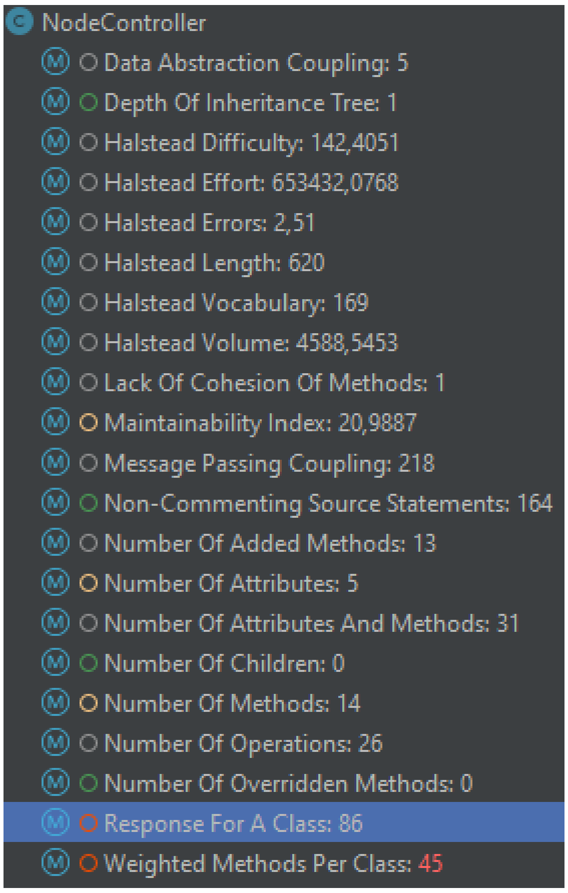
Group-DB-09b-Assignment-2

Refactored Classes:
1. NodeController
2. JobService
3. JobController
4. Scheduler-microservice/domain/processing/ProcessingJobsService
5. Scheduler-microservice/domain/processing/UpdatingJobsService

Refactored Methods:
1. NodeController/deleteNode
2. NodeController/addNode
3. NodeController/ReleaseFaculty
4. JobService/createJob()
5. ProcessingJobsService/scheduleJob()

Class: NodeController

First of all, the NodeController class was very smelly. It wasn't separated into a service based model. There was a lot of “blob code” and many change preventers along with dispensable methods.

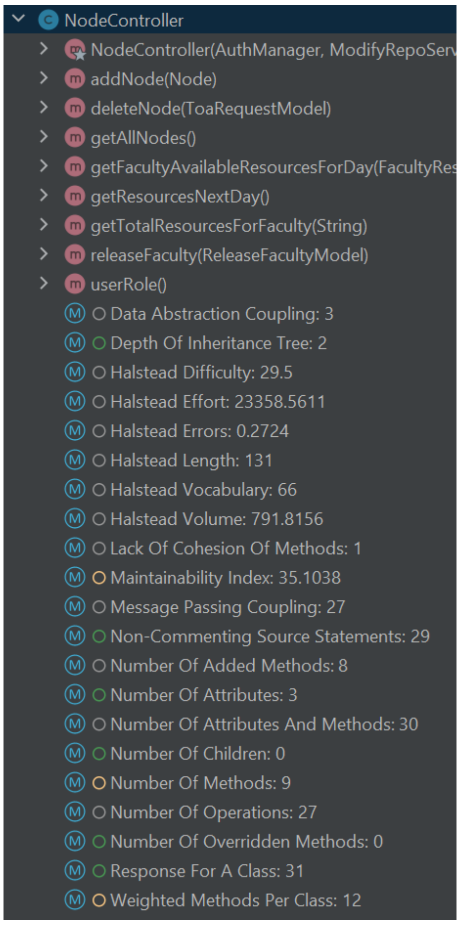


```
NodeController
  Data Abstraction Coupling: 5
  Depth Of Inheritance Tree: 1
  Halstead Difficulty: 142,4051
  Halstead Effort: 653432,0768
  Halstead Errors: 2,51
  Halstead Length: 620
  Halstead Vocabulary: 169
  Halstead Volume: 4588,5453
  Lack Of Cohesion Of Methods: 1
  Maintainability Index: 20,9887
  Message Passing Coupling: 218
  Non-Commenting Source Statements: 164
  Number Of Added Methods: 13
  Number Of Attributes: 5
  Number Of Attributes And Methods: 31
  Number Of Children: 0
  Number Of Methods: 14
  Number Of Operations: 26
  Number Of Overridden Methods: 0
  Response For A Class: 86
  Weighted Methods Per Class: 45
```

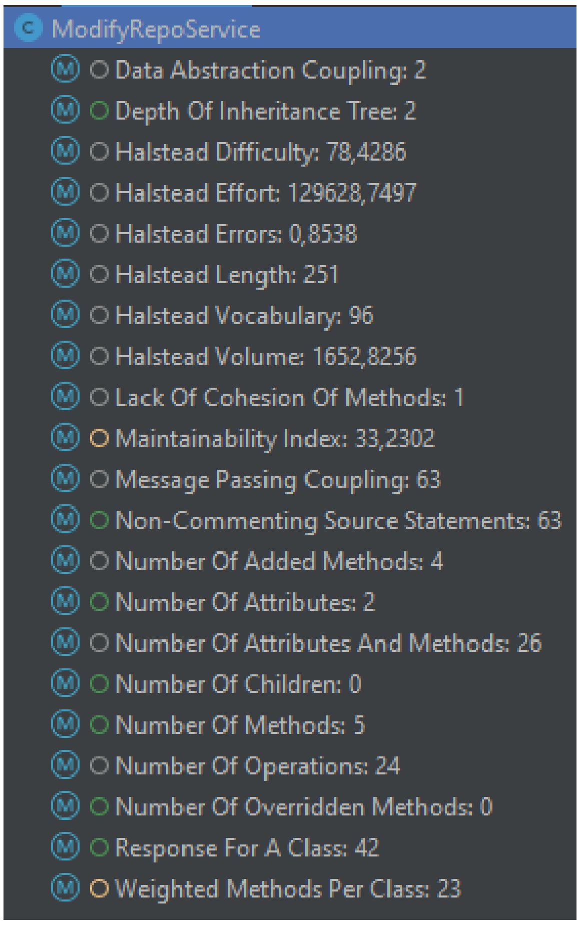
To fix this we first extracted classes and separated it into classes that would have similar functionality. We added ModifyRepoService class and GetResourceService class and extracted the main functionality of the methods from the controller to the respective classes. This allowed the NodeController to decrease in size and became more maintainable as changes to the logic happened in their respective classes and methods. This removed many change preventers in the code and made further refactoring much easier. Furthermore, a class CheckHelper was introduced in order to contain all the if-else checks of the NodeController and simplify its logic.

NodeController class had too many methods, we were able to cut it down by removing several methods that were not being used as they were only there to manually test in the development stage.

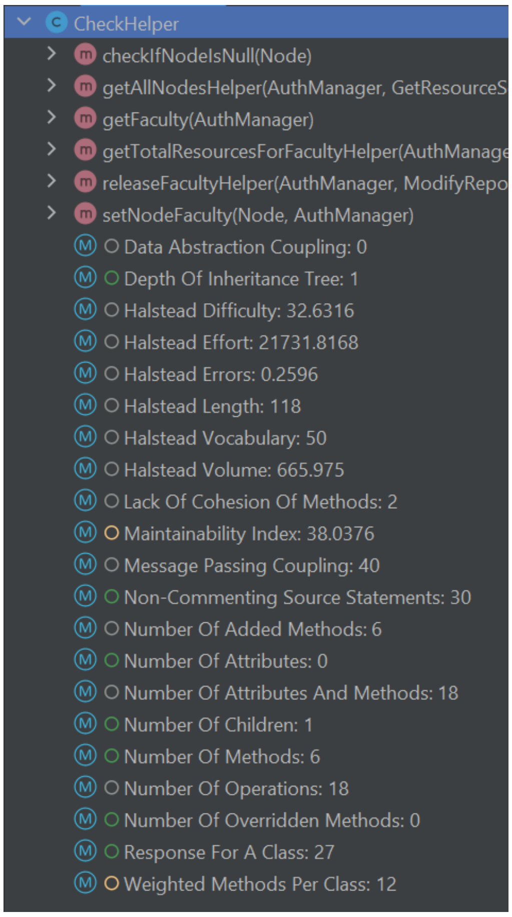
The NodeController class is now split into 4 classes with acceptable metrics. The number of attributes decreased from 5 to 3, number of methods from 14 to 9, response for a class from 86 to 31 and weighted method from 45 to 12 while the new classes also have mostly green metrics.



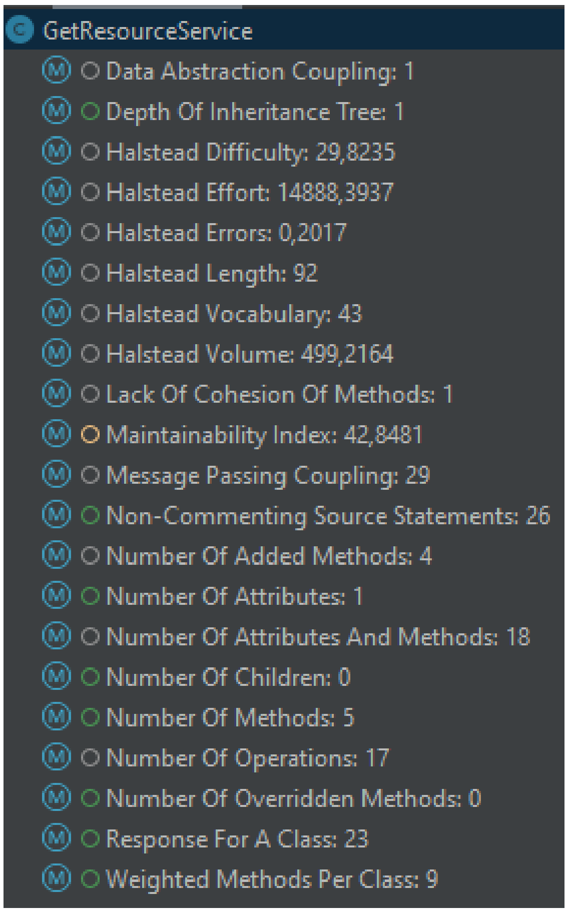
```
NodeController
  NodeController(AuthManager, ModifyRepoServ
  addNode(Node)
  deleteNode(ToaRequestModel)
  getAllNodes()
  getFacultyAvailableResourcesForDay(FacultyRes
  getResourcesNextDay()
  getTotalResourcesForFaculty(String)
  releaseFaculty(ReleaseFacultyModel)
  userRole()
  Data Abstraction Coupling: 3
  Depth Of Inheritance Tree: 2
  Halstead Difficulty: 29.5
  Halstead Effort: 23358.5611
  Halstead Errors: 0.2724
  Halstead Length: 131
  Halstead Vocabulary: 66
  Halstead Volume: 791.8156
  Lack Of Cohesion Of Methods: 1
  Maintainability Index: 35.1038
  Message Passing Coupling: 27
  Non-Commenting Source Statements: 29
  Number Of Added Methods: 8
  Number Of Attributes: 3
  Number Of Attributes And Methods: 30
  Number Of Children: 0
  Number Of Methods: 9
  Number Of Operations: 27
  Number Of Overridden Methods: 0
  Response For A Class: 31
  Weighted Methods Per Class: 12
```



```
ModifyRepoService
  Data Abstraction Coupling: 2
  Depth Of Inheritance Tree: 2
  Halstead Difficulty: 78,4286
  Halstead Effort: 129628,7497
  Halstead Errors: 0,8538
  Halstead Length: 251
  Halstead Vocabulary: 96
  Halstead Volume: 1652,8256
  Lack Of Cohesion Of Methods: 1
  Maintainability Index: 33,2302
  Message Passing Coupling: 63
  Non-Commenting Source Statements: 63
  Number Of Added Methods: 4
  Number Of Attributes: 2
  Number Of Attributes And Methods: 26
  Number Of Children: 0
  Number Of Methods: 5
  Number Of Operations: 24
  Number Of Overridden Methods: 0
  Response For A Class: 42
  Weighted Methods Per Class: 23
```



```
CheckHelper
  checkIfNodeIsNull(Node)
  getAllNodesHelper(AuthManager, GetResourceS
  getFaculty(AuthManager)
  getTotalResourcesForFacultyHelper(AuthManage
  releaseFacultyHelper(AuthManager, ModifyRepo
  setNodeFaculty(Node, AuthManager)
  Data Abstraction Coupling: 0
  Depth Of Inheritance Tree: 1
  Halstead Difficulty: 32.6316
  Halstead Effort: 21731.8168
  Halstead Errors: 0.2596
  Halstead Length: 118
  Halstead Vocabulary: 50
  Halstead Volume: 665.975
  Lack Of Cohesion Of Methods: 2
  Maintainability Index: 38.0376
  Message Passing Coupling: 40
  Non-Commenting Source Statements: 30
  Number Of Added Methods: 6
  Number Of Attributes: 0
  Number Of Attributes And Methods: 18
  Number Of Children: 1
  Number Of Methods: 6
  Number Of Operations: 18
  Number Of Overridden Methods: 0
  Response For A Class: 27
  Weighted Methods Per Class: 12
```

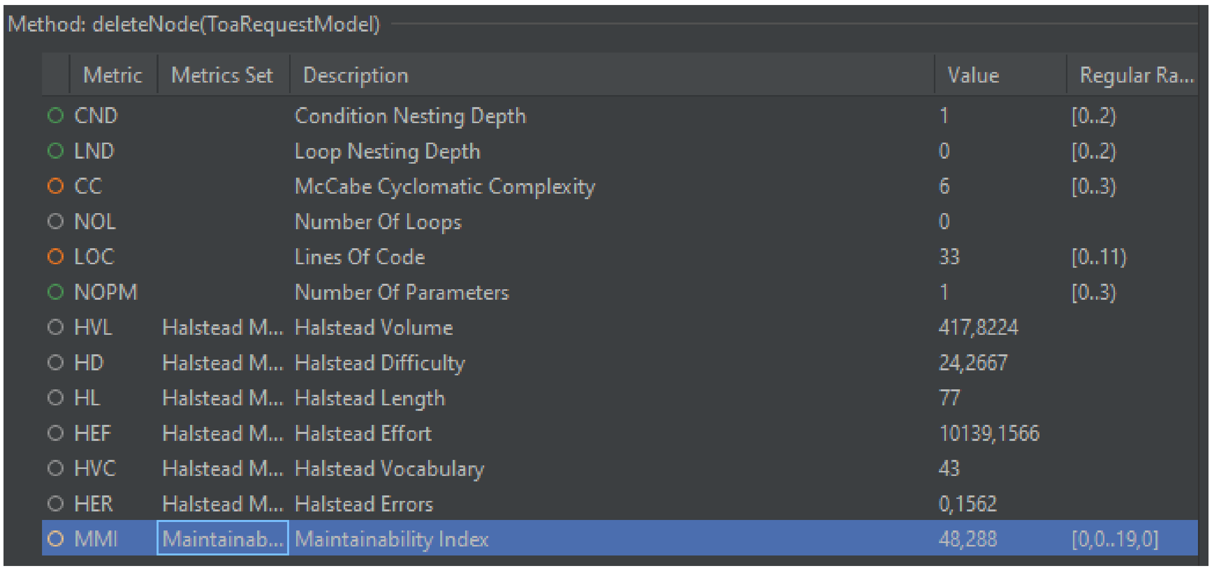


```
GetResourceService
  Data Abstraction Coupling: 1
  Depth Of Inheritance Tree: 1
  Halstead Difficulty: 29,8235
  Halstead Effort: 14888,3937
  Halstead Errors: 0,2017
  Halstead Length: 92
  Halstead Vocabulary: 43
  Halstead Volume: 499,2164
  Lack Of Cohesion Of Methods: 1
  Maintainability Index: 42,8481
  Message Passing Coupling: 29
  Non-Commenting Source Statements: 26
  Number Of Added Methods: 4
  Number Of Attributes: 1
  Number Of Attributes And Methods: 18
  Number Of Children: 0
  Number Of Methods: 5
  Number Of Operations: 17
  Number Of Overridden Methods: 0
  Response For A Class: 23
  Weighted Methods Per Class: 9
```

Method: NodeController/deleteNode

Introduction

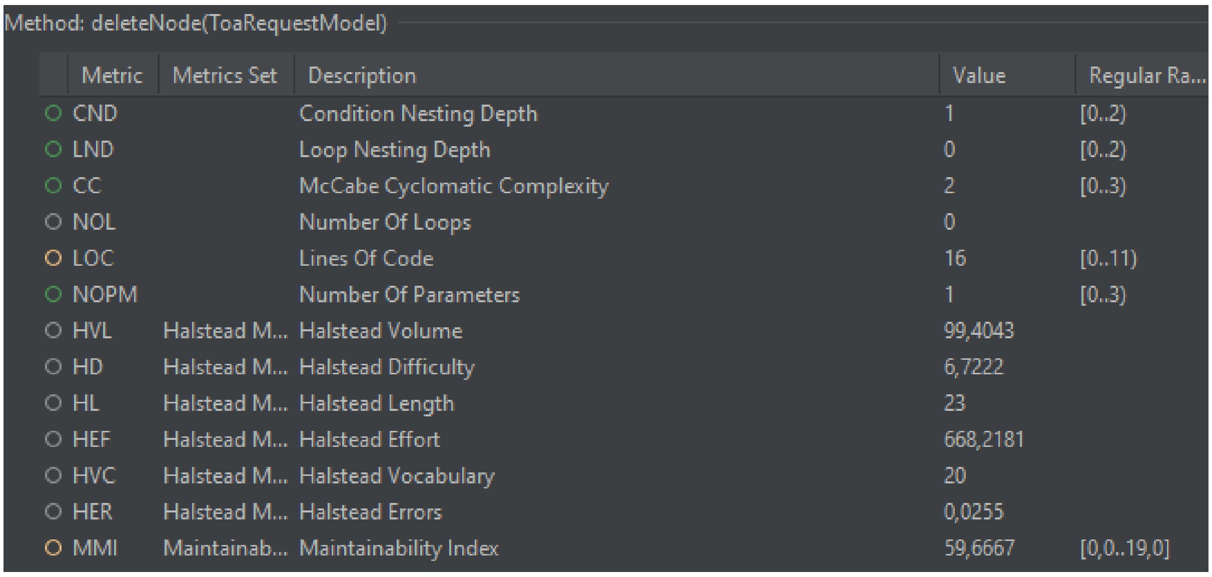
Here is the metrics before refactoring:



Method: deleteNode(ToaRequestModel)

	Metric	Metrics Set	Description	Value	Regular Ra...
○	CND		Condition Nesting Depth	1	[0..2)
○	LND		Loop Nesting Depth	0	[0..2)
○	CC		McCabe Cyclomatic Complexity	6	[0..3)
○	NOL		Number Of Loops	0	
○	LOC		Lines Of Code	33	[0..11)
○	NOPM		Number Of Parameters	1	[0..3)
○	HVL	Halstead M...	Halstead Volume	417,8224	
○	HD	Halstead M...	Halstead Difficulty	24,2667	
○	HL	Halstead M...	Halstead Length	77	
○	HEF	Halstead M...	Halstead Effort	10139,1566	
○	HVC	Halstead M...	Halstead Vocabulary	43	
○	HER	Halstead M...	Halstead Errors	0,1562	
○	MMI	Maintainab...	Maintainability Index	48,288	[0,0..19,0]

Before the changes, the cyclomatic complexity was 3 above normal. To fix that, we replaced a null chuck conditional statement with a method that was already created. This also reduced the number of lines of code. After the extraction of the functionality of the code into a separate method in a different class, the effective lines of code decreased to 8. The metric insists that the lines of code is 16, however that also includes the javadoc. We believe that the javadoc shouldn't be included in the LOC because usually the bigger the javadoc the better. The following image is the metrics after refactoring the deleteNode method:

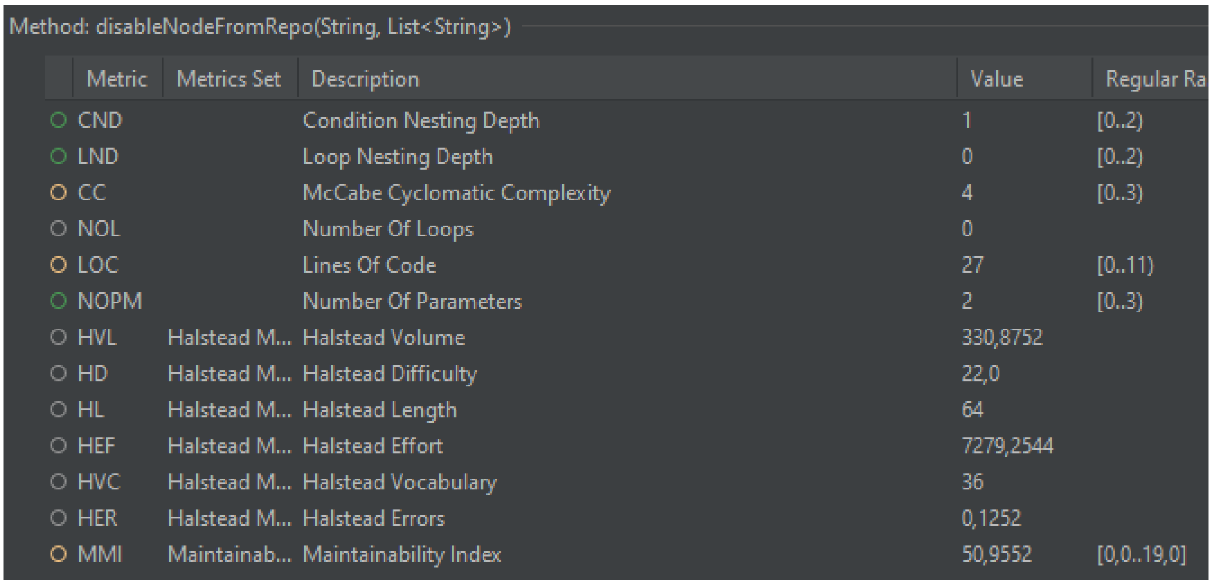


Method: deleteNode(ToaRequestModel)

	Metric	Metrics Set	Description	Value	Regular Ra...
○	CND		Condition Nesting Depth	1	[0..2)
○	LND		Loop Nesting Depth	0	[0..2)
○	CC		McCabe Cyclomatic Complexity	2	[0..3)
○	NOL		Number Of Loops	0	
○	LOC		Lines Of Code	16	[0..11)
○	NOPM		Number Of Parameters	1	[0..3)
○	HVL	Halstead M...	Halstead Volume	99,4043	
○	HD	Halstead M...	Halstead Difficulty	6,7222	
○	HL	Halstead M...	Halstead Length	23	
○	HEF	Halstead M...	Halstead Effort	668,2181	
○	HVC	Halstead M...	Halstead Vocabulary	20	
○	HER	Halstead M...	Halstead Errors	0,0255	
○	MMI	Maintainab...	Maintainability Index	59,6667	[0,0..19,0]

As you can see the MMI index increased which is good. The software has a bug in it where it treats 0-20 range for MMI as good whereas in reality it is meant to be good.
With the main functionality in a helper method, the following is the metrics after refactoring the helper method. We got rid of another conditional statement that was duplicated in the

main code and the helper code meaning the cyclomatic complexity decreased by 1 again and is now an acceptable 4.



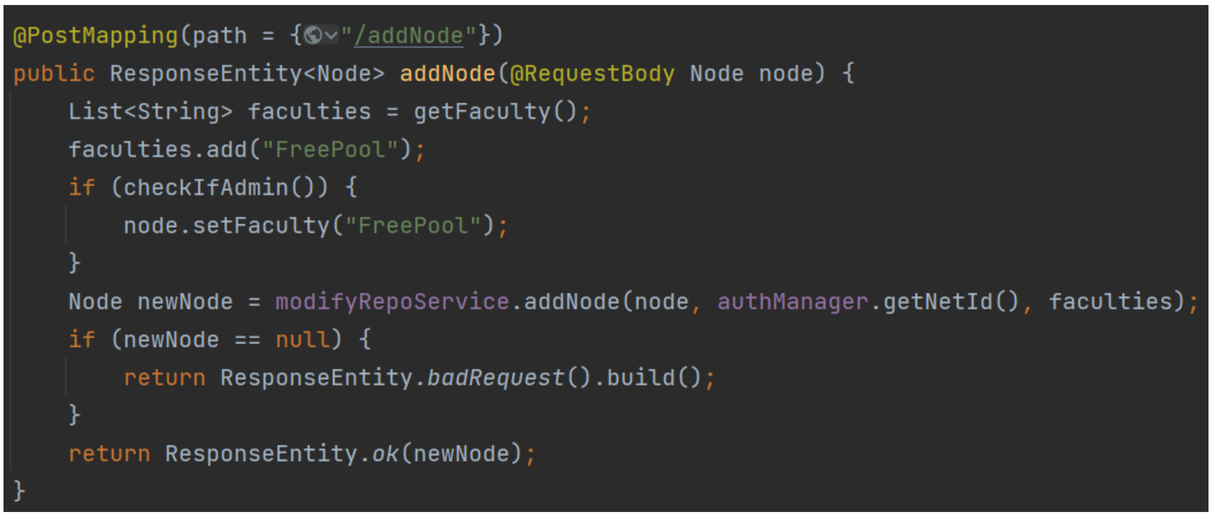
Method: disableNodeFromRepo(String, List<String>)

	Metric	Metrics Set	Description	Value	Regular Ra
O	CND		Condition Nesting Depth	1	[0..2)
O	LND		Loop Nesting Depth	0	[0..2)
O	CC		McCabe Cyclomatic Complexity	4	[0..3)
O	NOL		Number Of Loops	0	
O	LOC		Lines Of Code	27	[0..11)
O	NOPM		Number Of Parameters	2	[0..3)
O	HVL	Halstead M...	Halstead Volume	330,8752	
O	HD	Halstead M...	Halstead Difficulty	22,0	
O	HL	Halstead M...	Halstead Length	64	
O	HEF	Halstead M...	Halstead Effort	7279,2544	
O	HVC	Halstead M...	Halstead Vocabulary	36	
O	HER	Halstead M...	Halstead Errors	0,1252	
O	MMI	Maintainab...	Maintainability Index	50,9552	[0,0..19,0]

Method: NodeController/addNode

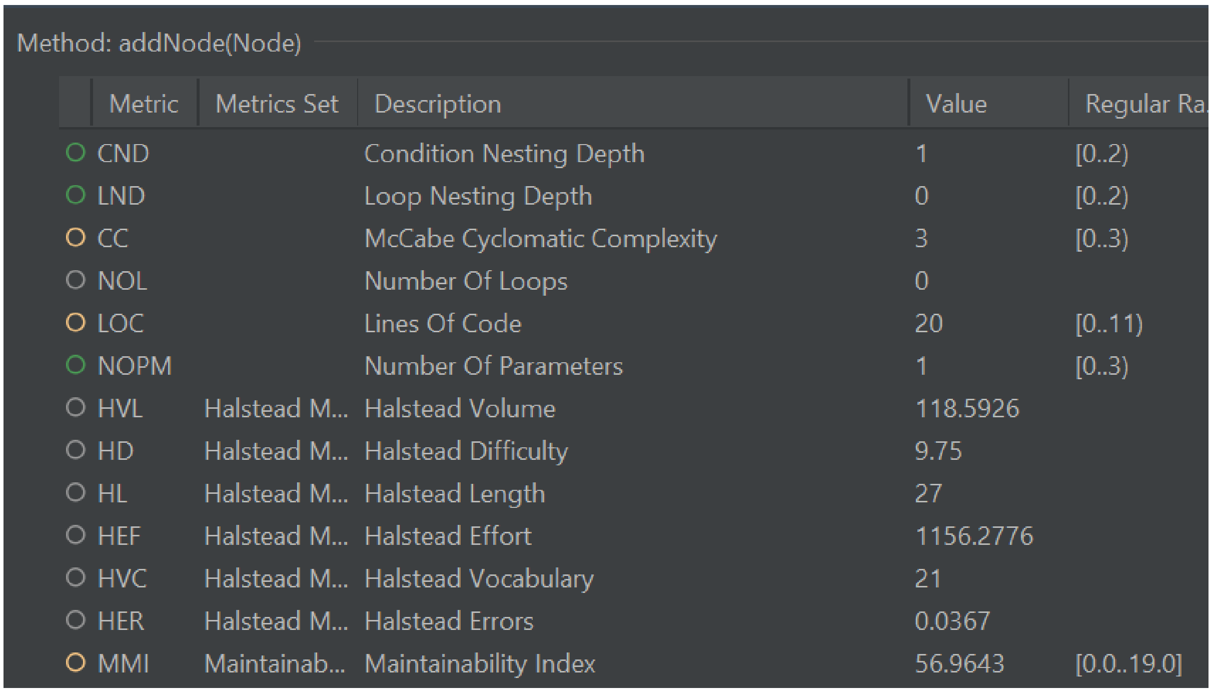
Introduction

Here is the method before refactoring:



```java
@PostMapping(path = {"/addNode"})
public ResponseEntity<Node> addNode(@RequestBody Node node) {
    List<String> faculties = getFaculty();
    faculties.add("FreePool");
    if (checkIfAdmin()) {
        node.setFaculty("FreePool");
    }
    Node newNode = modifyRepoService.addNode(node, authManager.getNetId(), faculties);
    if (newNode == null) {
        return ResponseEntity.badRequest().build();
    }
    return ResponseEntity.ok(newNode);
}
```

According to the metrics tool, there are three metrics over the regular range - Cyclomatic Complexity, Lines of Code and Maintainability Index.


Method: addNode(Node)

	Metric	Metrics Set	Description	Value	Regular Ra
○	CND		Condition Nesting Depth	1	[0..2)
○	LND		Loop Nesting Depth	0	[0..2)
○	CC		McCabe Cyclomatic Complexity	3	[0..3)
○	NOL		Number Of Loops	0	
○	LOC		Lines Of Code	20	[0..11)
○	NOPM		Number Of Parameters	1	[0..3)
○	HVL	Halstead M...	Halstead Volume	118.5926	
○	HD	Halstead M...	Halstead Difficulty	9.75	
○	HL	Halstead M...	Halstead Length	27	
○	HEF	Halstead M...	Halstead Effort	1156.2776	
○	HVC	Halstead M...	Halstead Vocabulary	21	
○	HER	Halstead M...	Halstead Errors	0.0367	
○	MMI	Maintainab...	Maintainability Index	56.9643	[0.0..19.0]

Cyclomatic Complexity

Before the changes the Cyclomatic complexity of this method was 3 - a bit over the regular values. I have decided that in order to reduce the Cyclomatic Complexity of this method, I should move some of the logic outside the body of the method.

I made a separate class - CheckHelper, which NodeController extends, where I basically moved most of the if-else statements in different methods so as to, first, reduce the Cyclomatic Complexity of the addNode method, and, second, simplify the whole class NodeController and improve its metrics as well (part of my work also).

After the refactoring the Cyclomatic complexity of the addNode method is 1, which is in the regular range.

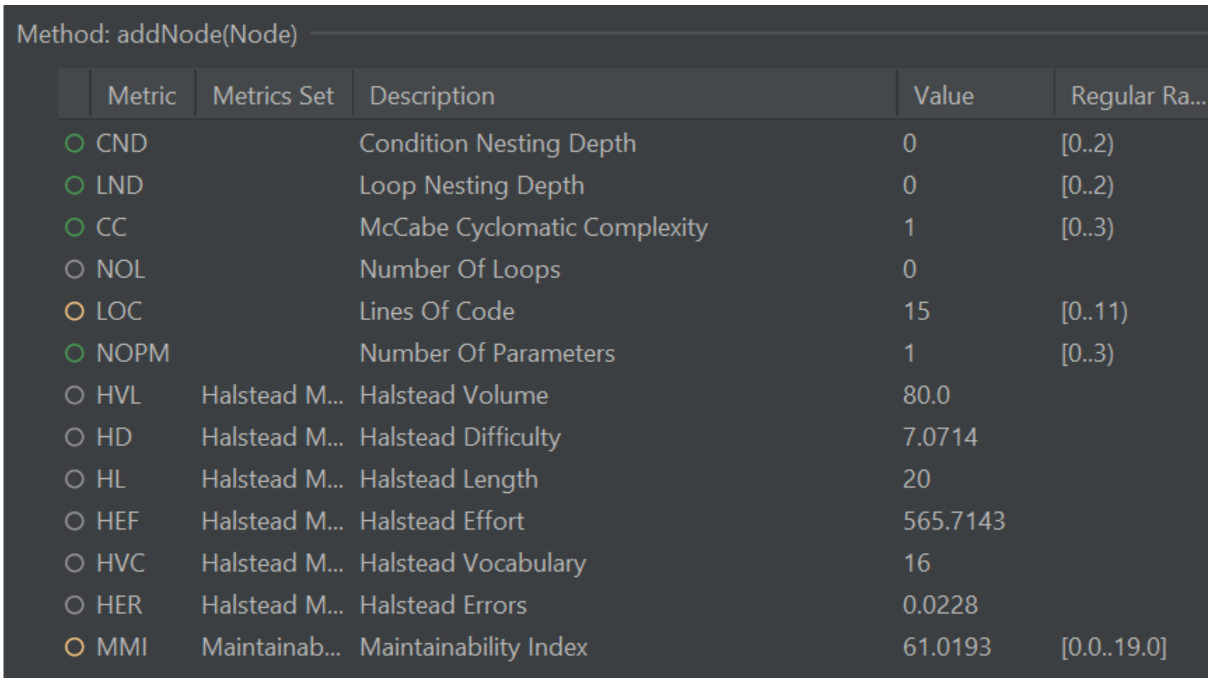
Lines of Code

Before the changes, the method had 20 lines of code, which was a bit over the regular range. After the refactoring, which, as stated above, included moving some of the logic of the addNode method to a separate helper class CheckHelper, the method now has 15 lines of code, which is ultimately the lowest achievable number possible so as to preserve the method structure as is.

Conclusion

All in all, I managed to reduce the Cyclomatic Complexity of the method to a value in the regular range and reduce the Lines of Code with nearly 30% so that the number is as close as possible to the regular range. These improvements came at the price of a slightly increased Maintainability Index of the method, which is possibly due to the introduction of the helper class CheckHelper (because changes in the helper class may also affect the addNode method). This was a tradeoff that I needed to have in mind when refactoring. As an addition, the Condition Nesting Depth was reduced from 1 to 0.

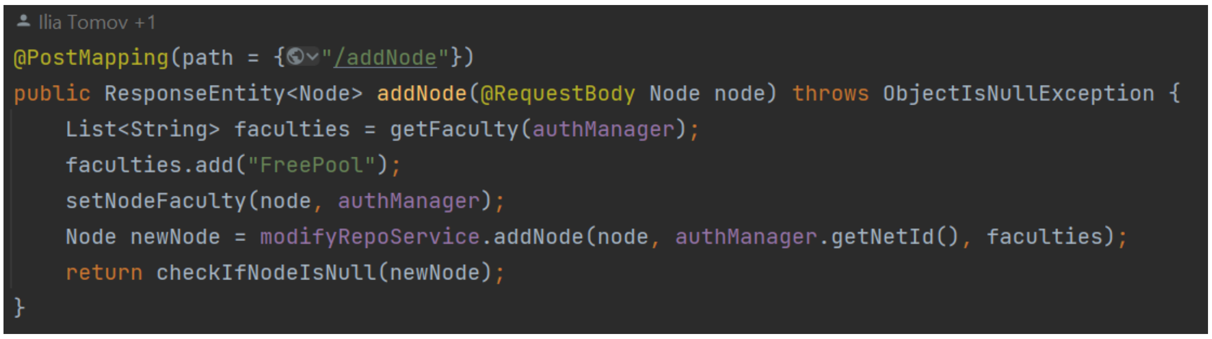
Below the new metrics of the method addNode are shown:



Method: addNode(Node)

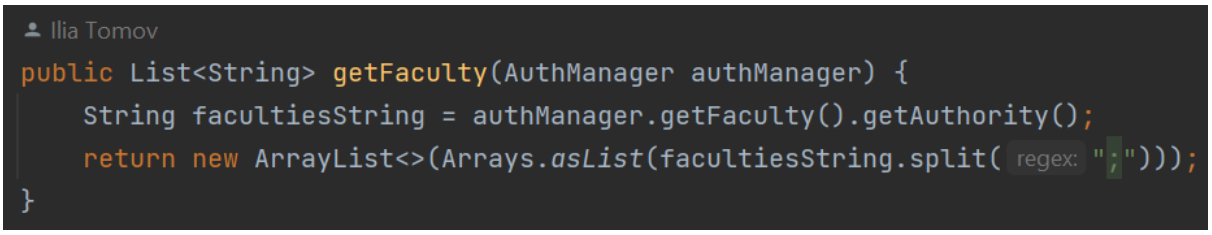
Metric	Metrics Set	Description	Value	Regular Ra...
○ CND		Condition Nesting Depth	0	[0..2)
○ LND		Loop Nesting Depth	0	[0..2)
○ CC		McCabe Cyclomatic Complexity	1	[0..3)
○ NOL		Number Of Loops	0	
○ LOC		Lines Of Code	15	[0..11)
○ NOPM		Number Of Parameters	1	[0..3)
○ HVL	Halstead M...	Halstead Volume	80.0	
○ HD	Halstead M...	Halstead Difficulty	7.0714	
○ HL	Halstead M...	Halstead Length	20	
○ HEF	Halstead M...	Halstead Effort	565.7143	
○ HVC	Halstead M...	Halstead Vocabulary	16	
○ HER	Halstead M...	Halstead Errors	0.0228	
○ MMI	Maintainab...	Maintainability Index	61.0193	[0.0..19.0]

Here is the method after refactoring:

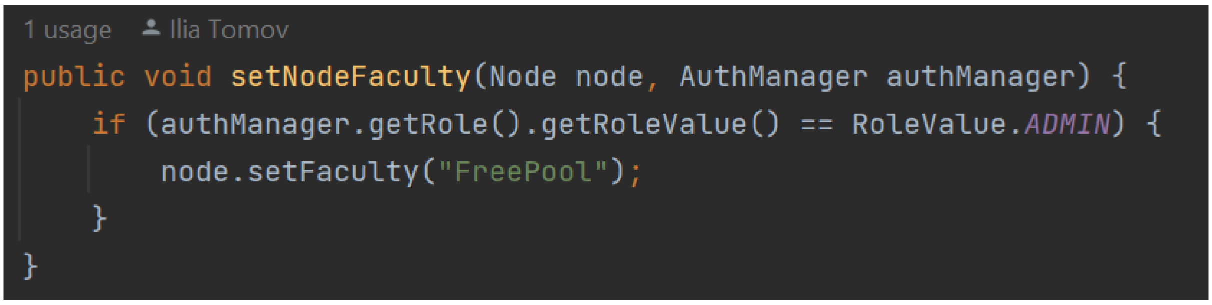


```
Ilia Tomov +1
@PostMapping(path = {"/addNode"})
public ResponseEntity<Node> addNode(@RequestBody Node node) throws ObjectIsNullException {
    List<String> faculties = getFaculty(authManager);
    faculties.add("FreePool");
    setNodeFaculty(node, authManager);
    Node newNode = modifyRepoService.addNode(node, authManager.getNetId(), faculties);
    return checkIfNodeIsNull(newNode);
}
```

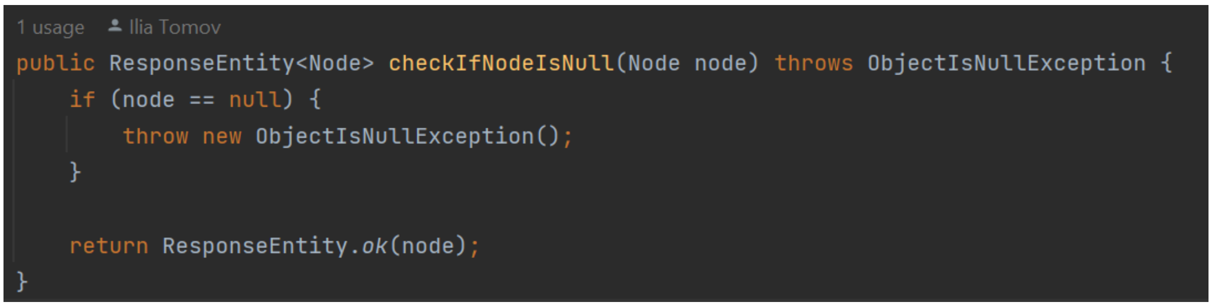
The methods used from CheckHelper:



```
Ilia Tomov
public List<String> getFaculty(AuthManager authManager) {
    String facultiesString = authManager.getFaculty().getAuthority();
    return new ArrayList<>(Arrays.asList(facultiesString.split( regex: ";")));
}
```



```java
1 usage   Ilia Tomov
public void setNodeFaculty(Node node, AuthManager authManager) {
    if (authManager.getRole().getRoleValue() == RoleValue.ADMIN) {
        node.setFaculty("FreePool");
    }
}
```

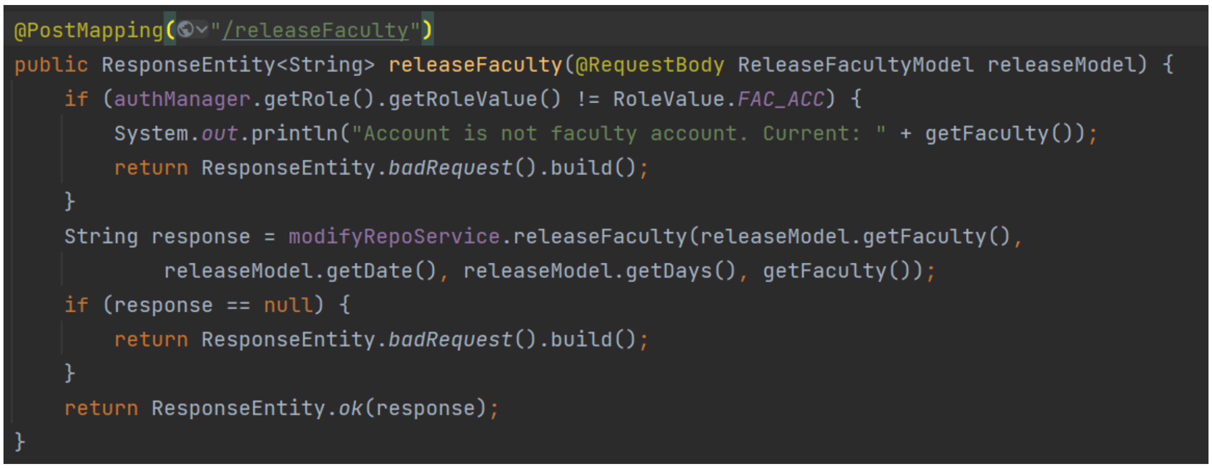


```java
1 usage   Ilia Tomov
public ResponseEntity<Node> checkIfNodeIsNull(Node node) throws ObjectIsNullException {
    if (node == null) {
        throw new ObjectIsNullException();
    }

    return ResponseEntity.ok(node);
}
```

Method: NodeController/releaseFaculty

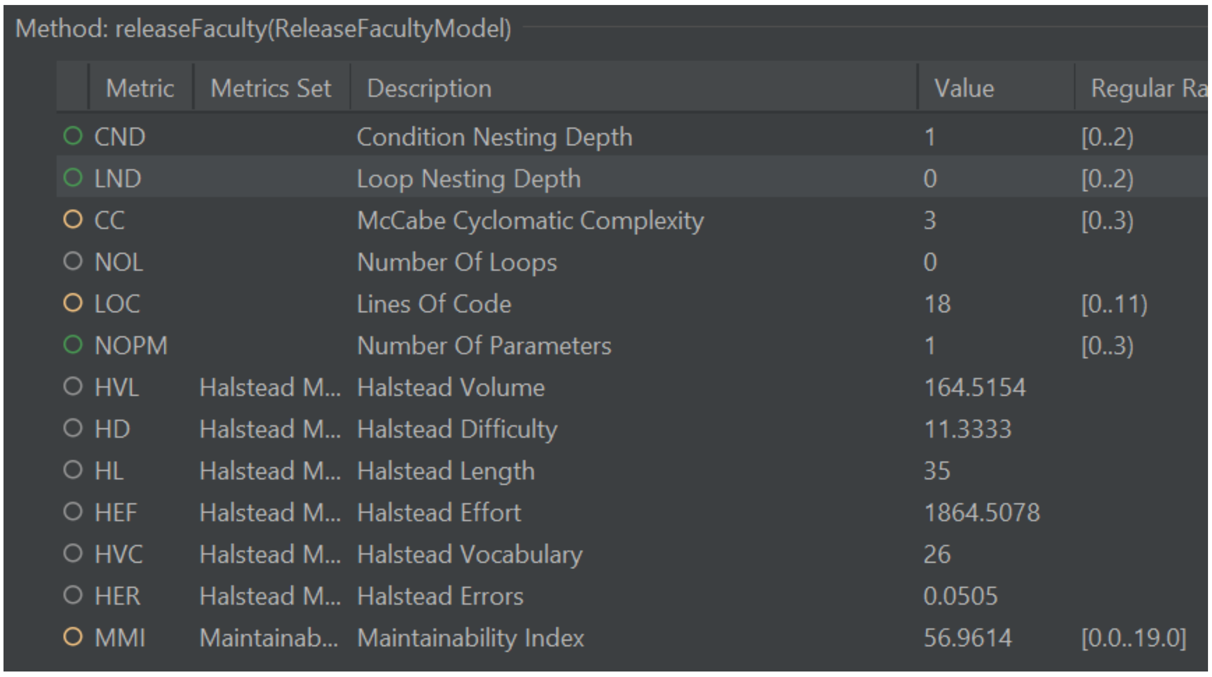
Introduction

Here is the method before refactoring:



```java
@PostMapping("/releaseFaculty")
public ResponseEntity<String> releaseFaculty(@RequestBody ReleaseFacultyModel releaseModel) {
    if (authManager.getRole().getRoleValue() != RoleValue.FAC_ACC) {
        System.out.println("Account is not faculty account. Current: " + getFaculty());
        return ResponseEntity.badRequest().build();
    }
    String response = modifyRepoService.releaseFaculty(releaseModel.getFaculty(),
            releaseModel.getDate(), releaseModel.getDays(), getFaculty());
    if (response == null) {
        return ResponseEntity.badRequest().build();
    }
    return ResponseEntity.ok(response);
}
```

According to the metrics tool, there are three metrics over the regular range - Cyclomatic Complexity, Lines of Code and Maintainability Index.



Method: releaseFaculty(ReleaseFacultyModel)

	Metric	Metrics Set	Description	Value	Regular Ra
○	CND		Condition Nesting Depth	1	[0..2)
○	LND		Loop Nesting Depth	0	[0..2)
○	CC		McCabe Cyclomatic Complexity	3	[0..3)
○	NOL		Number Of Loops	0	
○	LOC		Lines Of Code	18	[0..11)
○	NOPM		Number Of Parameters	1	[0..3)
○	HVL	Halstead M...	Halstead Volume	164.5154	
○	HD	Halstead M...	Halstead Difficulty	11.3333	
○	HL	Halstead M...	Halstead Length	35	
○	HEF	Halstead M...	Halstead Effort	1864.5078	
○	HVC	Halstead M...	Halstead Vocabulary	26	
○	HER	Halstead M...	Halstead Errors	0.0505	
○	MMI	Maintainab...	Maintainability Index	56.9614	[0.0..19.0]

Cyclomatic Complexity

Before the changes the Cyclomatic complexity of this method was 3 - a bit over the regular values. I have decided that in order to reduce the Cyclomatic Complexity of this method, I should move some of the logic outside the body of the method.

I made a separate class - CheckHelper, which NodeController extends, where I basically moved most of the if-else statements in different methods so as to, first, reduce the Cyclomatic Complexity of the releaseFaculty method, and, second, simplify the whole class NodeController and improve its metrics as well (part of my work also).

After the refactoring the Cyclomatic complexity of the releaseFaculty method is 1, which is in the regular range.

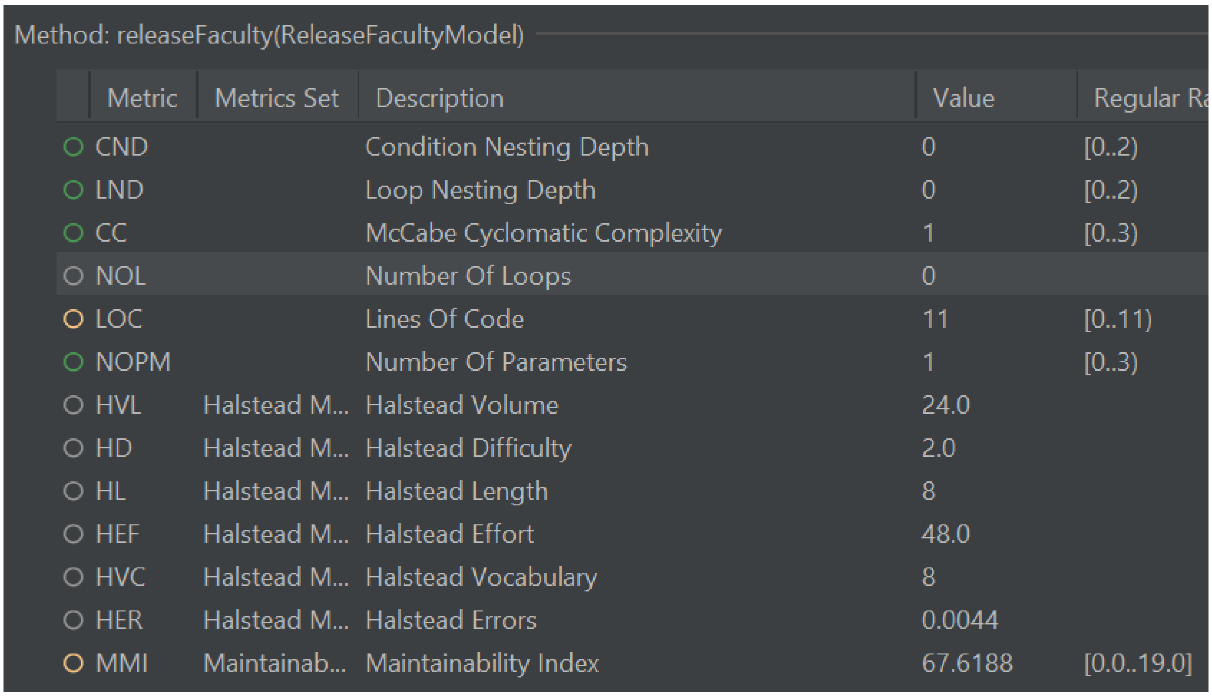
Lines of Code

Before the changes, the method had 18 lines of code, which was a bit over the regular range. After the refactoring, which, as stated above, included moving some of the logic of the releaseFacucty method to a separate helper class CheckHelper, the method now has 11 lines of code, which is on the border of the regular range.

Conclusion

All in all, I managed to reduce the Cyclomatic Complexity and the Lines of Code of the method to a value in the regular range. These improvements came at the price of a slightly increased Maintainability Index of the method, which is possibly due to the introduction of the helper class CheckHelper (because changes in the helper class may also affect the releaseFaculty method). This was a tradeoff that I needed to have in mind when refactoring. As an addition, the Condition Nesting Depth was reduced from 1 to 0.

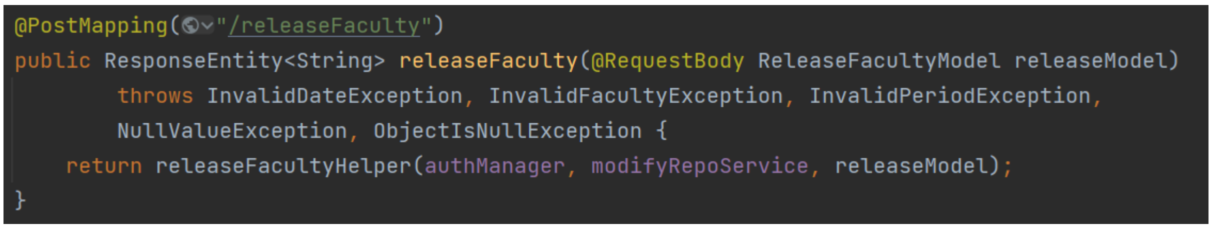
Below the new metrics of the method releaseFaculty are shown:



Method: releaseFaculty(ReleaseFacultyModel)

	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	11	[0..11)
○	NOPM		Number Of Parameters	1	[0..3)
○	HVL	Halstead M...	Halstead Volume	24.0	
○	HD	Halstead M...	Halstead Difficulty	2.0	
○	HL	Halstead M...	Halstead Length	8	
○	HEF	Halstead M...	Halstead Effort	48.0	
○	HVC	Halstead M...	Halstead Vocabulary	8	
○	HER	Halstead M...	Halstead Errors	0.0044	
○	MMI	Maintainab...	Maintainability Index	67.6188	[0.0..19.0]

Here is the method after refactoring:

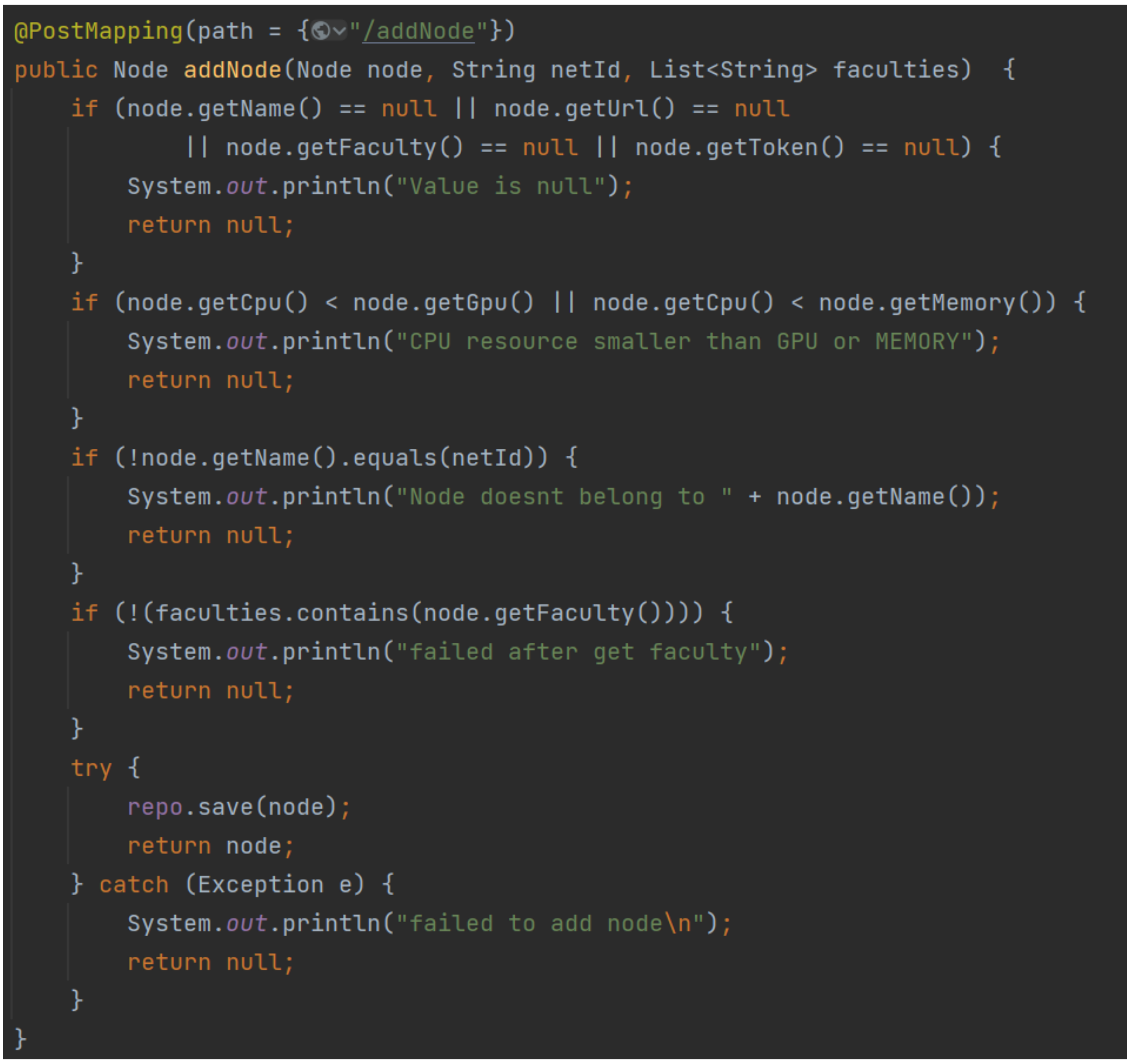


```java
@PostMapping("/releaseFaculty")
public ResponseEntity<String> releaseFaculty(@RequestBody ReleaseFacultyModel releaseModel)
        throws InvalidDateException, InvalidFacultyException, InvalidPeriodException,
        NullValueException, ObjectIsNullException {
    return releaseFacultyHelper(authManager, modifyRepoService, releaseModel);
}
```

Method: ModifyRepoService/addNode

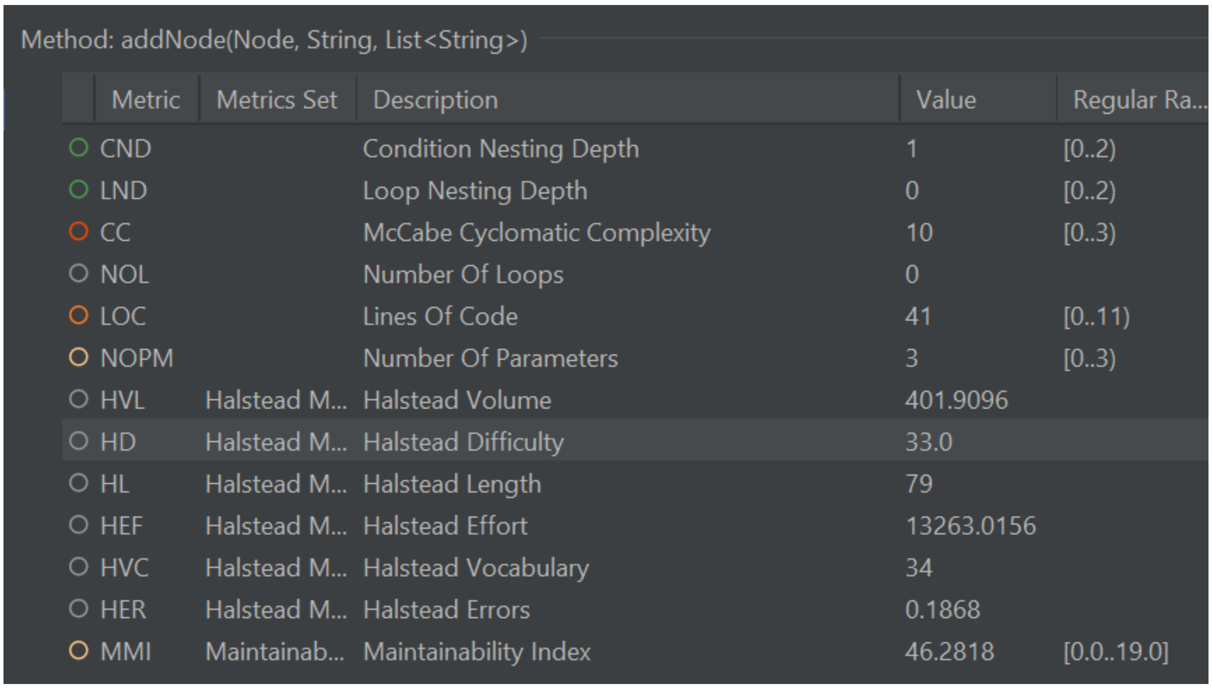
Introduction

Here is the method before refactoring:



```java
@PostMapping(path = {"/addNode"})
public Node addNode(Node node, String netId, List<String> faculties)  {
    if (node.getName() == null || node.getUrl() == null
            || node.getFaculty() == null || node.getToken() == null) {
        System.out.println("Value is null");
        return null;
    }
    if (node.getCpu() < node.getGpu() || node.getCpu() < node.getMemory()) {
        System.out.println("CPU resource smaller than GPU or MEMORY");
        return null;
    }
    if (!node.getName().equals(netId)) {
        System.out.println("Node doesnt belong to " + node.getName());
        return null;
    }
    if (!(faculties.contains(node.getFaculty()))) {
        System.out.println("failed after get faculty");
        return null;
    }
    try {
        repo.save(node);
        return node;
    } catch (Exception e) {
        System.out.println("failed to add node\n");
        return null;
    }
}
```

According to the metrics tool, there are four metrics over the regular range - Cyclomatic Complexity, Lines of Code, Number of Parameters and Maintainability Index.



Method: addNode(Node, String, List<String>)

Metric	Metrics Set	Description	Value	Regular Ra...
CND		Condition Nesting Depth	1	[0..2)
LND		Loop Nesting Depth	0	[0..2)
CC		McCabe Cyclomatic Complexity	10	[0..3)
NOL		Number Of Loops	0	
LOC		Lines Of Code	41	[0..11)
NOPM		Number Of Parameters	3	[0..3)
HVL	Halstead M...	Halstead Volume	401.9096	
HD	Halstead M...	Halstead Difficulty	33.0	
HL	Halstead M...	Halstead Length	79	
HEF	Halstead M...	Halstead Effort	13263.0156	
HVC	Halstead M...	Halstead Vocabulary	34	
HER	Halstead M...	Halstead Errors	0.1868	
MMI	Maintainab...	Maintainability Index	46.2818	[0.0..19.0]

Cyclomatic Complexity

Before the changes the Cyclomatic Complexity of this method was 10 - significantly over the regular range. I have decided that in order to reduce the Cyclomatic Complexity of this method, I should move some of the logic outside the body of the method.

I made a separate class - ModifyRepoHelper, which ModifyRepoService extends, where I basically moved most of the if-else statements in different methods so as to reduce the cyclomatic complexity of the addNode method.

After the refactoring, the value of the Cyclomatic Complexity is 1, a value in the regular range, which is a significant improvement from the previous value.

Lines of Code

Before the changes, the method had 41 lines of code, which was significantly over the regular range. After the refactoring, which, as stated above, included moving some of the logic of the addNode method to a separate helper class ModifyRepoHelper, the method now has 16 lines of code, which is a significant improvement from the previous value.

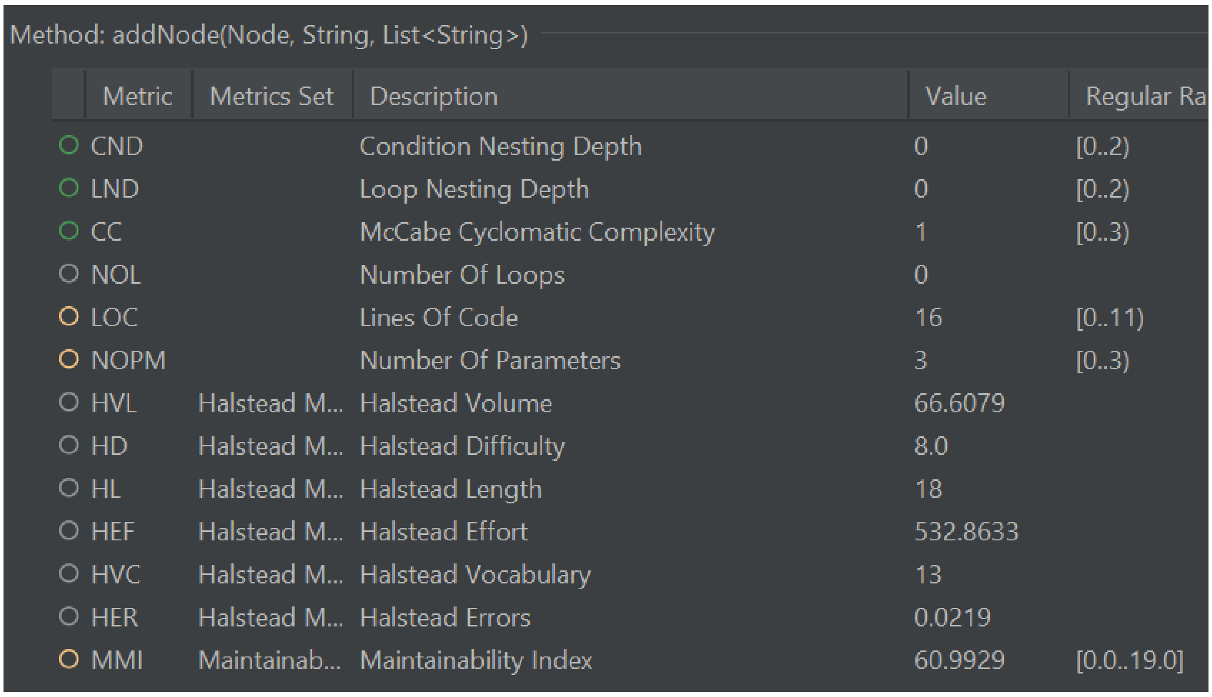
Number of parameters

The number of parameters for this method is 3, just outside the regular range. Though, no further improvement for this metric could be done, as these parameters play an important role in the method.

Conclusion

All in all, I managed to reduce the Cyclomatic Complexity of the method to a value in the regular range and reduce the Lines of Code by more than a factor of two so that the number is as close as possible to the regular range. These improvements came at the price of a slightly increased Maintainability Index of the method, which is possibly due to the introduction of the helper class ModifyRepoHelper (because changes in the helper class may also affect the addNode method). This was a tradeoff that I needed to have in mind when refactoring. As an addition, the Condition Nesting Depth was reduced from 1 to 0.

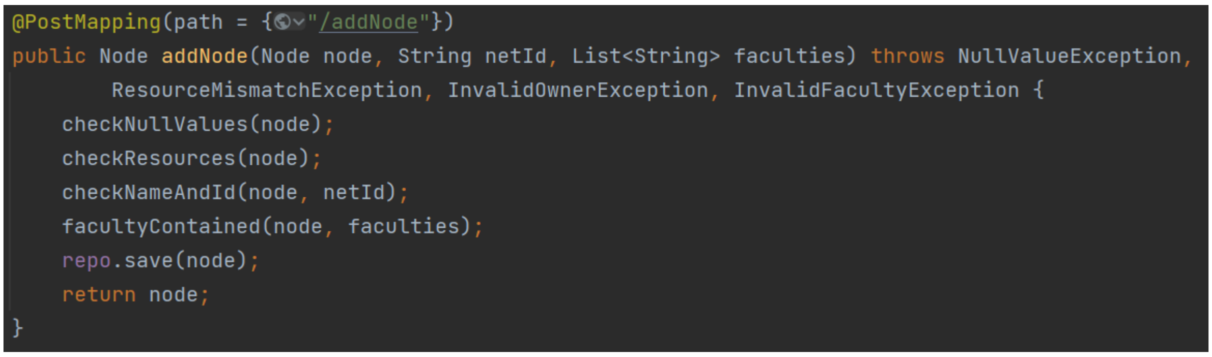
Below the new metrics of the method addNode are shown:



Method: addNode(Node, String, List<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	16	[0..11)
○	NOPM		Number Of Parameters	3	[0..3)
○	HVL	Halstead M...	Halstead Volume	66.6079	
○	HD	Halstead M...	Halstead Difficulty	8.0	
○	HL	Halstead M...	Halstead Length	18	
○	HEF	Halstead M...	Halstead Effort	532.8633	
○	HVC	Halstead M...	Halstead Vocabulary	13	
○	HER	Halstead M...	Halstead Errors	0.0219	
○	MMI	Maintainab...	Maintainability Index	60.9929	[0.0..19.0]

Here is the method after refactoring:

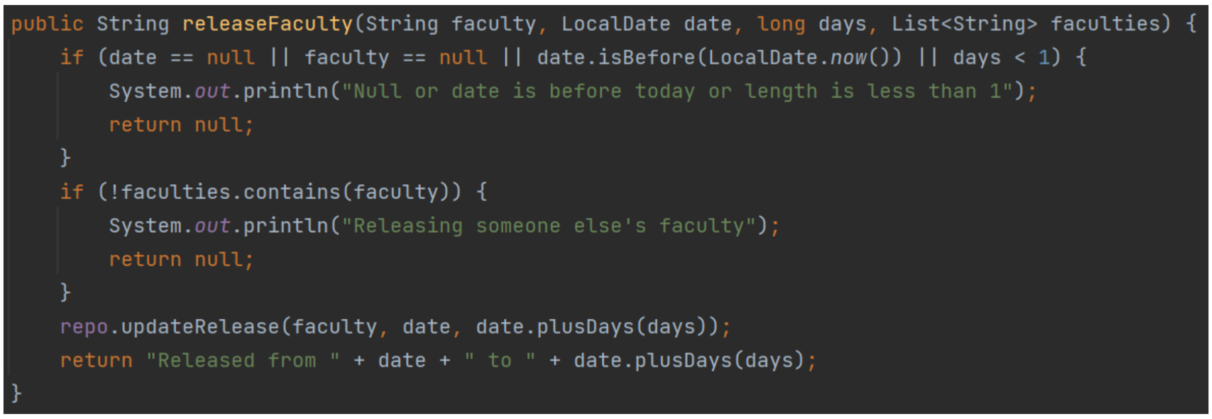


```java
@PostMapping(path = {"/addNode"})
public Node addNode(Node node, String netId, List<String> faculties) throws NullValueException,
        ResourceMismatchException, InvalidOwnerException, InvalidFacultyException {
    checkNullValues(node);
    checkResources(node);
    checkNameAndId(node, netId);
    facultyContained(node, faculties);
    repo.save(node);
    return node;
}
```

Method: ModifyRepoService/releaseFaculty

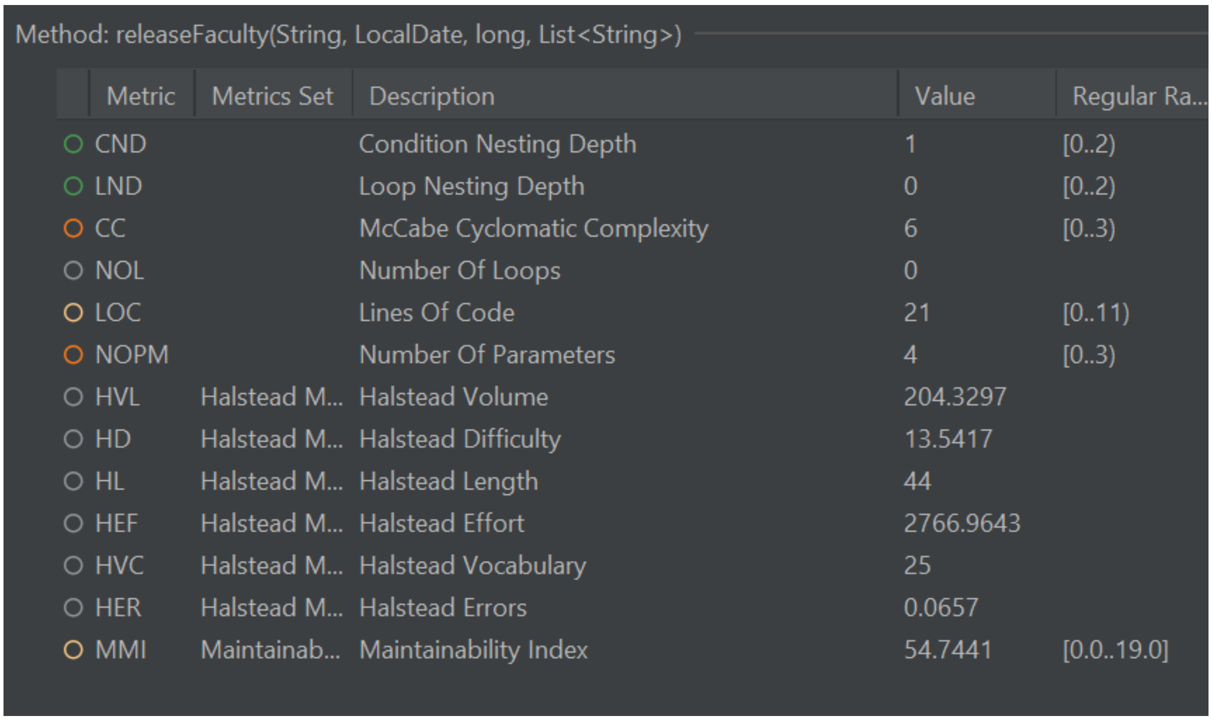
Introduction

Here is the method before refactoring:



```java
public String releaseFaculty(String faculty, LocalDate date, long days, List<String> faculties) {
    if (date == null || faculty == null || date.isBefore(LocalDate.now()) || days < 1) {
        System.out.println("Null or date is before today or length is less than 1");
        return null;
    }
    if (!faculties.contains(faculty)) {
        System.out.println("Releasing someone else's faculty");
        return null;
    }
    repo.updateRelease(faculty, date, date.plusDays(days));
    return "Released from " + date + " to " + date.plusDays(days);
}
```

According to the metrics tool, there are four metrics over the regular range - Cyclomatic Complexity, Lines of Code, Number of Parameters and Maintainability Index.



Method: releaseFaculty(String, LocalDate, long, List<String>)

Metric	Metrics Set	Description	Value	Regular Ra...
CND		Condition Nesting Depth	1	[0..2)
LND		Loop Nesting Depth	0	[0..2)
CC		McCabe Cyclomatic Complexity	6	[0..3)
NOL		Number Of Loops	0	
LOC		Lines Of Code	21	[0..11)
NOPM		Number Of Parameters	4	[0..3)
HVL	Halstead M...	Halstead Volume	204.3297	
HD	Halstead M...	Halstead Difficulty	13.5417	
HL	Halstead M...	Halstead Length	44	
HEF	Halstead M...	Halstead Effort	2766.9643	
HVC	Halstead M...	Halstead Vocabulary	25	
HER	Halstead M...	Halstead Errors	0.0657	
MMI	Maintainab...	Maintainability Index	54.7441	[0.0..19.0]

Cyclomatic Complexity

Before the changes the Cyclomatic Complexity of this method was 6 - significantly over the regular range. I have decided that in order to reduce the Cyclomatic Complexity of this method, I should move some of the logic outside the body of the method.

I made a separate class - ModifyRepoHelper, which ModifyRepoService extends, where I basically moved most of the if-else statements in different methods so as to reduce the cyclomatic complexity of the releaseFaculty method.

After the refactoring, the value of the Cyclomatic Complexity is 1, a value in the regular range, which is a significant improvement from the previous value.

Lines of Code

Before the changes, the method had 21 lines of code, which was significantly over the regular range. After the refactoring, which, as stated above, included moving some of the logic of the releaseFaculty method to a separate helper class ModifyRepoHelper, the method now has 16 lines of code, which is a slight improvement from the previous value.

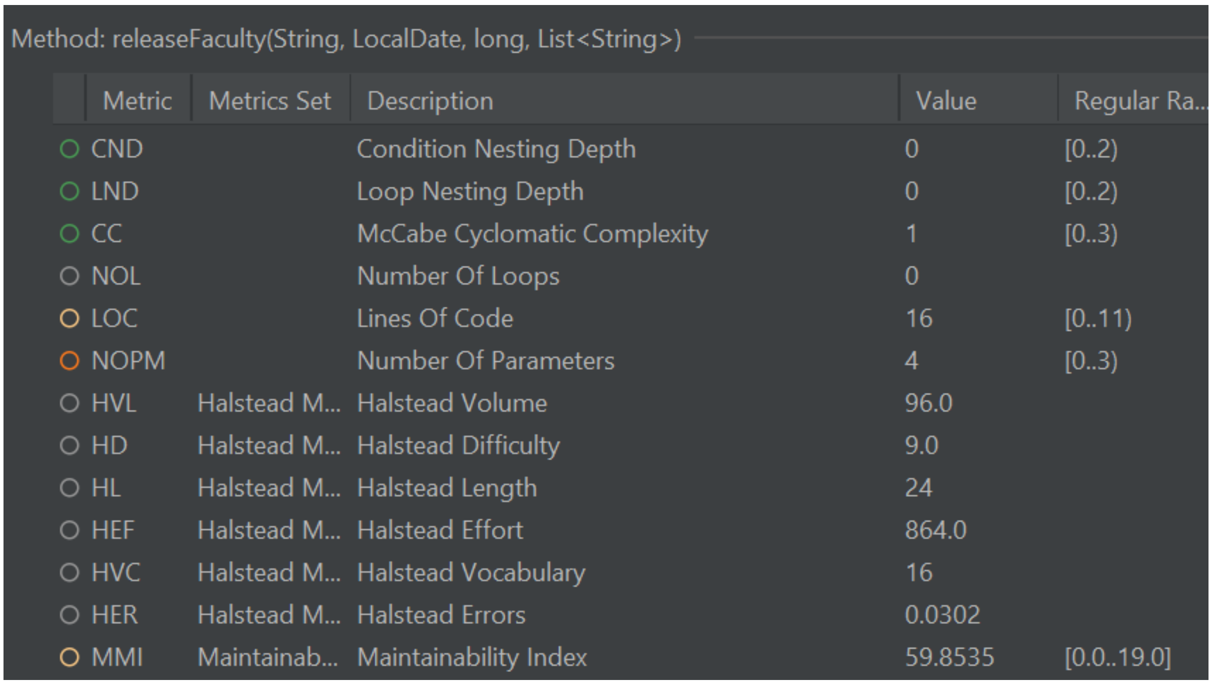
Number of parameters

The number of parameters for this method is 4, just outside the regular range. Though, no further improvement for this metric could be done, as these parameters play an important role in the method.

Conclusion

All in all, I managed to reduce the Cyclomatic Complexity of the method to a value in the regular range and reduce the Lines of Code with 25% so that the number is as close as possible to the regular range. These improvements came at the price of a slightly increased Maintainability Index of the method, which is possibly due to the introduction of the helper class ModifyRepoHelper (because changes in the helper class may also affect the releaseFaculty method). This was a tradeoff that I needed to have in mind when refactoring. As an addition, the Condition Nesting Depth was reduced from 1 to 0.

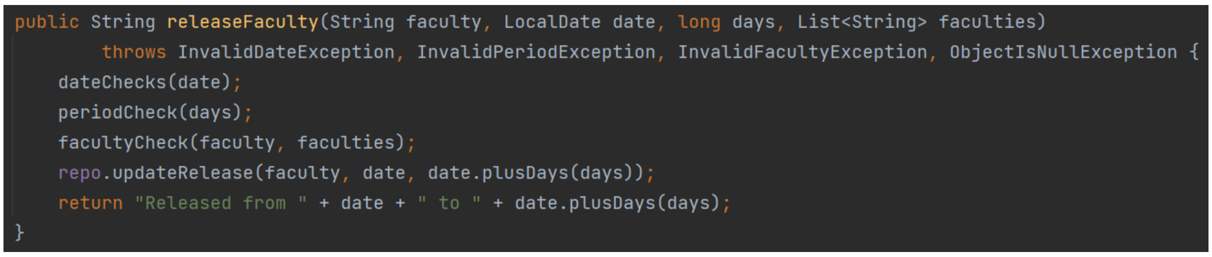
Below the new metrics of the method releaseFaculty are shown:



Method: releaseFaculty(String, LocalDate, long, List<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	16	[0..11)
NOPM		Number Of Parameters	4	[0..3)
HVL	Halstead M...	Halstead Volume	96.0	
HD	Halstead M...	Halstead Difficulty	9.0	
HL	Halstead M...	Halstead Length	24	
HEF	Halstead M...	Halstead Effort	864.0	
HVC	Halstead M...	Halstead Vocabulary	16	
HER	Halstead M...	Halstead Errors	0.0302	
MMI	Maintainab...	Maintainability Index	59.8535	[0.0..19.0]

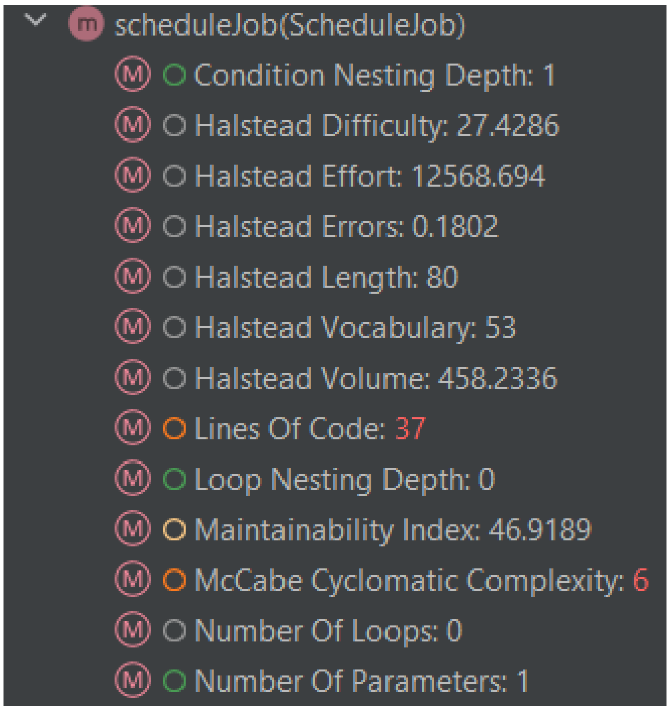
Here is the method after refactoring:



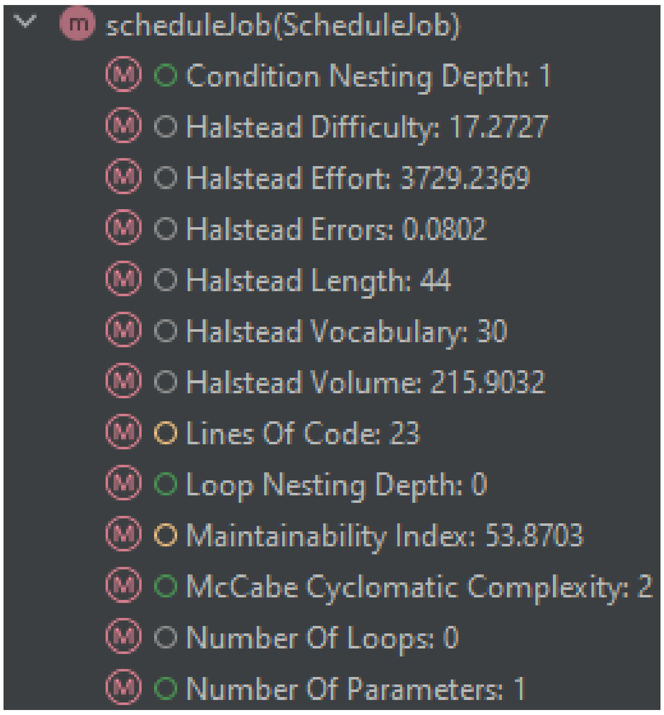
```
public String releaseFaculty(String faculty, LocalDate date, long days, List<String> faculties)
        throws InvalidDateException, InvalidPeriodException, InvalidFacultyException, ObjectIsNullException {
    dateChecks(date);
    periodCheck(days);
    facultyCheck(faculty, faculties);
    repo.updateRelease(faculty, date, date.plusDays(days));
    return "Released from " + date + " to " + date.plusDays(days);
}
```

Method: ProcessingJobService/scheduleJob

The Method scheduleJob() in the class ProcessingJobService has been chosen to be refactored, since the LOC is 37 and CC is 6. The metrics before any changes can be seen on the left and after refactoring on the right image.

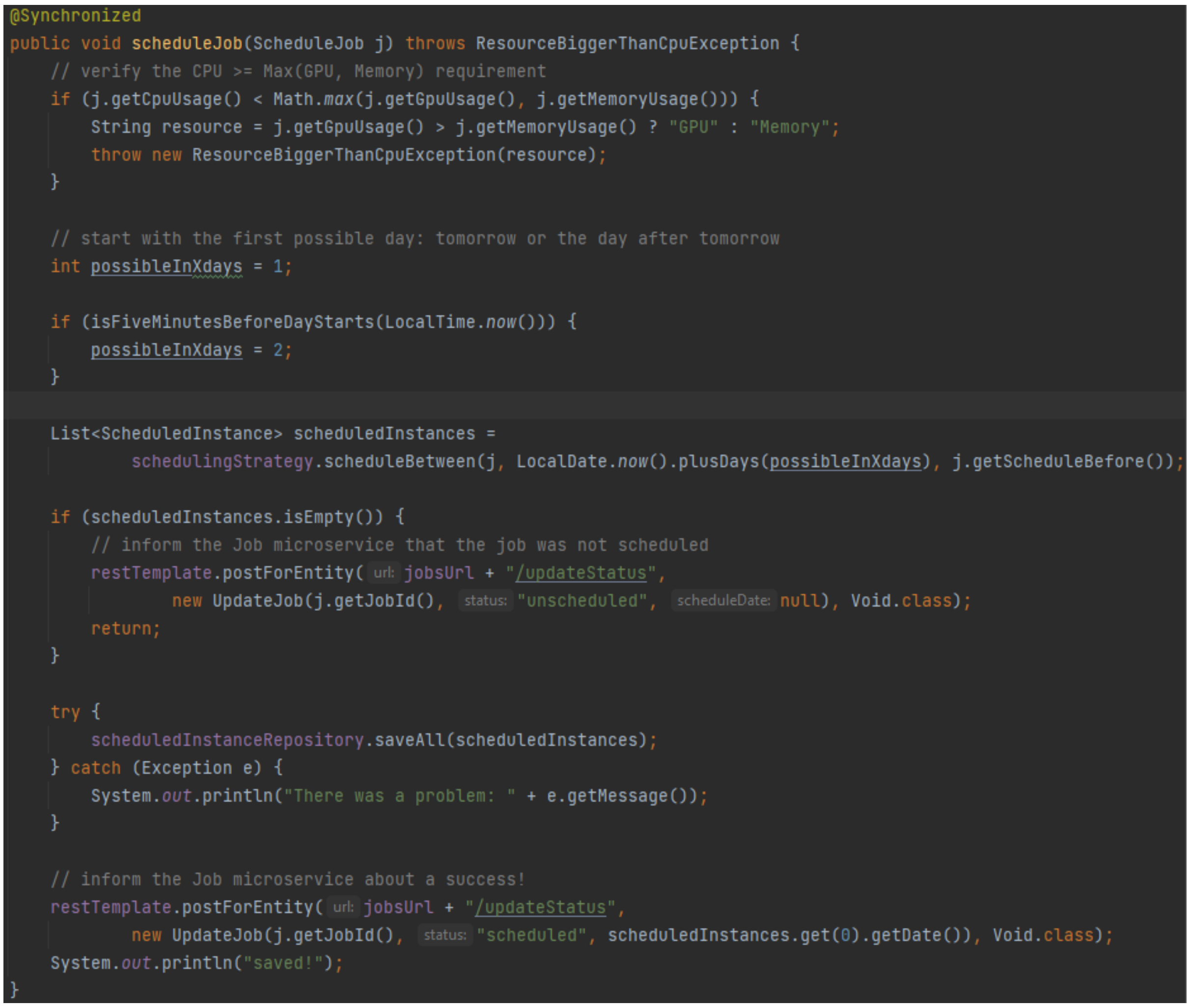


```
scheduleJob(ScheduleJob)
    Condition Nesting Depth: 1
    Halstead Difficulty: 27.4286
    Halstead Effort: 12568.694
    Halstead Errors: 0.1802
    Halstead Length: 80
    Halstead Vocabulary: 53
    Halstead Volume: 458.2336
    Lines Of Code: 37
    Loop Nesting Depth: 0
    Maintainability Index: 46.9189
    McCabe Cyclomatic Complexity: 6
    Number Of Loops: 0
    Number Of Parameters: 1
```



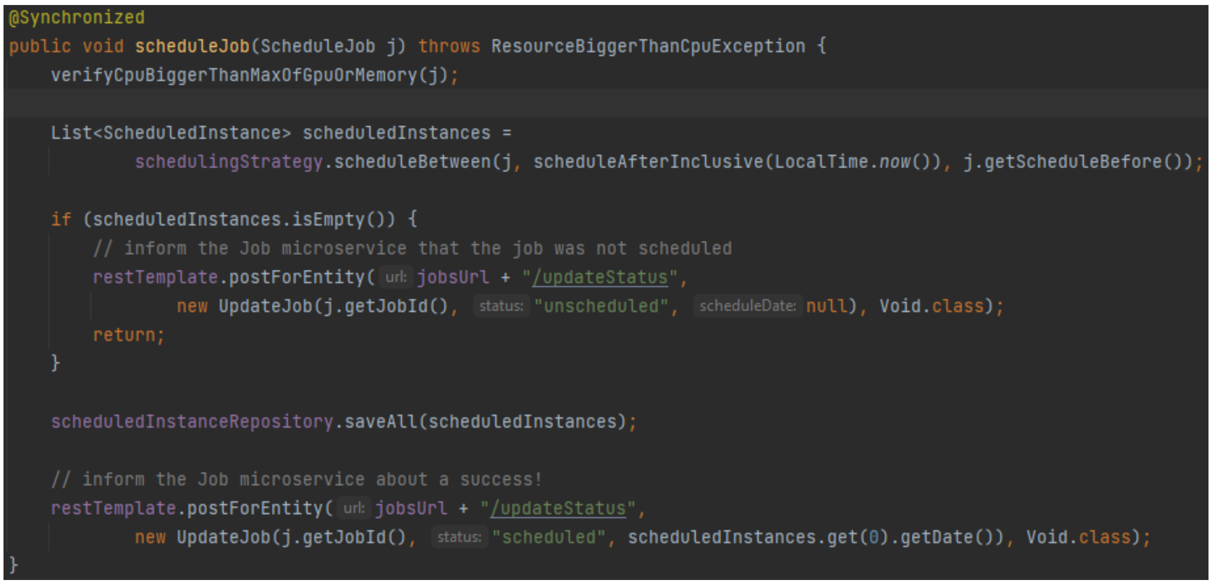
```
scheduleJob(ScheduleJob)
    Condition Nesting Depth: 1
    Halstead Difficulty: 17.2727
    Halstead Effort: 3729.2369
    Halstead Errors: 0.0802
    Halstead Length: 44
    Halstead Vocabulary: 30
    Halstead Volume: 215.9032
    Lines Of Code: 23
    Loop Nesting Depth: 0
    Maintainability Index: 53.8703
    McCabe Cyclomatic Complexity: 2
    Number Of Loops: 0
    Number Of Parameters: 1
```

The “Extract Method” has been used as the refactoring method to reduce LOC and CC. This refactoring method was done by creating two new methods that are called from the scheduleJob() method. Following you can see the method before any changes.

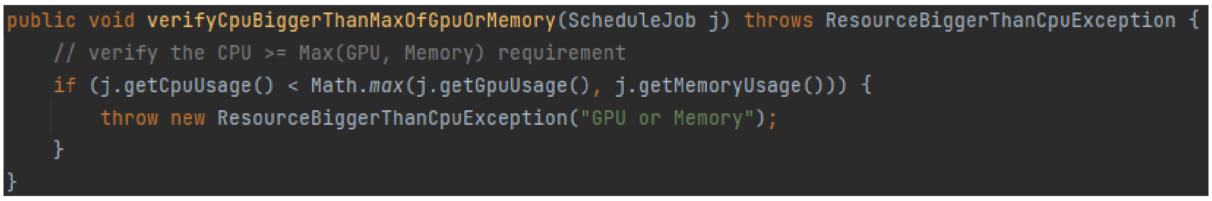


```java
@Synchronized
public void scheduleJob(ScheduleJob j) throws ResourceBiggerThanCpuException {
    // verify the CPU >= Max(GPU, Memory) requirement
    if (j.getCpuUsage() < Math.max(j.getGpuUsage(), j.getMemoryUsage())) {
        String resource = j.getGpuUsage() > j.getMemoryUsage() ? "GPU" : "Memory";
        throw new ResourceBiggerThanCpuException(resource);
    }

    // start with the first possible day: tomorrow or the day after tomorrow
    int possibleInXdays = 1;

    if (isFiveMinutesBeforeDayStarts(LocalTime.now())) {
        possibleInXdays = 2;
    }

    List<ScheduledInstance> scheduledInstances =
            schedulingStrategy.scheduleBetween(j, LocalDate.now().plusDays(possibleInXdays), j.getScheduleBefore());

    if (scheduledInstances.isEmpty()) {
        // inform the Job microservice that the job was not scheduled
        restTemplate.postForEntity( url: jobsUrl + "/updateStatus",
                new UpdateJob(j.getJobId(),  status: "unscheduled",  scheduleDate: null), Void.class);
        return;
    }

    try {
        scheduledInstanceRepository.saveAll(scheduledInstances);
    } catch (Exception e) {
        System.out.println("There was a problem: " + e.getMessage());
    }

    // inform the Job microservice about a success!
    restTemplate.postForEntity( url: jobsUrl + "/updateStatus",
            new UpdateJob(j.getJobId(),  status: "scheduled", scheduledInstances.get(0).getDate()), Void.class);
    System.out.println("saved!");
}
```

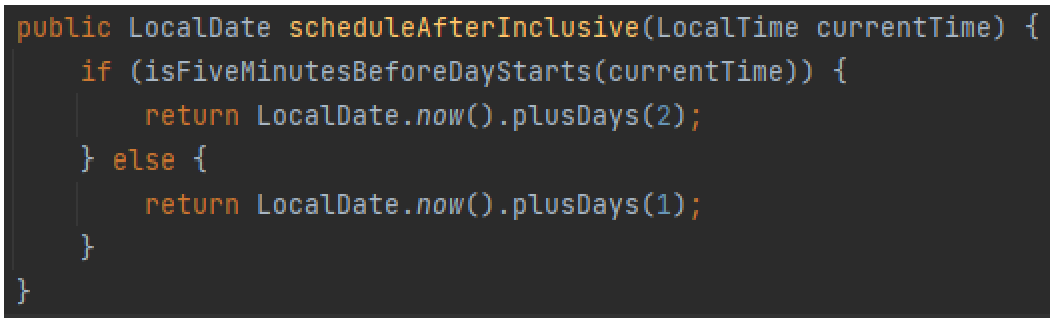
Following you can see the refactored scheduleJob() method and the two newly created methods.



```java
@Synchronized
public void scheduleJob(ScheduleJob j) throws ResourceBiggerThanCpuException {
    verifyCpuBiggerThanMaxOfGpuOrMemory(j);

    List<ScheduledInstance> scheduledInstances =
            schedulingStrategy.scheduleBetween(j, scheduleAfterInclusive(LocalTime.now()), j.getScheduleBefore());

    if (scheduledInstances.isEmpty()) {
        // inform the Job microservice that the job was not scheduled
        restTemplate.postForEntity( url: jobsUrl + "/updateStatus",
                new UpdateJob(j.getJobId(),  status: "unscheduled",  scheduleDate: null), Void.class);
        return;
    }

    scheduledInstanceRepository.saveAll(scheduledInstances);

    // inform the Job microservice about a success!
    restTemplate.postForEntity( url: jobsUrl + "/updateStatus",
            new UpdateJob(j.getJobId(),  status: "scheduled", scheduledInstances.get(0).getDate()), Void.class);
}
```

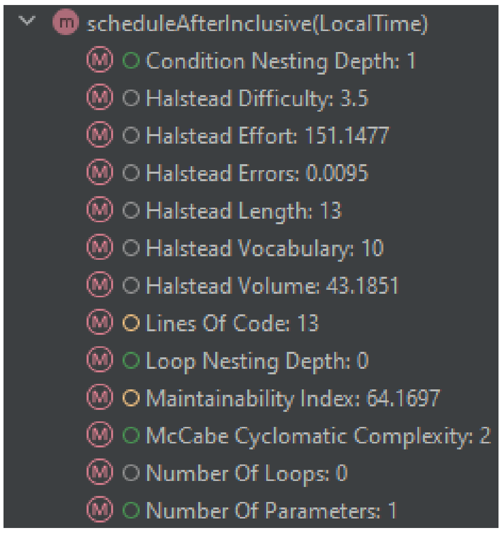


```java
public void verifyCpuBiggerThanMaxOfGpuOrMemory(ScheduleJob j) throws ResourceBiggerThanCpuException {
    // verify the CPU >= Max(GPU, Memory) requirement
    if (j.getCpuUsage() < Math.max(j.getGpuUsage(), j.getMemoryUsage())) {
        throw new ResourceBiggerThanCpuException("GPU or Memory");
    }
}
```

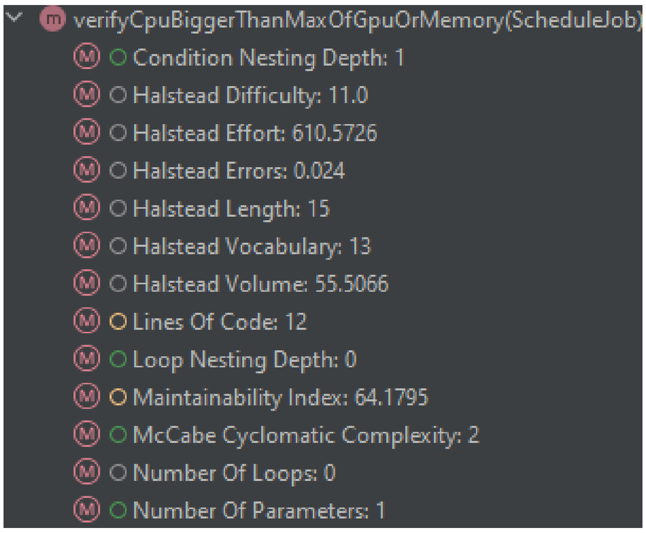


```java
public LocalDate scheduleAfterInclusive(LocalTime currentTime) {
    if (isFiveMinutesBeforeDayStarts(currentTime)) {
        return LocalDate.now().plusDays(2);
    } else {
        return LocalDate.now().plusDays(1);
    }
}
```

The two newly created methods also have good metrics, thus the refactoring worked well. The metrics are in the following two images:



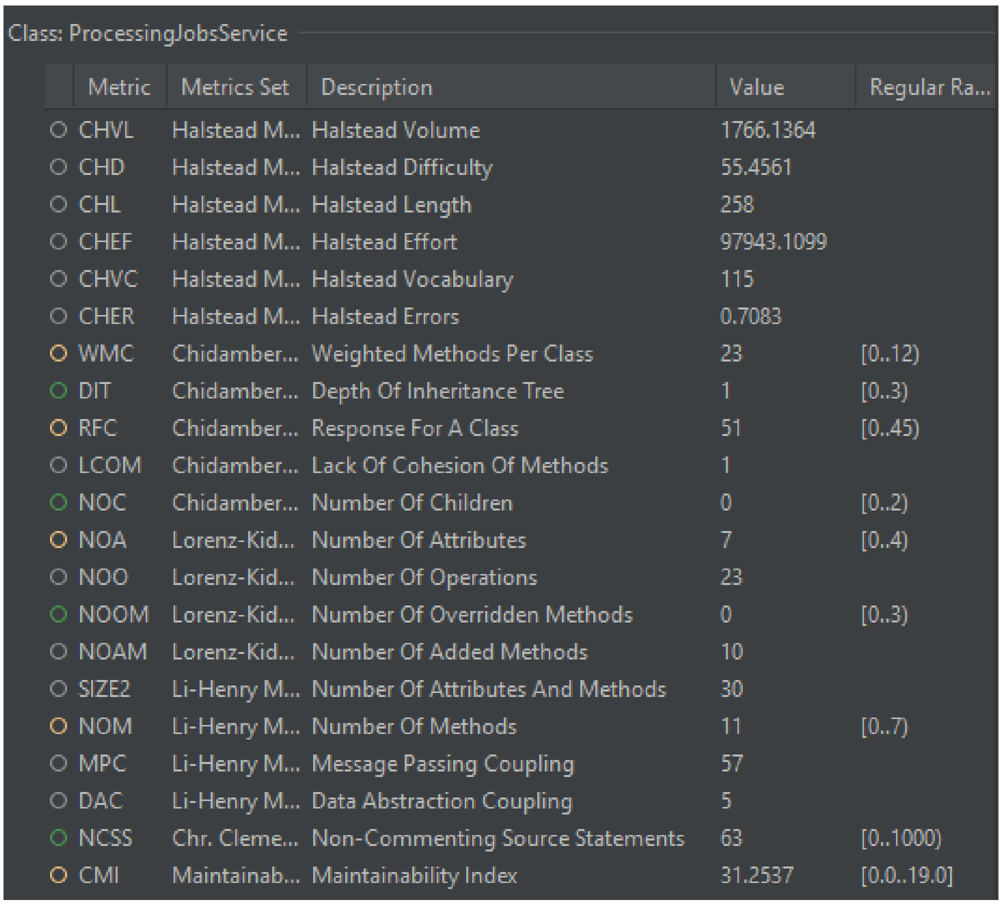
```
scheduleAfterInclusive(LocalTime)
    Condition Nesting Depth: 1
    Halstead Difficulty: 3.5
    Halstead Effort: 151.1477
    Halstead Errors: 0.0095
    Halstead Length: 13
    Halstead Vocabulary: 10
    Halstead Volume: 43.1851
    Lines Of Code: 13
    Loop Nesting Depth: 0
    Maintainability Index: 64.1697
    McCabe Cyclomatic Complexity: 2
    Number Of Loops: 0
    Number Of Parameters: 1
```



```
verifyCpuBiggerThanMaxOfGpuOrMemory(ScheduleJob)
    Condition Nesting Depth: 1
    Halstead Difficulty: 11.0
    Halstead Effort: 610.5726
    Halstead Errors: 0.024
    Halstead Length: 15
    Halstead Vocabulary: 13
    Halstead Volume: 55.5066
    Lines Of Code: 12
    Loop Nesting Depth: 0
    Maintainability Index: 64.1795
    McCabe Cyclomatic Complexity: 2
    Number Of Loops: 0
    Number Of Parameters: 1
```

Class: ProcessingJobService

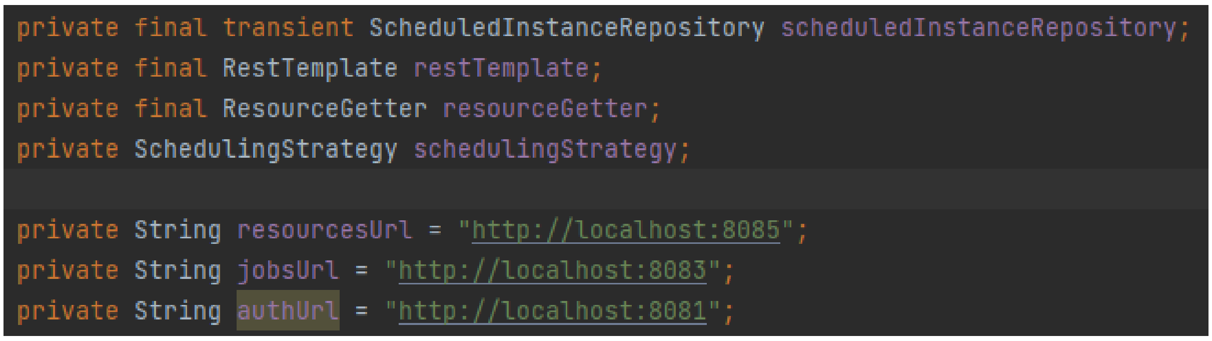
The ProcessingJobService class was considered for the refactoring, since some improvements could have been made according to the metrics. For example, the WMC, RFC, NOA and NOM could be better (yellow marked). The metrics before refactoring can be seen below.



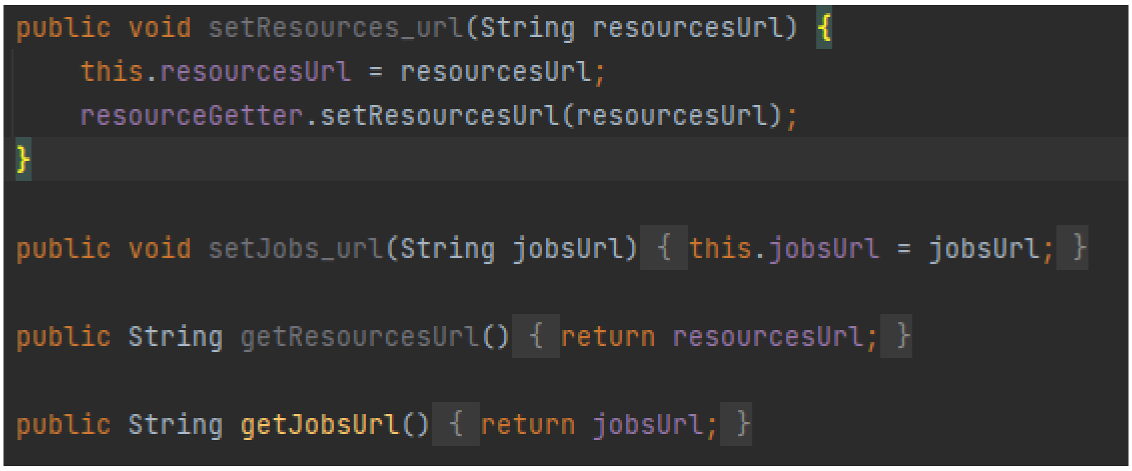
Class: ProcessingJobsService

	Metric	Metrics Set	Description	Value	Regular Ra...
○	CHVL	Halstead M...	Halstead Volume	1766.1364	
○	CHD	Halstead M...	Halstead Difficulty	55.4561	
○	CHL	Halstead M...	Halstead Length	258	
○	CHEF	Halstead M...	Halstead Effort	97943.1099	
○	CHVC	Halstead M...	Halstead Vocabulary	115	
○	CHER	Halstead M...	Halstead Errors	0.7083	
○	WMC	Chidamber...	Weighted Methods Per Class	23	[0..12)
○	DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
○	RFC	Chidamber...	Response For A Class	51	[0..45)
○	LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
○	NOC	Chidamber...	Number Of Children	0	[0..2)
○	NOA	Lorenz-Kid...	Number Of Attributes	7	[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	10	
○	SIZE2	Li-Henry M...	Number Of Attributes And Methods	30	
○	NOM	Li-Henry M...	Number Of Methods	11	[0..7)
○	MPC	Li-Henry M...	Message Passing Coupling	57	
○	DAC	Li-Henry M...	Data Abstraction Coupling	5	
○	NCSS	Chr. Cleme...	Non-Commenting Source Statements	63	[0..1000)
○	CMI	Maintainab...	Maintainability Index	31.2537	[0.0..19.0]

In order to refactor this class, the "Extract Class" technique has been applied twice. First, the class contained three String values with the url path to other microservices as well as some getter and setter methods to change those url paths. The following two images demonstrate those code snippets.



```
private final transient ScheduledInstanceRepository scheduledInstanceRepository;
private final RestTemplate restTemplate;
private final ResourceGetter resourceGetter;
private SchedulingStrategy schedulingStrategy;

private String resourcesUrl = "http://localhost:8085";
private String jobsUrl = "http://localhost:8083";
private String authUrl = "http://localhost:8081";
```

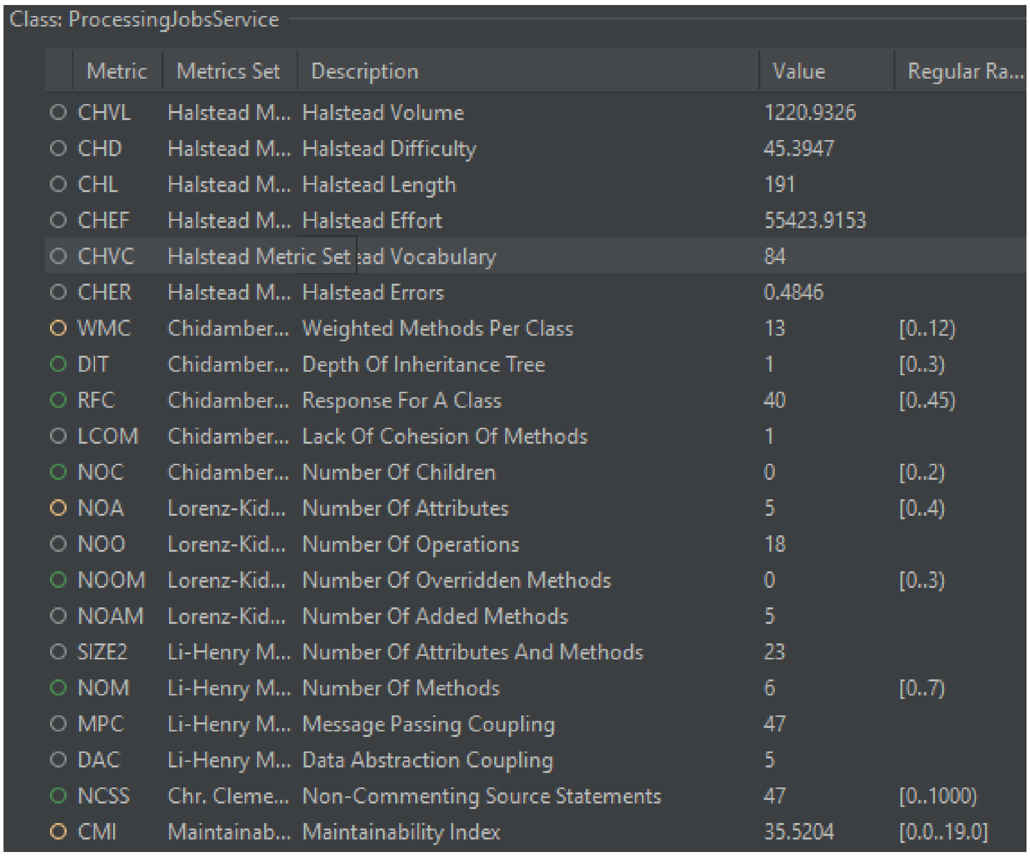


```
public void setResources_url(String resourcesUrl) {
    this.resourcesUrl = resourcesUrl;
    resourceGetter.setResourcesUrl(resourcesUrl);
}

public void setJobs_url(String jobsUrl) { this.jobsUrl = jobsUrl; }

public String getResourcesUrl() { return resourcesUrl; }

public String getJobsUrl() { return jobsUrl; }
```

This increases unnecessarily the NOA, NOM and RFC. Therefore, a class called “Url” was created, which stores all url paths to the microservices. This is stored in the “commons” folder, which is accessible for all microservices, since multiple microservices need to have access to the url paths.

Furthermore, the “Extract Class” technique has been applied again to further reduce the NOA, NOM, RFC and WMC. This was done by creating a new class called “SchedulingCheckService”, which will contain the methods that the “ProcessingJobsService” used to check certain values before something can be scheduled. For instance, “verifyCpuBiggerThanMaxOfGpuOrMemory()” and “isFiveMinutesBeforeDayStarts()”.

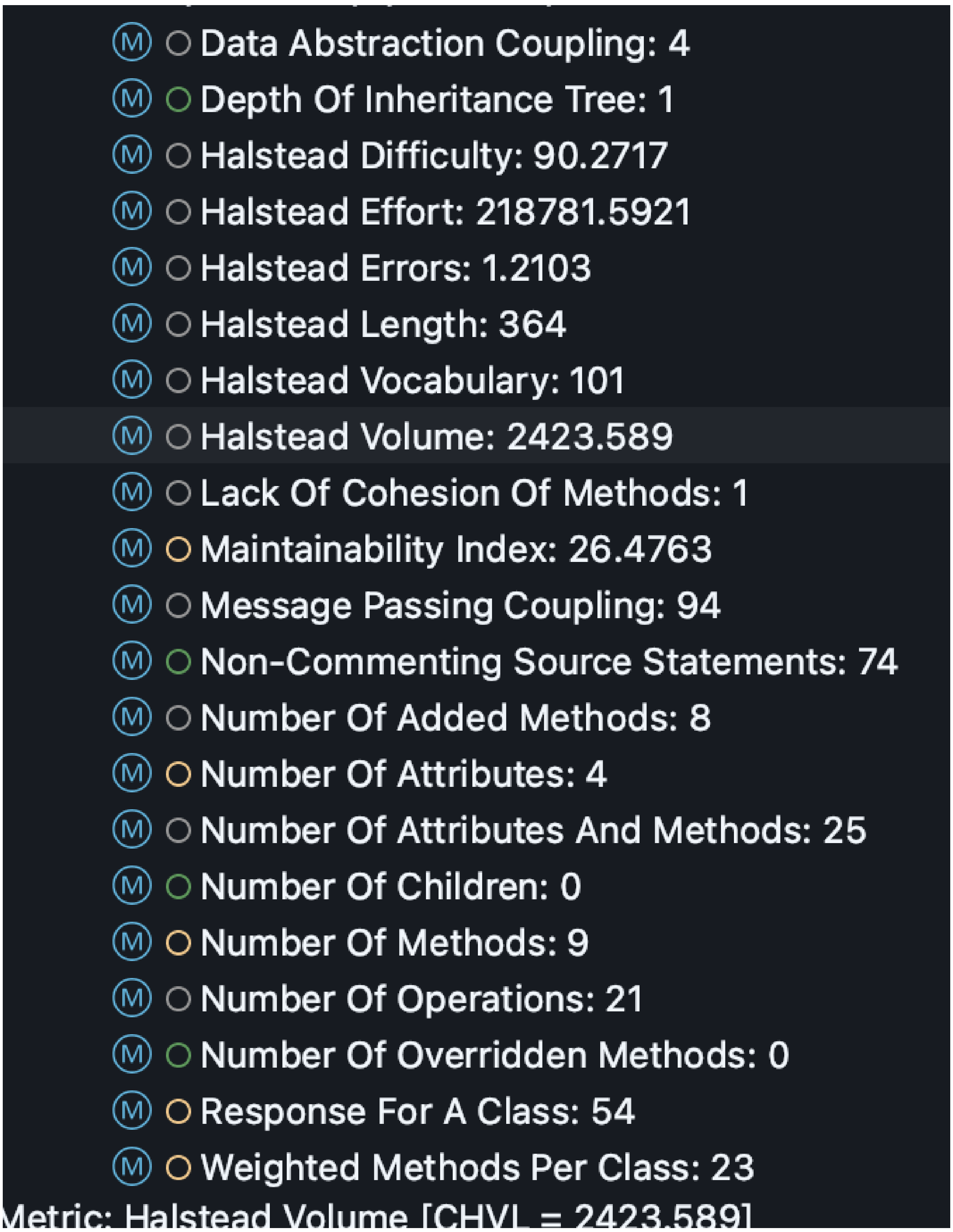
All of that has decreased the WMC from 23 to 13, RFC from 51 to 40, NOA from 7 to 5 and NOM from 11 to 6, as can be seen in the following image.



Class: ProcessingJobsService

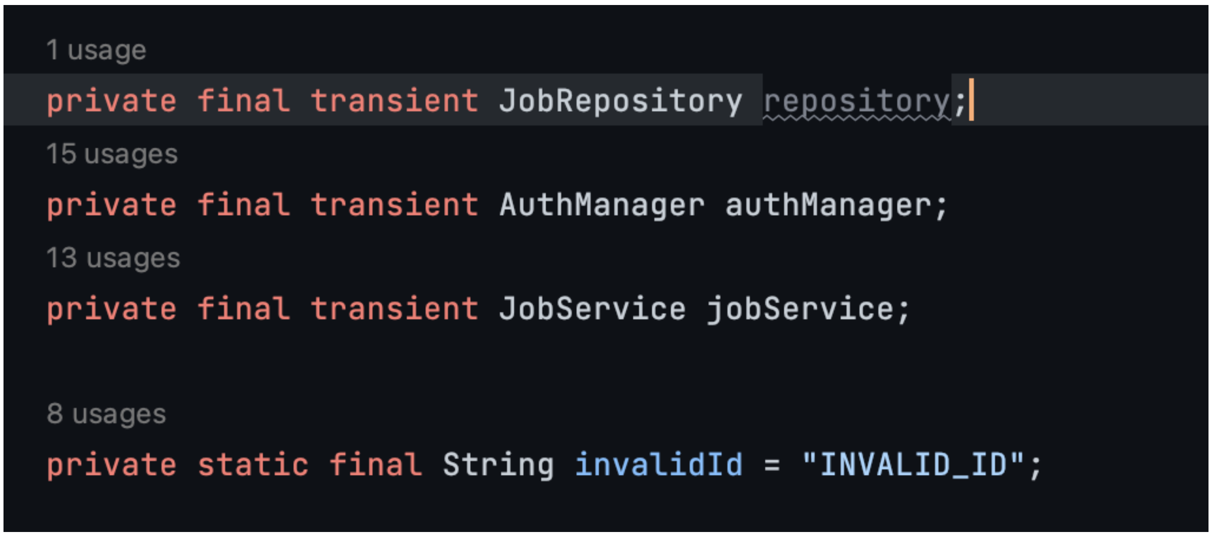
	Metric	Metrics Set	Description	Value	Regular Ra...
○	CHVL	Halstead M...	Halstead Volume	1220.9326	
○	CHD	Halstead M...	Halstead Difficulty	45.3947	
○	CHL	Halstead M...	Halstead Length	191	
○	CHEF	Halstead M...	Halstead Effort	55423.9153	
○	CHVC	Halstead Metric Set	ead Vocabulary	84	
○	CHER	Halstead M...	Halstead Errors	0.4846	
○	WMC	Chidamber...	Weighted Methods Per Class	13	[0..12)
○	DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
○	RFC	Chidamber...	Response For A Class	40	[0..45)
○	LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
○	NOC	Chidamber...	Number Of Children	0	[0..2)
○	NOA	Lorenz-Kid...	Number Of Attributes	5	[0..4)
○	NOO	Lorenz-Kid...	Number Of Operations	18	
○	NOOM	Lorenz-Kid...	Number Of Overridden Methods	0	[0..3)
○	NOAM	Lorenz-Kid...	Number Of Added Methods	5	
○	SIZE2	Li-Henry M...	Number Of Attributes And Methods	23	
○	NOM	Li-Henry M...	Number Of Methods	6	[0..7)
○	MPC	Li-Henry M...	Message Passing Coupling	47	
○	DAC	Li-Henry M...	Data Abstraction Coupling	5	
○	NCSS	Chr. Cleme...	Non-Commenting Source Statements	47	[0..1000)
○	CMI	Maintainab...	Maintainability Index	35.5204	[0.0..19.0]

Class: JobController



```
Ⓜ ○ Data Abstraction Coupling: 4
Ⓜ ○ Depth Of Inheritance Tree: 1
Ⓜ ○ Halstead Difficulty: 90.2717
Ⓜ ○ Halstead Effort: 218781.5921
Ⓜ ○ Halstead Errors: 1.2103
Ⓜ ○ Halstead Length: 364
Ⓜ ○ Halstead Vocabulary: 101
Ⓜ ○ Halstead Volume: 2423.589
Ⓜ ○ Lack Of Cohesion Of Methods: 1
Ⓜ ○ Maintainability Index: 26.4763
Ⓜ ○ Message Passing Coupling: 94
Ⓜ ○ Non-Commenting Source Statements: 74
Ⓜ ○ Number Of Added Methods: 8
Ⓜ ○ Number Of Attributes: 4
Ⓜ ○ Number Of Attributes And Methods: 25
Ⓜ ○ Number Of Children: 0
Ⓜ ○ Number Of Methods: 9
Ⓜ ○ Number Of Operations: 21
Ⓜ ○ Number Of Overridden Methods: 0
Ⓜ ○ Response For A Class: 54
Ⓜ ○ Weighted Methods Per Class: 23
Metric: Halstead Volume [CHVL = 2423.589]
```

The first class metric that could be easily solved to improve the JobController class is Number of Attributes. At the moment there are 4 attributes: repository, authManager, jobService and invalidId.

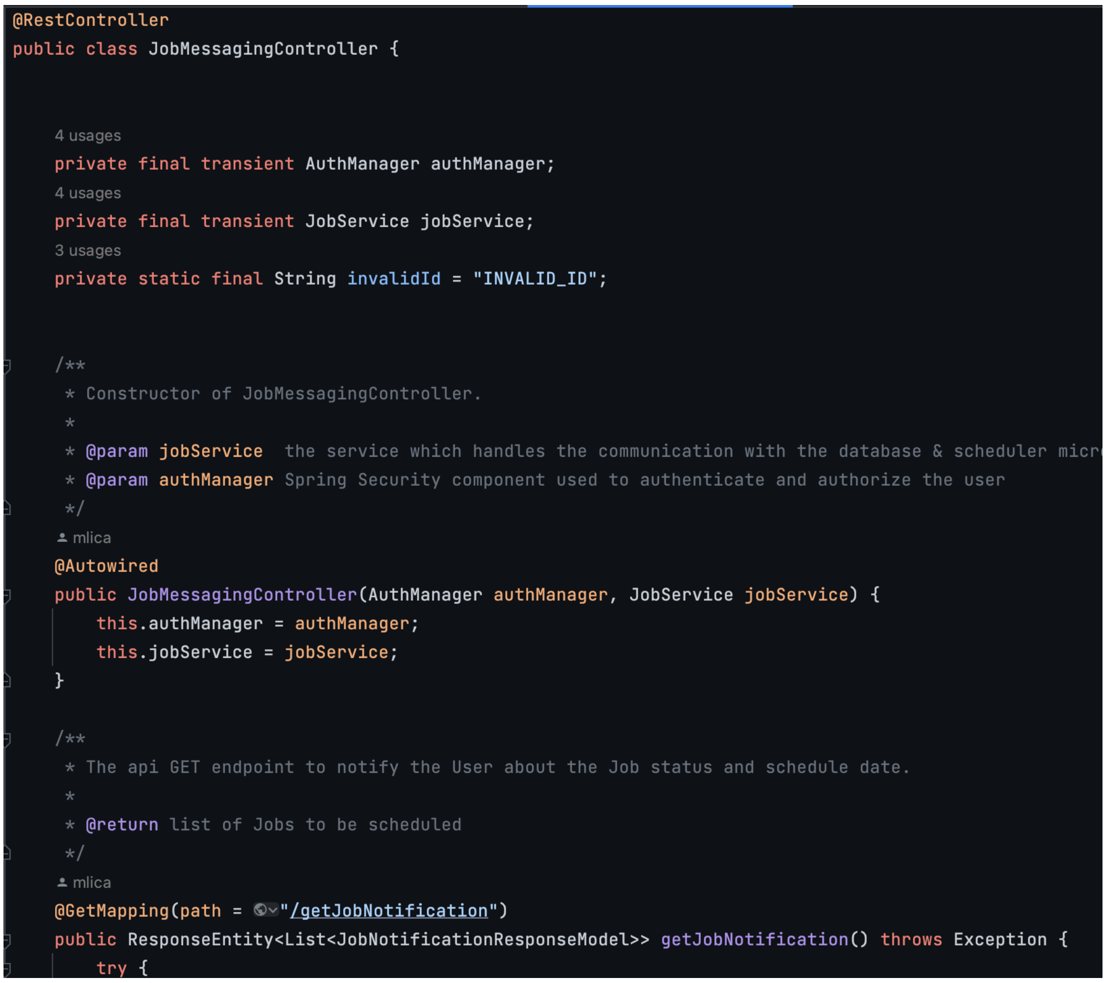


```
1 usage
private final transient JobRepository repository;
15 usages
private final transient AuthManager authManager;
13 usages
private final transient JobService jobService;

8 usages
private static final String invalidId = "INVALID_ID";
```

As seen in the picture, the repository class is not used so we can easily eliminate it and improve the metric.

The second metric we should fix is the number of methods in the class which is 9, at the moment. As the class is a controller in our application, a solution for this would be to create another controller for related methods that can be decoupled from the rest.

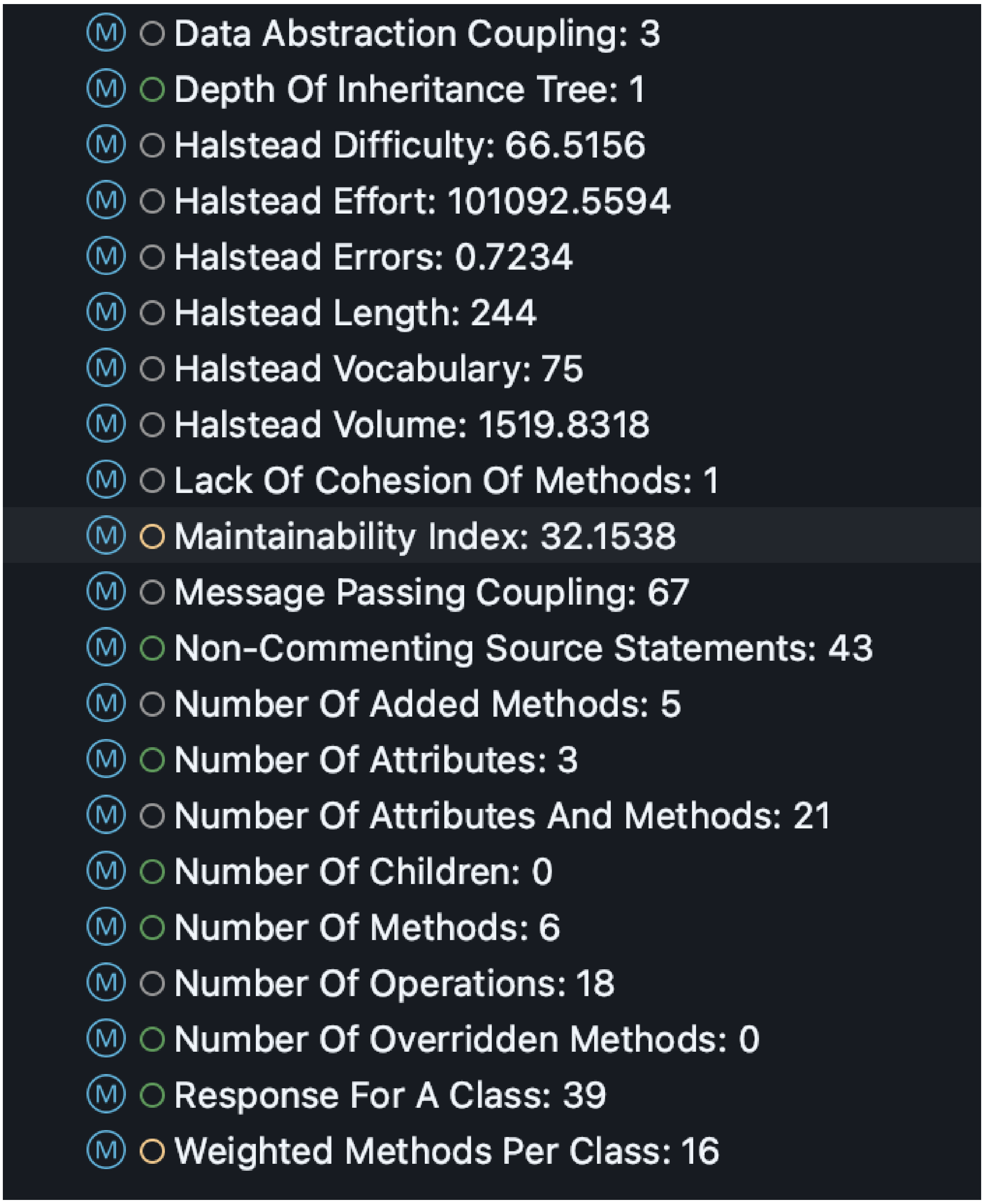
Looking at all the methods we see that the endpoint *jobStatus* ,*getJobNotification* and can be moved to a separate controller



```
@RestController
public class JobMessagingController {


    4 usages
    private final transient AuthManager authManager;
    4 usages
    private final transient JobService jobService;
    3 usages
    private static final String invalidId = "INVALID_ID";


    /**
     * Constructor of JobMessagingController.
     *
     * @param jobService  the service which handles the communication with the database & scheduler micr
     * @param authManager Spring Security component used to authenticate and authorize the user
     */
    mlica
    @Autowired
    public JobMessagingController(AuthManager authManager, JobService jobService) {
        this.authManager = authManager;
        this.jobService = jobService;
    }

    /**
     * The api GET endpoint to notify the User about the Job status and schedule date.
     *
     * @return list of Jobs to be scheduled
     */
    mlica
    @GetMapping(path = "/getJobNotification")
    public ResponseEntity<List<JobNotificationResponseModel>> getJobNotification() throws Exception {
        try {
```

This is the new controller which contains the 3 endpoints taken from the JobController class in order to solve the *Number of Methods* metric.
Once we fixed the number of methods in the JobController class, the metric Response of a Class also got an optimal value of 39 (initial value was 54).

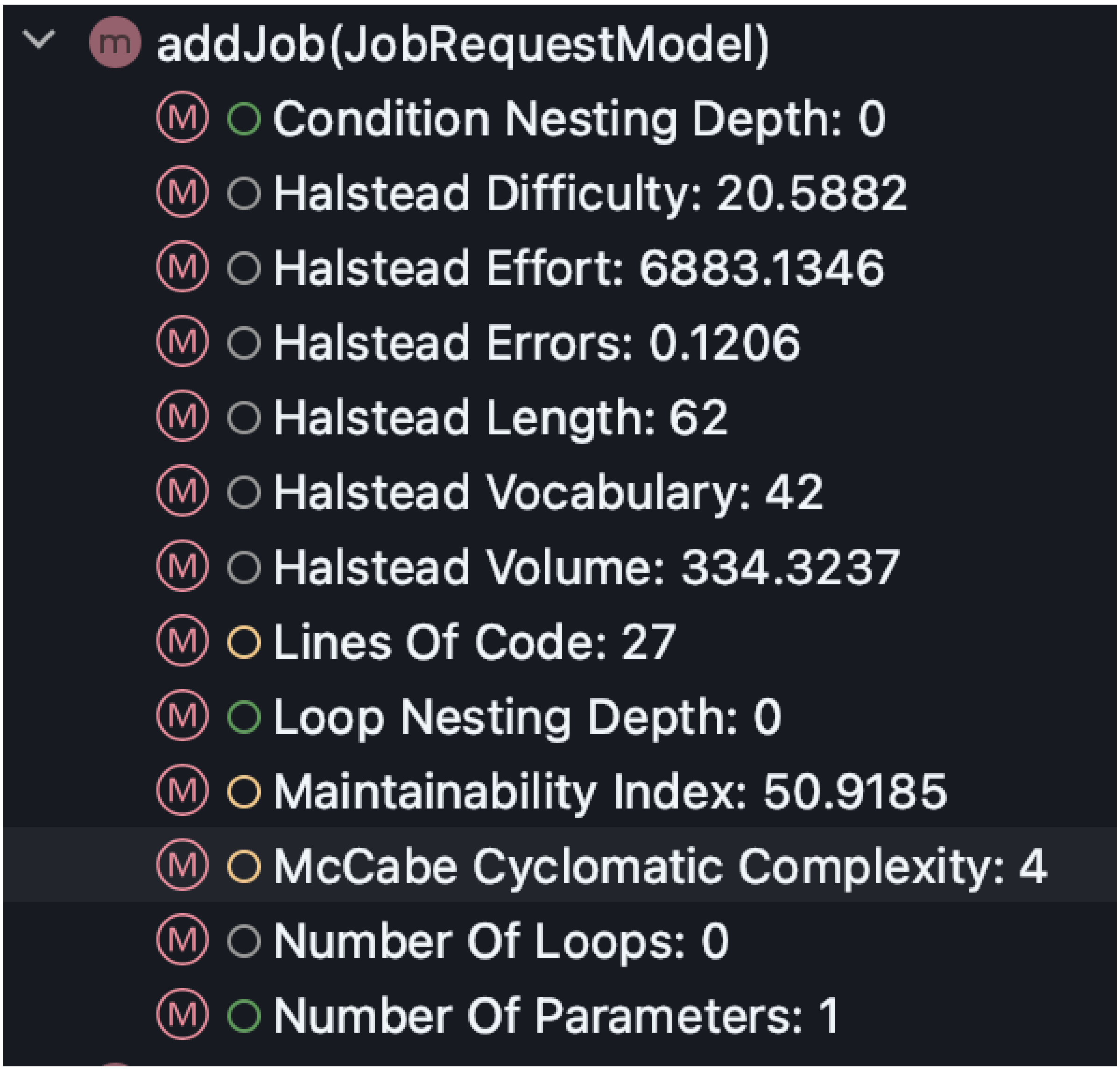
In the JobController class, there are 2 other metrics that are not colored green (higher value than needed). However, Weighted Methods Per Class and Maintainability Index, have a value extremely close to the optimal one. The value for Weighted Methods per Class is 16 where the interval for Good is between 0 and 12. Moreover, the Maintainability Index has a value of 32.1538 but it is mostly related to refactoring the methods inside the class which is not our goal. As a reference the optimal value for this metric should be between 0 and 19.



```
Data Abstraction Coupling: 3
Depth Of Inheritance Tree: 1
Halstead Difficulty: 66.5156
Halstead Effort: 101092.5594
Halstead Errors: 0.7234
Halstead Length: 244
Halstead Vocabulary: 75
Halstead Volume: 1519.8318
Lack Of Cohesion Of Methods: 1
Maintainability Index: 32.1538
Message Passing Coupling: 67
Non-Commenting Source Statements: 43
Number Of Added Methods: 5
Number Of Attributes: 3
Number Of Attributes And Methods: 21
Number Of Children: 0
Number Of Methods: 6
Number Of Operations: 18
Number Of Overridden Methods: 0
Response For A Class: 39
Weighted Methods Per Class: 16
```

This is the final report of the JobController class after applying proper refactoring techniques and improving most of the metrics provided by MetricsTree plugin at class level.

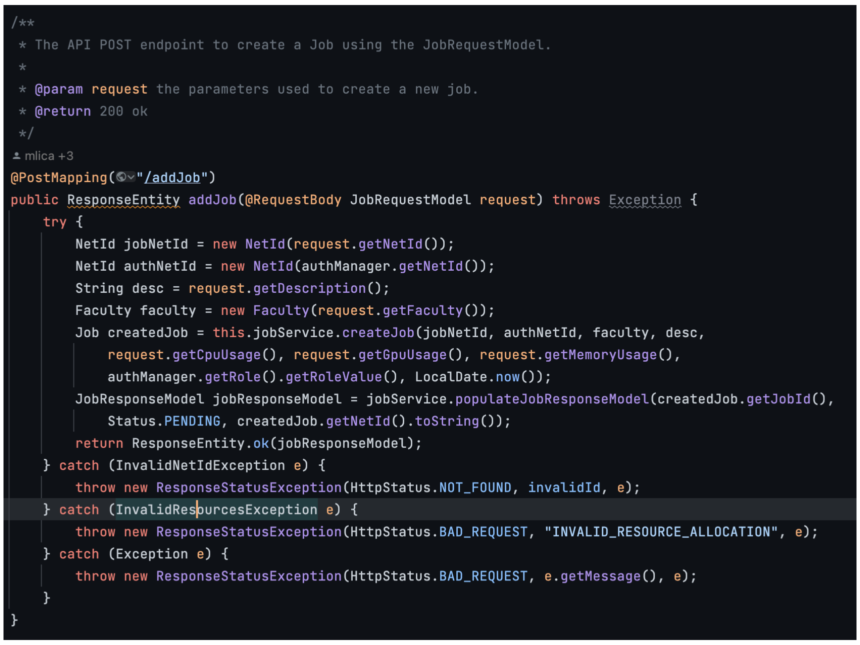
Job Controller/addJob method



```
addJob(JobRequestModel)
    Condition Nesting Depth: 0
    Halstead Difficulty: 20.5882
    Halstead Effort: 6883.1346
    Halstead Errors: 0.1206
    Halstead Length: 62
    Halstead Vocabulary: 42
    Halstead Volume: 334.3237
    Lines Of Code: 27
    Loop Nesting Depth: 0
    Maintainability Index: 50.9185
    McCabe Cyclomatic Complexity: 4
    Number Of Loops: 0
    Number Of Parameters: 1
```

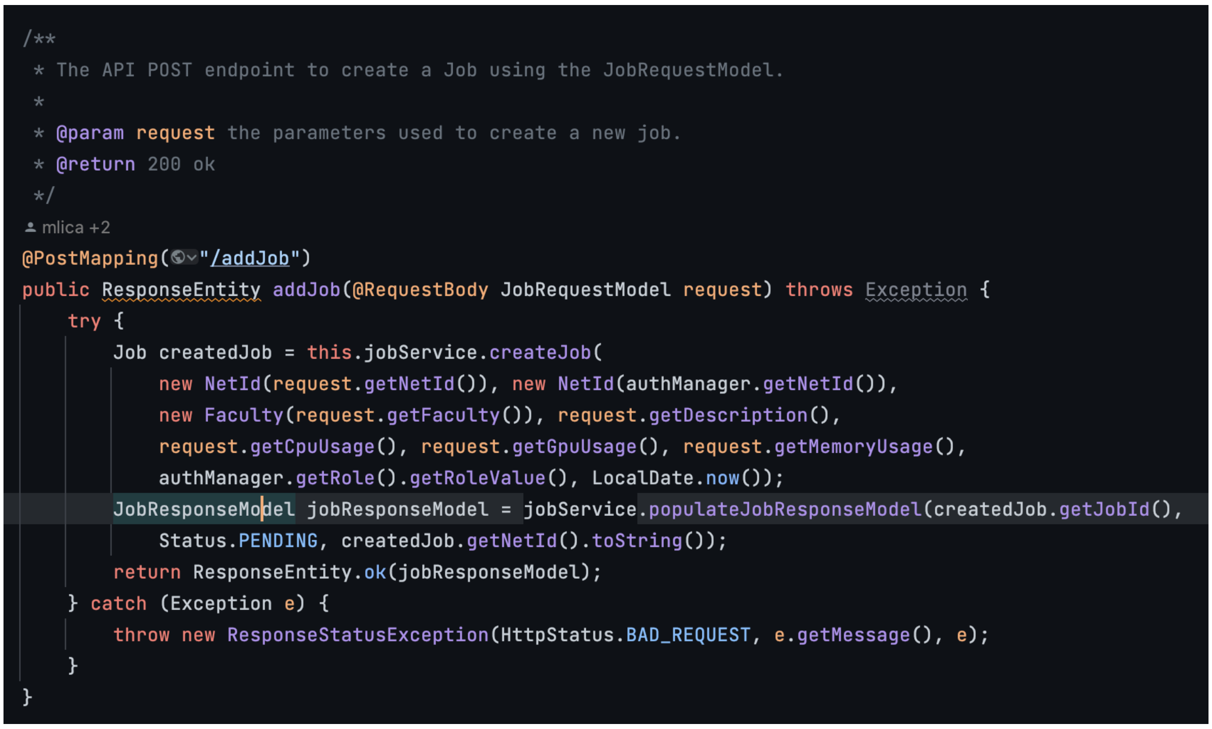
According to the MetricsTree plugin the method/endpoint addJob in JobController class could be improved in three metrics: Lines of Code, Maintainability Index and Cyclomatic Complexity.

This is the method before any refactoring changes:



```java
/**
 * The API POST endpoint to create a Job using the JobRequestModel.
 *
 * @param request the parameters used to create a new job.
 * @return 200 ok
 */
mlica +3
@PostMapping("/addJob")
public ResponseEntity addJob(@RequestBody JobRequestModel request) throws Exception {
    try {
        NetId jobNetId = new NetId(request.getNetId());
        NetId authNetId = new NetId(authManager.getNetId());
        String desc = request.getDescription();
        Faculty faculty = new Faculty(request.getFaculty());
        Job createdJob = this.jobService.createJob(jobNetId, authNetId, faculty, desc,
            request.getCpuUsage(), request.getGpuUsage(), request.getMemoryUsage(),
            authManager.getRole().getRoleValue(), LocalDate.now());
        JobResponseModel jobResponseModel = jobService.populateJobResponseModel(createdJob.getJobId(),
            Status.PENDING, createdJob.getNetId().toString());
        return ResponseEntity.ok(jobResponseModel);
    } catch (InvalidNetIdException e) {
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, invalidId, e);
    } catch (InvalidResourcesException e) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "INVALID_RESOURCE_ALLOCATION", e);
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage(), e);
    }
}
```

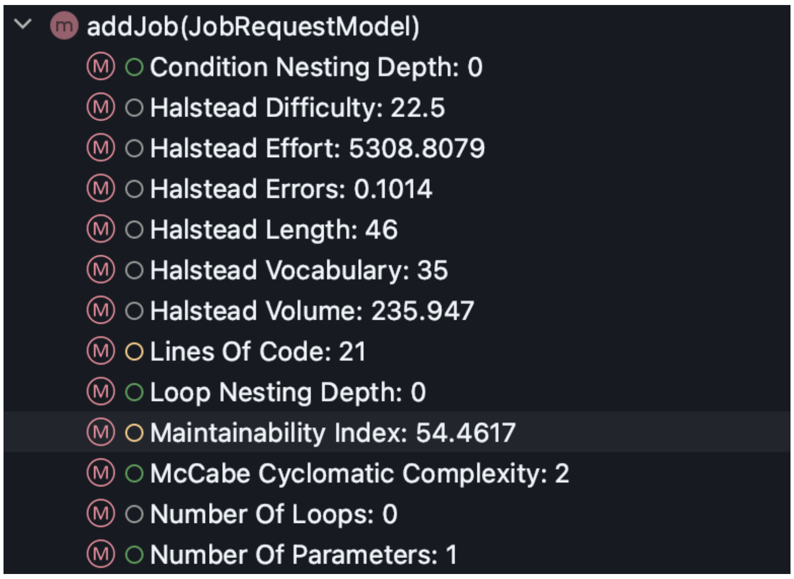
One way to improve both the cyclomatic complexity metric and the lines of code is to eliminate the first two catch statements in the function. This change is possible because both exceptions InvalidNetIdExecption and InvalidResourcesException are subclasses of the Exception class. Following this observation, the last catch of the method will be able to handle all exceptions provided.



```java
/**
 * The API POST endpoint to create a Job using the JobRequestModel.
 *
 * @param request the parameters used to create a new job.
 * @return 200 ok
 */
mlica +2
@PostMapping("/addJob")
public ResponseEntity addJob(@RequestBody JobRequestModel request) throws Exception {
    try {
        Job createdJob = this.jobService.createJob(
            new NetId(request.getNetId()), new NetId(authManager.getNetId()),
            new Faculty(request.getFaculty()), request.getDescription(),
            request.getCpuUsage(), request.getGpuUsage(), request.getMemoryUsage(),
            authManager.getRole().getRoleValue(), LocalDate.now());
        JobResponseModel jobResponseModel = jobService.populateJobResponseModel(createdJob.getJobId(),
            Status.PENDING, createdJob.getNetId().toString());
        return ResponseEntity.ok(jobResponseModel);
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage(), e);
    }
}
```

After removing the catch statement and compressing some lines of code when calling createJob method, we managed to get an optimal value for the cyclomatic complexity metric (value of two) and reduced the lines of code from 27 to 21. However, the Maintainability Index increased to 54.46 from 50. Overall, the refactoring improved the addJob method.

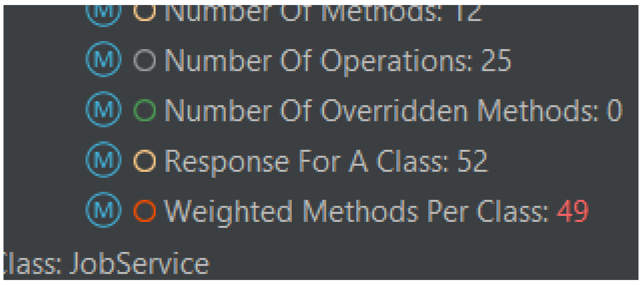
The statistics provided by MetricsTree after refactoring the method.



```
addJob(JobRequestModel)
    Condition Nesting Depth: 0
    Halstead Difficulty: 22.5
    Halstead Effort: 5308.8079
    Halstead Errors: 0.1014
    Halstead Length: 46
    Halstead Vocabulary: 35
    Halstead Volume: 235.947
    Lines Of Code: 21
    Loop Nesting Depth: 0
    Maintainability Index: 54.4617
    McCabe Cyclomatic Complexity: 2
    Number Of Loops: 0
    Number Of Parameters: 1
```

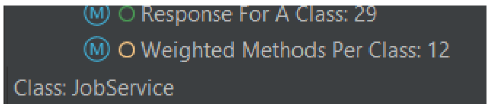
Class: JobService

Job service class has an exceptionally high WMC metric. This is because the class is way too long has many methods with high CC.

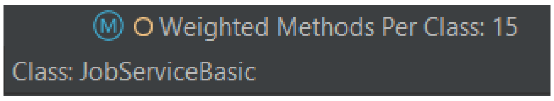


```
    Number Of Methods: 12
    Number Of Operations: 25
    Number Of Overridden Methods: 0
    Response For A Class: 52
    Weighted Methods Per Class: 49
lass: JobService
```

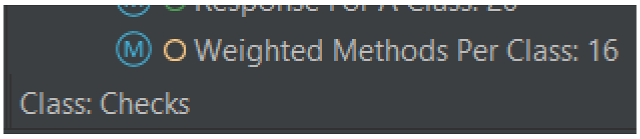
I have decided to solve this by separating this class into two: JobService, JobServiceBasic. JobServiceBasic handles more basic requests like create, delete or schedule a job while JobService handles more complex requests that work with multiple jobs at once, for example collectAllJobsByNetId, or getAllJobs. At the end I have edded one more class called Checks that does all the checks and throws an exception if the check is not met.



```
    Response For A Class: 29
    Weighted Methods Per Class: 12
Class: JobService
```



```
    Weighted Methods Per Class: 15
Class: JobServiceBasic
```

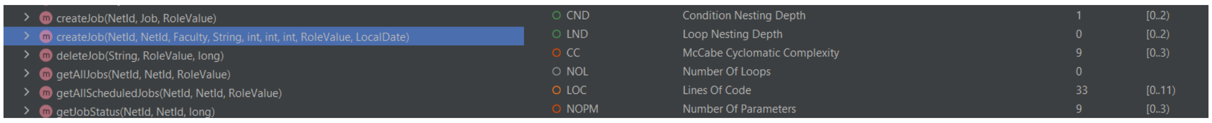


```
    Weighted Methods Per Class: 16
Class: Checks
```

WMC metric seems to be improved.

Method: createJob

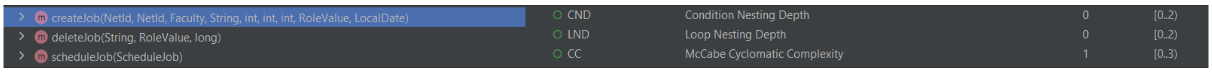
Has high cyclomatic complexity and the number of parameters. This can be improved by introducing a class called Checks that does all the checks and throws an exception if the check is not met.



createJob(NetId, Job, RoleValue)
createJob(NetId, NetId, Faculty, String, int, int, int, RoleValue, LocalDate)
deleteJob(String, RoleValue, long)
getAllJobs(NetId, NetId, RoleValue)
getAllScheduledJobs(NetId, NetId, RoleValue)
getJobStatus(NetId, NetId, long)

CND	Condition Nesting Depth	1	[0..2)
LND	Loop Nesting Depth	0	[0..2)
CC	McCabe Cyclomatic Complexity	9	[0..3)
NOL	Number Of Loops	0	
LOC	Lines Of Code	33	[0..11)
NOPM	Number Of Parameters	9	[0..3)

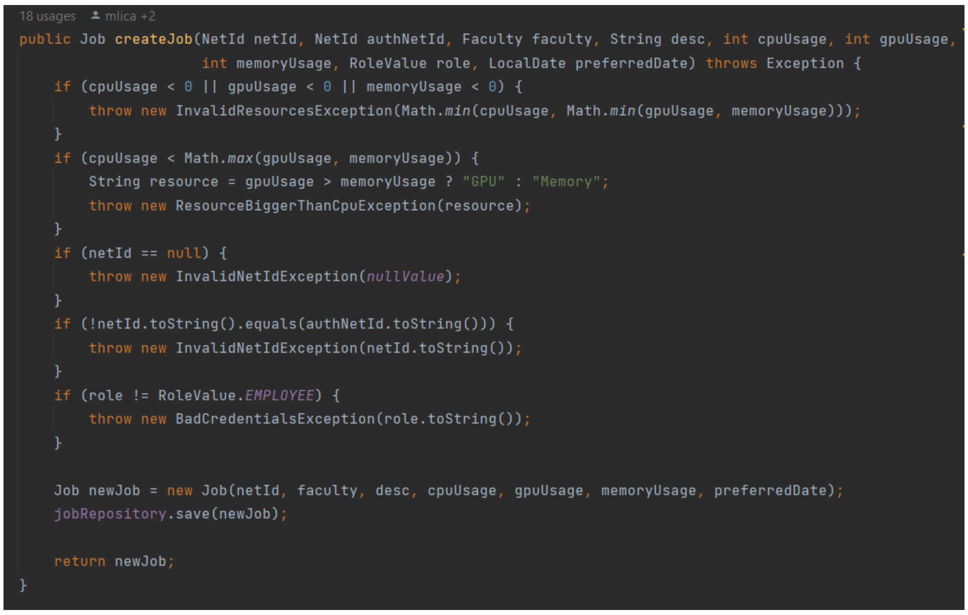
By introducing the Checks class that does the checks for me, I dont need to have if statements and useless paths in my method I can just call a function that either passes and returns nothing or throws an exception. The CC has reduced.



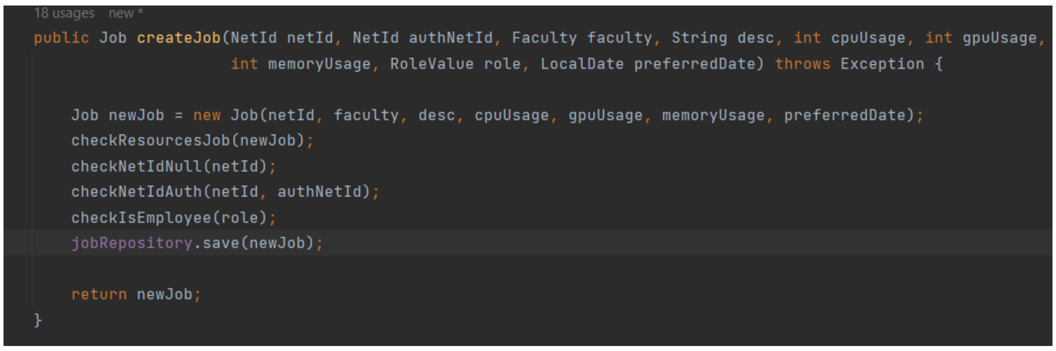
createJob(NetId, NetId, Faculty, String, int, int, int, RoleValue, LocalDate)
deleteJob(String, RoleValue, long)
scheduleJob(ScheduleJob)

CND	Condition Nesting Depth	0	[0..2)
LND	Loop Nesting Depth	0	[0..2)
CC	McCabe Cyclomatic Complexity	1	[0..3)

Class createJob before and after refactoring.



```
18 usages  mlica +2
public Job createJob(NetId netId, NetId authNetId, Faculty faculty, String desc, int cpuUsage, int gpuUsage,
                     int memoryUsage, RoleValue role, LocalDate preferredDate) throws Exception {
    if (cpuUsage < 0 || gpuUsage < 0 || memoryUsage < 0) {
        throw new InvalidResourcesException(Math.min(cpuUsage, Math.min(gpuUsage, memoryUsage)));
    }
    if (cpuUsage < Math.max(gpuUsage, memoryUsage)) {
        String resource = gpuUsage > memoryUsage ? "GPU" : "Memory";
        throw new ResourceBiggerThanCpuException(resource);
    }
    if (netId == null) {
        throw new InvalidNetIdException(nullValue);
    }
    if (!netId.toString().equals(authNetId.toString())) {
        throw new InvalidNetIdException(netId.toString());
    }
    if (role != RoleValue.EMPLOYEE) {
        throw new BadCredentialsException(role.toString());
    }

    Job newJob = new Job(netId, faculty, desc, cpuUsage, gpuUsage, memoryUsage, preferredDate);
    jobRepository.save(newJob);

    return newJob;
}
```

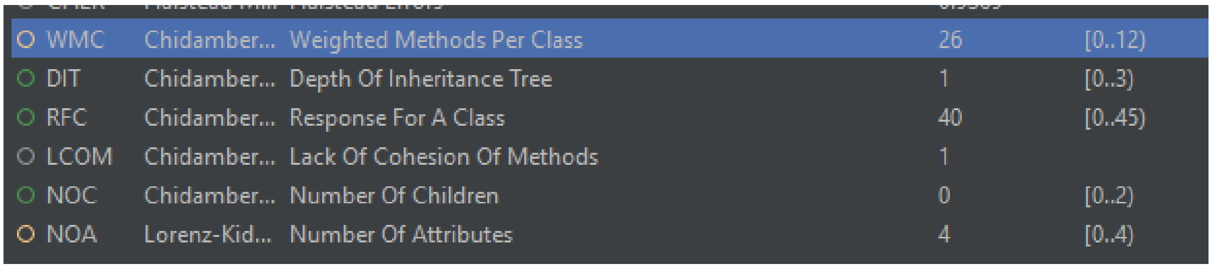


```
18 usages  new *
public Job createJob(NetId netId, NetId authNetId, Faculty faculty, String desc, int cpuUsage, int gpuUsage,
                     int memoryUsage, RoleValue role, LocalDate preferredDate) throws Exception {

    Job newJob = new Job(netId, faculty, desc, cpuUsage, gpuUsage, memoryUsage, preferredDate);
    checkResourcesJob(newJob);
    checkNetIdNull(netId);
    checkNetIdAuth(netId, authNetId);
    checkIsEmployee(role);
    jobRepository.save(newJob);

    return newJob;
}
```

We have decided to leave all the parameters in the method there because they are needed for the job construction.

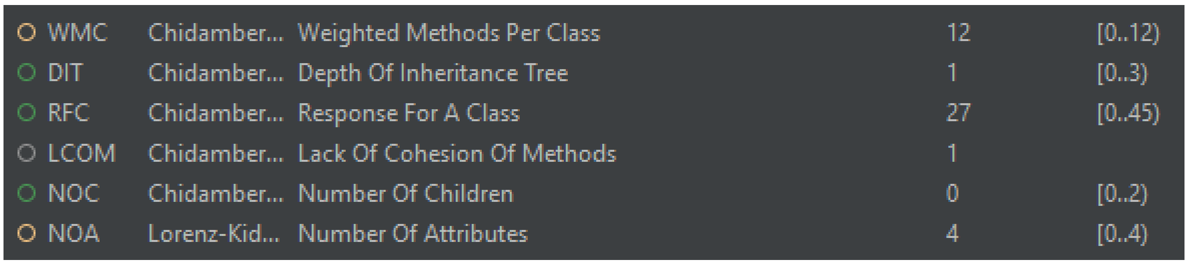
Class: UpdatingJobsService

The UpdatingJobsService class had functionality related to updating scheduled jobs, as well as rescheduling them in the event of a change in the amount of resources. The metrics for the class were as shown on the following image:



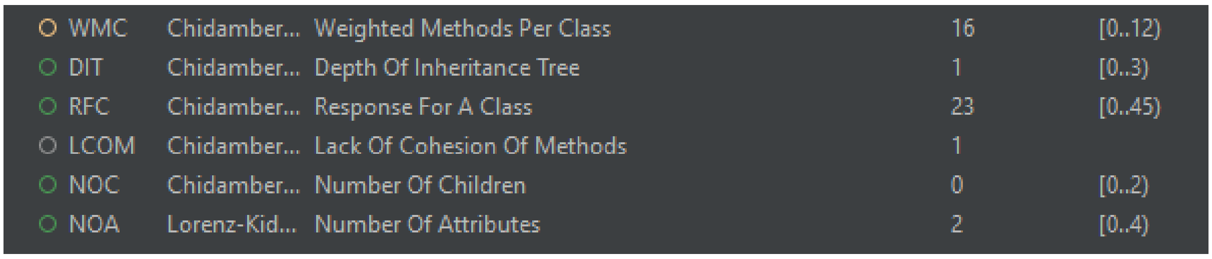
○	WMC	Chidamber...	Weighted Methods Per Class	26	[0..12)
○	DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
○	RFC	Chidamber...	Response For A Class	40	[0..45)
○	LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
○	NOC	Chidamber...	Number Of Children	0	[0..2)
○	NOA	Lorenz-Kid...	Number Of Attributes	4	[0..4)

We can see two metrics (WMC and NOA) that are yellow. All of the other ones (including the metrics not shown on this image) were green. There was not much to improve in this class, but the Weighted Methods Per Class metric indicates that there is too much complexity in the methods of this class, and so it may be difficult to maintain in the future.
I decided to separate the updating logic from logic that removes scheduled jobs from the database when there is a change in the amount of resources. I created another service class ExcessRemovalService and moved two methods (reduceExcess and recreateScheduleJobFromScheduledInstances) to the new class. The new, improved metric for UpdatingJobsService is the following:



○	WMC	Chidamber...	Weighted Methods Per Class	12	[0..12)
○	DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
○	RFC	Chidamber...	Response For A Class	27	[0..45)
○	LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
○	NOC	Chidamber...	Number Of Children	0	[0..2)
○	NOA	Lorenz-Kid...	Number Of Attributes	4	[0..4)

The NOA metric has not improved as this class needs access to methods from 4 other classes, but the WMC metric was reduced from 26 to 12.
The metric for the ExcessRemovalService is also optimistic:



○	WMC	Chidamber...	Weighted Methods Per Class	16	[0..12)
○	DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
○	RFC	Chidamber...	Response For A Class	23	[0..45)
○	LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
○	NOC	Chidamber...	Number Of Children	0	[0..2)
○	NOA	Lorenz-Kid...	Number Of Attributes	2	[0..4)

Its WMC is 16 - it is not much above the recommended bound and all of the other metrics are green. It is worth mentioning that by splitting the class into two simpler ones, other already good metrics have improved as well.