SEM 13a - DelftBlue assignment 2

Michiel Bakker - Bink Boëtius - Ionut Ciobanu - Ivans Kravcevs - Nils Mikk

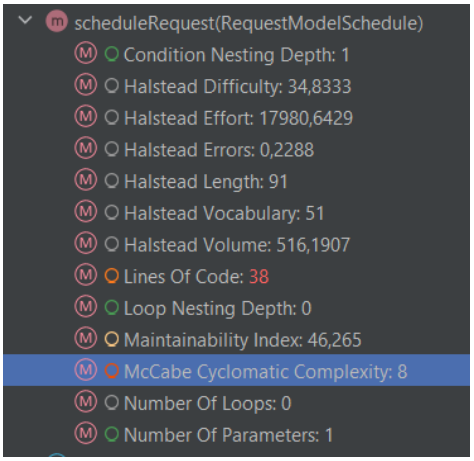
For this assignment, the MetricsTree IntelliJ plugin is used to compute the code metrics at both the method level and the class level. Using these results, we identified multiple methods and classes that could be improved by applying code refactoring. The methods and classes we have improved, including the reasoning as to why and how we improved them, are covered in this document.

Method level

scheduleRequest (Merge request !48)

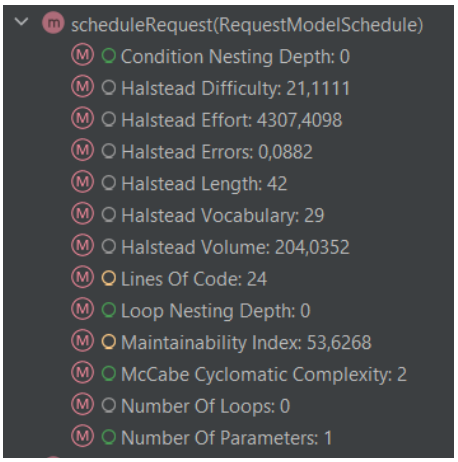
Schedule-microservice -> controllers -> ScheduleController.scheduleRequest()

When computing the code metrics, the scheduleRequest method turned out to have a McCabe Cyclomatic Complexity of eight, which is way above the threshold of four, which we took from MetricsTree. MetricsTree, namely, says that a value below three is normal, but a value of three or four is still ‘casual’, which is interpreted as still being sufficient. Besides that, the method had 38 lines of code, which is ‘uncommon’ according to MetricsTree and thus something we would preferably also lower.



The reason this was the case was because there were three different ways a bad request could be made, scheduling a request on an invalid date, with invalid resources or with too many resources when there are not enough available. All these cases were checked inside the method and, for each, exceptions with different error messages were thrown. This caused the high McCabe Cyclomatic Complexity and large number of lines of code.
To fix this, three methods were extracted, each examining one of the conditions for the request to be valid and throwing an exception if the condition does not hold for the specific request. Using a try-catch block, the scheduleRequest method now simply calls the other methods that check for the validity of the request and if an exception is thrown in one of them, the scheduleRequest method will also throw an exception with the same message.

By applying this refactoring operation, the McCabe Cyclomatic Complexity of the method decreased to a value of two and the number of lines of code also decreased to 24, which is still not ‘common’, but at least ‘casual’ according to MetricsTree, which is already a good improvement to the 38 lines of code there were before.



WaitingListController (Merge request !49)

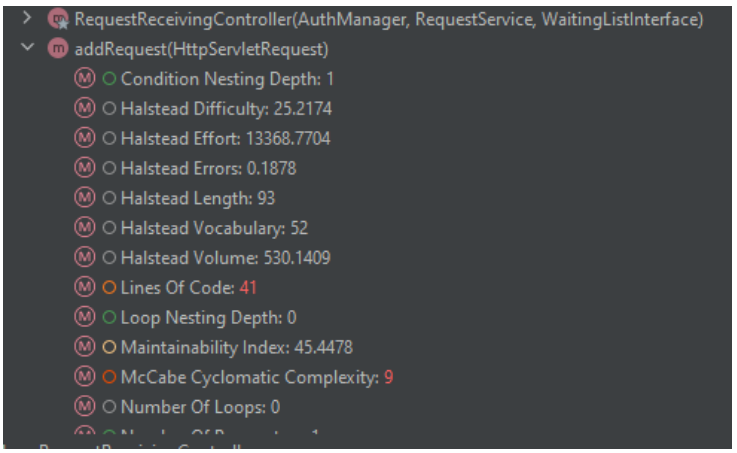
WaitingList-mircoservice -> controllers -> WaitingListController

In the controller of the waitingList the rejectRequest method had a McCabe Cyclomatic Complexity of 5 and the acceptRequest method had a McCabe Cyclomatic Complexity of 6 and was 34 lines of code. The too high McCabe Cyclomatic Complexity and the too high number of lines of code, were found using MetricsTree.
Because there was code duplication within rejectRequest and acceptRequest it was possible to fix both of the methods by extracting a new method adminAuthority.
After extracting the method, the rejectRequest method now has a McCabe Cyclomatic Complexity of 2 and the acceptRequest now has a McCabe Cyclomatic Complexity of 3. The number of lines of code of the acceptRequest method now is 30, so also under the threshold.

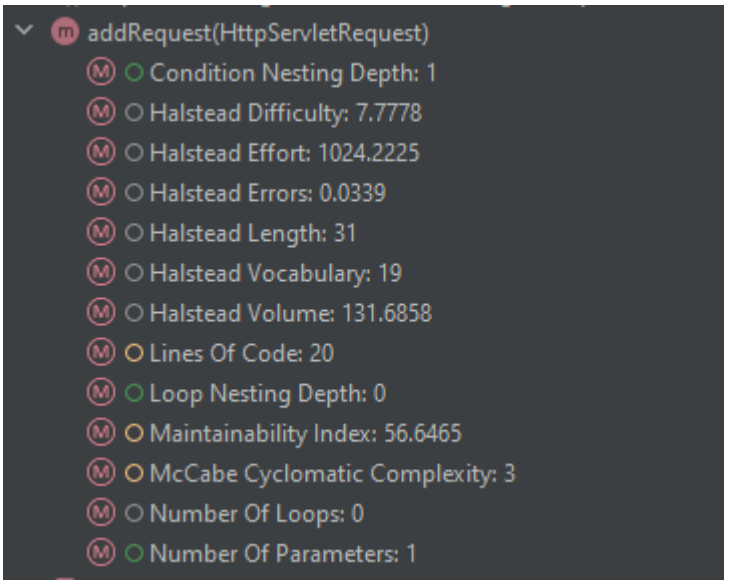
addRequest (Merge request !44)

User-microservice -> controllers -> RequestReceivingController.addRequest()

addRequest method was a big method to receive data model for request from a user, create an actual model and send it to the WaitingList microservice. It had too much logic inside one method (41 lines and cyclomatic complexity of 9).



We have changed this method to 4 methods – 1 old method and 3 new methods, that are located in a new requests/AddNewRequestsService class. Now this method is divided to:
public addRequest with 20 lines of code and a complexity of 3
AddNewRequestsService:
public createRequest with 12 lines of code and a complexity of 2
private checkAuthorization with 7 lines of code and a complexity of 4
private contactWaitingList with 14 lines and a complexity of 3



Request (Merge request !45)

WaitingList-microservice -> domain -> Request

The Request constructor in the waiting list microservice had a high Cyclomatic Complexity (8). It was also quite long (29 lines, without counting Javadoc). We wanted to reduce these metrics so that this constructor would be more readable and maintainable.

Method metrics before refactoring:

Method: Request(String, String, String, Resources, LocalDate, LocalDateTime)

Metric	Metrics Set	Description	Value	Regular Range
CND		Condition Nesting Depth	2	[0..2)
LND		Loop Nesting Depth	0	[0..2)
CC		McCabe Cyclomatic Complexity	8	[0..3)
NOL		Number Of Loops	0	
LOC		Lines Of Code	38	[0..11)
NOPM		Number Of Parameters	6	[0..3)
HVL	Halstead ...	Halstead Volume	364.6928	
HD	Halstead ...	Halstead Difficulty	11.1304	
HL	Halstead ...	Halstead Length	69	
HEF	Halstead ...	Halstead Effort	4059.1889	
HVC	Halstead ...	Halstead Vocabulary	39	
HER	Halstead ...	Halstead Errors	0.0848	
MMI	Maintaina...	Maintainability Index	47.3261	[0.0..19.0]

To achieve this, we extracted the parts of the method which had to do with checking the attributes. We created a new class, RequestValidator, which has all the methods for validating the attributes.

In the end, we managed to reduce the Cyclomatic Complexity to 1, and the Lines Of Code to 9 (without counting Javadoc). By refactoring, we also reduced the Condition Nesting Depth from 2 to 0.

Method metrics after refactoring:

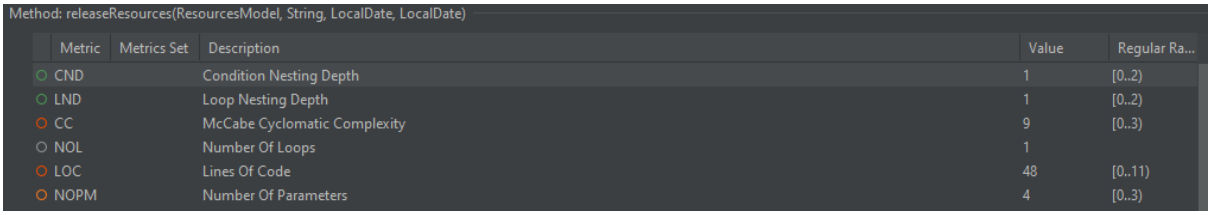
Method: Request(String, String, String, Resources, LocalDate, LocalDateTime)

Metric	Metrics Set	Description	Value	Regular Range
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	20	[0..11)
NOPM		Number Of Parameters	6	[0..3)
HVL	Halstead ...	Halstead Volume	101.9503	
HD	Halstead ...	Halstead Difficulty	3.0	
HL	Halstead ...	Halstead Length	24	
HEF	Halstead ...	Halstead Effort	305.8508	
HVC	Halstead ...	Halstead Vocabulary	19	
HER	Halstead ...	Halstead Errors	0.0151	
MMI	Maintaina...	Maintainability Index	57.5851	[0.0..19.0]

releaseResources(Merge request !53)

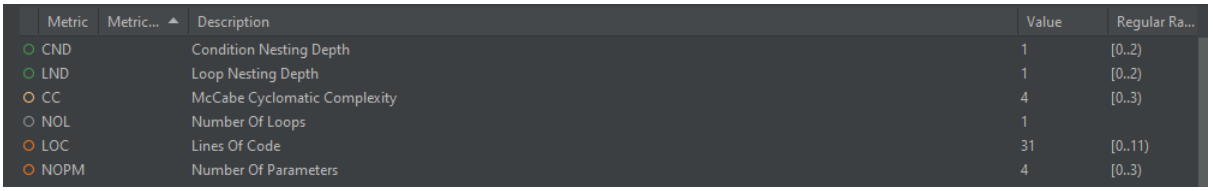
Resources-microservice->domain->resources->ResourceReleaseService

The releaseResources method in resourceRepositoryService had a Cyclomatic Complexity of 9 which was deemed to be too high. In order to reduce the CC of this method the logic for checking if resources could be released was moved to another method in ResourceLogicUtils.


Method: releaseResources(ResourcesModel, String, LocalDate, LocalDate)

Metric	Metrics Set	Description	Value	Regular Ra...
CND		Condition Nesting Depth	1	[0..2)
LND		Loop Nesting Depth	1	[0..2)
CC		McCabe Cyclomatic Complexity	9	[0..3)
NOL		Number Of Loops	1	
LOC		Lines Of Code	48	[0..11)
NOPM		Number Of Parameters	4	[0..3)

CC before refactor



Metric	Metric...	Description	Value	Regular Ra...
CND		Condition Nesting Depth	1	[0..2)
LND		Loop Nesting Depth	1	[0..2)
CC		McCabe Cyclomatic Complexity	4	[0..3)
NOL		Number Of Loops	1	
LOC		Lines Of Code	31	[0..11)
NOPM		Number Of Parameters	4	[0..3)

CC after refactor

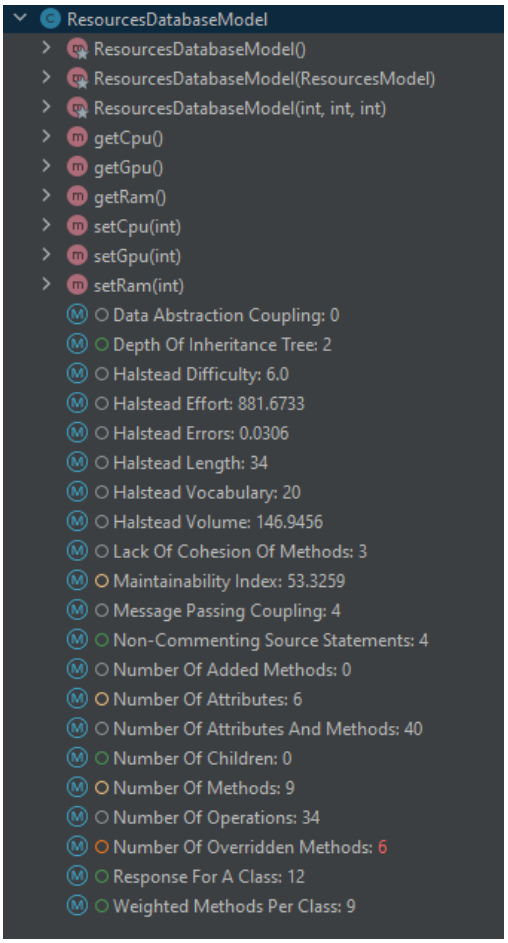
Class level

ResourcesDatabaseModel (Merge request !44)

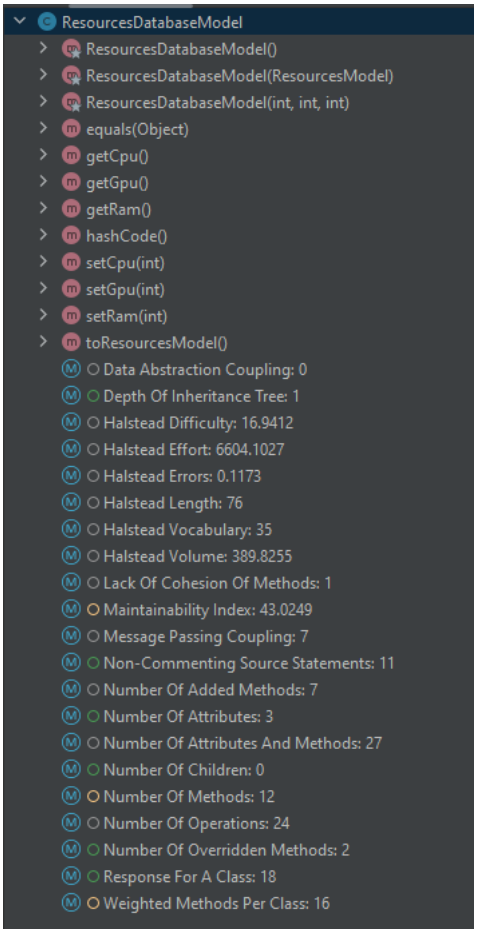
Resources-microservice -> domain -> ResourcesDatabaseModel

Previously, this method had a redundant inheritance from ResourcesModel class from the Commons module. A lot of the superclass methods were overridden (6 overridden methods

and 3 unused superclass attributes) and it gave code smell because of it.



It was decided to remove the inheritance from this class and add a method toResourcesModel, which translates the object of this class into the object of the class ResourcesModel. It was also needed to fix some related classes and tests to have working Resources microservice.



PostNodeRequestModel (Merge request !50)

commons -> models -> resources -> PostNodeRequestModel
commons -> models -> Node
In commons there were two models (PostNodeRequestModel and Node) which had the same attributes. Both models were used for communication between different parts of the system. Because having two roughly the same models is not ideal, we switched to using only one of the models. All the PostNodeRequestModel usages were replaced by the Node model and after that the PostNodeRequestModel Class was removed from the system.

SingleTableWaitingList (Merge request !51)

waiting-list-microservice -> domain -> SingleTableWaitingList

The SingleTableWaitingList class in the waiting list microservice had a bit too many methods (Weighted Methods Per Class 19), which made the class more difficult to read and also harder to maintain.

Class metrics before refactoring:

Class: SingleTableWaitingList

Metric	Metrics Set	Value	Description	Regular Range
CHVL	Halstead Metric Set	1379.3312	Halstead Volume	
CHD	Halstead Metric Set	71.1765	Halstead Difficulty	
CHL	Halstead Metric Set	213	Halstead Length	
CHEF	Halstead Metric Set	98175.9281	Halstead Effort	
CHVC	Halstead Metric Set	89	Halstead Vocabulary	
CHER	Halstead Metric Set	0.7094	Halstead Errors	
WMC	Chidamber-Kemerer Metrics Set	19	Weighted Methods Per Class	[0..12)
DIT	Chidamber-Kemerer Metrics Set	1	Depth Of Inheritance Tree	[0..3)
RFC	Chidamber-Kemerer Metrics Set	45	Response For A Class	[0..45)
LCOM	Chidamber-Kemerer Metrics Set	1	Lack Of Cohesion Of Methods	
NOC	Chidamber-Kemerer Metrics Set	0	Number Of Children	[0..2)
NOA	Lorenz-Kidd Metrics Set	4	Number Of Attributes	[0..4)
NOO	Lorenz-Kidd Metrics Set	25	Number Of Operations	
NOOM	Lorenz-Kidd Metrics Set	0	Number Of Overridden Methods	[0..3)
NOAM	Lorenz-Kidd Metrics Set	7	Number Of Added Methods	
SIZE2	Li-Henry Metrics Set	29	Number Of Attributes And Methods	
NOM	Li-Henry Metrics Set	8	Number Of Methods	[0..7)
MPC	Li-Henry Metrics Set	53	Message Passing Coupling	
DAC	Li-Henry Metrics Set	4	Data Abstraction Coupling	
NCSS	Chr. Clemens Lee Metrics Set	50	Non-Commenting Source Statements	[0..1000)
CMI	Maintainability Index	33.4407	Maintainability Index	[0.0..19.0]

To reduce the number of methods, we extracted two methods from the class that logically belonged together (both of these methods were executed periodically by the system) into a new class (WaitingListAutomaticTasks).

We reduced the Weighted Methods Per Class to 11. Since three of the attributes were used only by these two methods, we also reduced the Number Of Attributes from 4 to 1. The Response For A Class metric also went down from 45 to 15.

Class metrics after refactoring:

Class: SingleTableWaitingList

Metric	Metrics Set	Description	Value	Regular Range
CHVL	Halstead Metric Set	Halstead Volume	380.0	
CHD	Halstead Metric Set	Halstead Difficulty	29.7	
CHL	Halstead Metric Set	Halstead Length	76	
CHEF	Halstead Metric Set	Halstead Effort	11286.0	
CHVC	Halstead Metric Set	Halstead Vocabulary	32	
CHER	Halstead Metric Set	Halstead Errors	0.1677	
WMC	Chidamber-Kemerer Metrics Set	Weighted Methods Per Class	11	[0..12)
DIT	Chidamber-Kemerer Metrics Set	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber-Kemerer Metrics Set	Response For A Class	15	[0..45)
LCOM	Chidamber-Kemerer Metrics Set	Lack Of Cohesion Of Methods	1	
NOC	Chidamber-Kemerer Metrics Set	Number Of Children	0	[0..2)
NOA	Lorenz-Kidd Metrics Set	Number Of Attributes	1	[0..4)
NOO	Lorenz-Kidd Metrics Set	Number Of Operations	23	
NOOM	Lorenz-Kidd Metrics Set	Number Of Overridden Methods	0	[0..3)
NOAM	Lorenz-Kidd Metrics Set	Number Of Added Methods	5	
SIZE2	Li-Henry Metrics Set	Number Of Attributes And Methods	24	
NOM	Li-Henry Metrics Set	Number Of Methods	6	[0..7)
MPC	Li-Henry Metrics Set	Message Passing Coupling	9	
DAC	Li-Henry Metrics Set	Data Abstraction Coupling	1	
NCSS	Chr. Clemens Lee Metrics Set	Non-Commenting Source Statements	22	[0..1000)
CMI	Maintainability Index	Maintainability Index	43.3112	[0.0..19.0]

ResourceRepositoryService (Merge request !53)

resources-microservice->src->main->java->nl.tudelft.sem.resources->domain->resources->ResourceRepositoryService

In the Resources Microservice, the ResourceRepositoryService was responsible for handling the resource repositories and any interactions with them. This solution however proved to be bloated and hard to read/maintain, as proven by the very high WMC metric value.

Class: ResourceRepositoryService

Metric	Metric...	Description	Value	Regular Ra...
WMC	Chidamber...	Weighted Methods Per Class	29	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	43	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	0	[0..2)

The solution to this problem was to split this class up into three different classes. For ease of implementation, the original class still exists as an interface to the other classes and still has its original method signatures intact. The new classes are:
ResourceAllocationService which handles adding new resources to the cluster and using resources from the cluster.
ResourceReleaseService which handles releasing resources.
And ResourceLogicUtils which handles manipulating ResourcesModel objects and other miscellaneous logic related to ResourcesModels.

The metrics post refactor are:

Metric	Metric...	Description	Value	Regular Ra...
WMC	Chidamber...	Weighted Methods Per Class	7	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	13	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	2	
NOC	Chidamber...	Number Of Children	0	[0..2)

for ResourceRepositoryService

Class: ResourceLogicService

Metric	Metric...	Description	Value	Regular Ra...
WMC	Chidamber...	Weighted Methods Per Class	27	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	17	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	3	
NOC	Chidamber...	Number Of Children	0	[0..2)

for ResourcesLogicUtils. Here an explanation is in order for the high WMC metric: This class holds a lot of checks for data integrity that need to be done somewhere, previously in the monolithical ResourceRepositoryService. It is decided to move all this checking logic into its own class in order to isolate it from the rest of the code, thus the high WMC metric.

Metric	Metric...	Description	Value	Regular Ra...
WMC	Chidamber...	Weighted Methods Per Class	6	[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)

for ResourceAllocationService

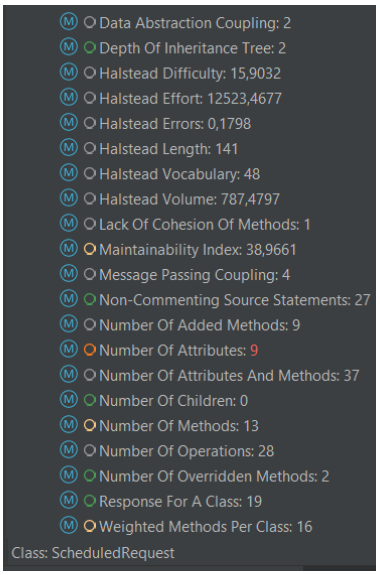
Metric	Metric...	Description	Value	Regular Ra...
WMC	Chidamber...	Weighted Methods Per Class	11	[0..12)
DIT	Chidamber...	Depth Of Inheritance Tree	1	[0..3)
RFC	Chidamber...	Response For A Class	33	[0..45)
LCOM	Chidamber...	Lack Of Cohesion Of Methods	1	
NOC	Chidamber...	Number Of Children	0	[0..2)

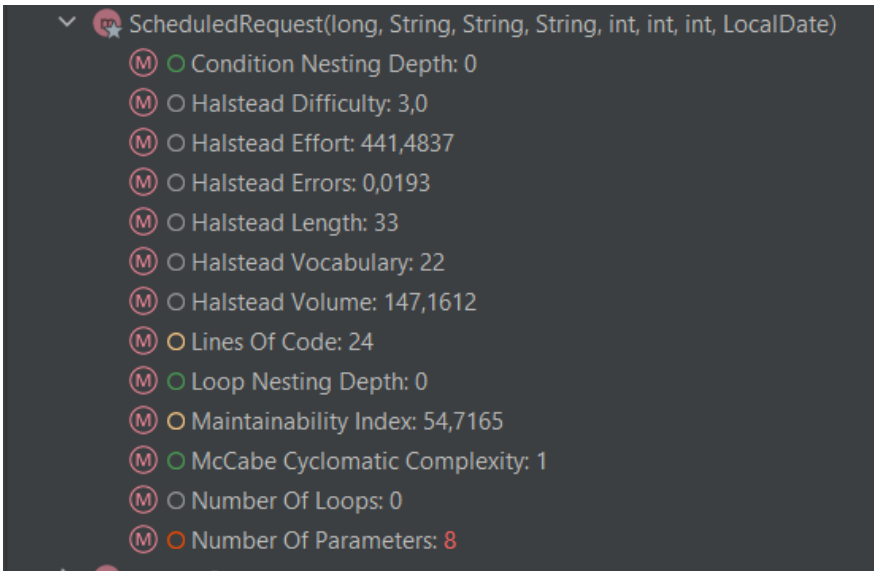
for ResourceReleaseService

ScheduledRequest (Merge request !47)

Schedule-microservice -> domain -> request -> ScheduledRequest

Before refactoring, the ScheduledRequest class had nine attributes, which is ‘uncommon’ according to MetricsTree and so was thought to be an amount too large. This in turn also caused the constructor of the class to have too many parameters, eight, since almost all the attributes had to be instantiated when a ScheduledRequest object was created.





The ScheduledRequest class had so many parameters, since there are a lot of things that need to be stored when a request is scheduled, an ID, name, description, faculty, three different resources and a planned date. Besides this, the class also extended the HasEvents class, that documented when a ScheduledRequest object was created, and this also had one attribute, adding up to nine total attributes.
This now has been solved by extracting classes. As said before, the ScheduledRequest class contains three attributes to indicate the resource usage of a request. To reduce this to one attribute, a Resources class has been extracted containing only these three attributes. Furthermore, this new resources attribute is combined with the name, description and faculty of the request, the basic information for a request, and extracted again in another Request class. Lastly, the HasEvents class has been deleted, since it has not been updated much during the project and thus is not really useful in the application.
This deletion together with the extraction of two classes caused the ScheduledRequest method to only have three attributes, the ID, the request and the planned date, which is a ‘common’ value according to MetricsTree and thus a big improvement in the quality of the code. As a result of the decrease in the amount of attributes, the amount of parameters of the constructor are now also three, which is an acceptable amount.