positions, Timeslot, String, Boolean, String, String)

Metric	Metrics Set	Description	Value	Regular Ra...
CND		Condition Nesting Depth	0	[0..2)
LND		Loop Nesting Depth	0	[0..2)
CC		McCabe Cyclomatic Complexity	1	[0..3)
NOL		Number Of Loops	0	
LOC		Lines Of Code	21	[0..11)
NOPM		Number Of Parameters	7	[0..3)
HVL	Halstead M...	Halstead Volume	100.0782	
HD	Halstead M...	Halstead Difficulty	1.5	
HL	Halstead M...	Halstead Length	24	
HEF	Halstead M...	Halstead Effort	150.1173	
HVC	Halstead M...	Halstead Vocabulary	18	
HER	Halstead M...	Halstead Errors	0.0094	
MMI	Maintainab...	Maintainability Index	57.1531	[0.0..19.0]

```
public Activity(String ownerId, Positions positions, Timeslot timeslot, String certificate,
                Boolean competition, String gender, String organisation) {
    this.ownerId = ownerId;
    this.positions = positions;
    this.timeslot = timeslot;
    this.certificate = certificate;
    this.competition = Objects.requireNonNullElse(competition, defaultObj: false);
    this.gender = gender;
    this.organisation = organisation;
}
```

Note that although the 7 parameters are marked as red, we must keep in mind that the number of parameters depends on the number of attributes. The reasoning behind keeping the attribute count is explained in the Activity class refactoring.