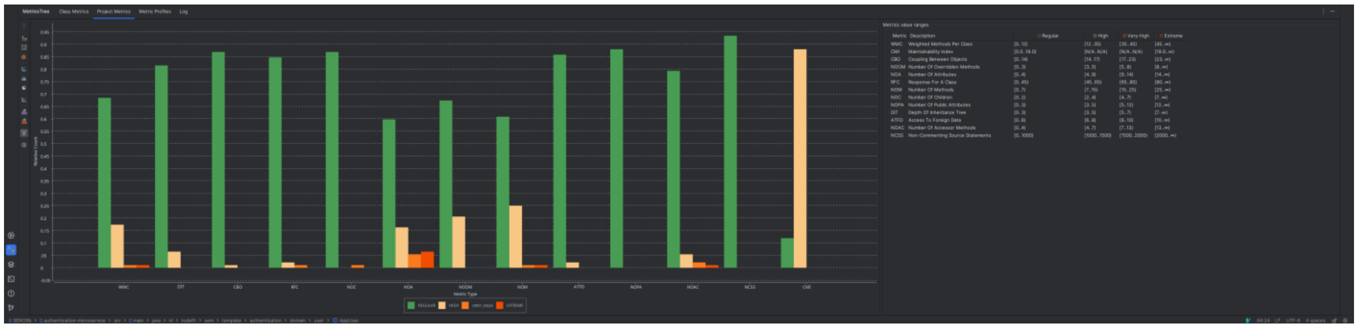
Assignment 2

Group 26b: Alan Styslavski, Attila Birke, Nicolas Zink, Javier Perez, Berken Tekin, Jort Boon

Metrics Before Refactoring

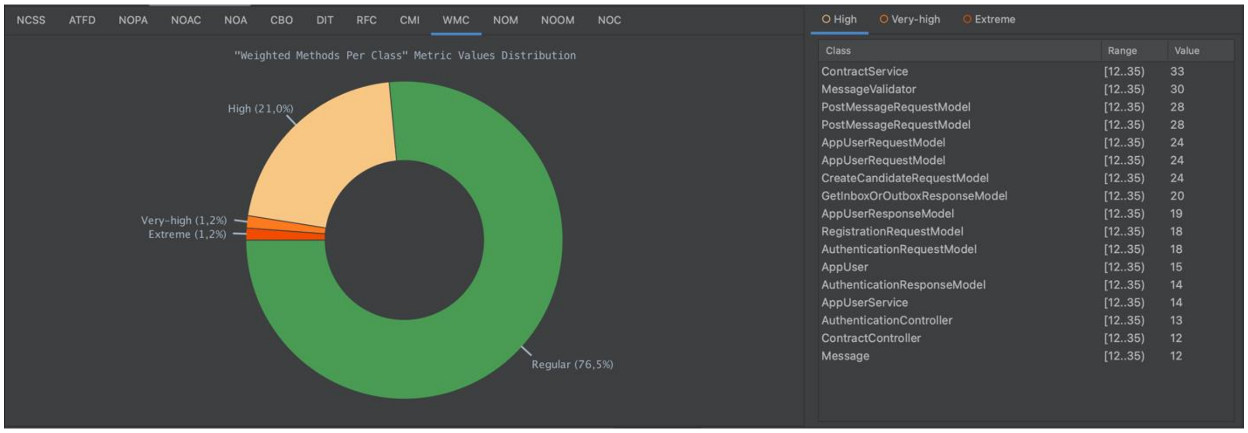


Metrics value ranges

Metric	Description	Regular	High	Very-high	Extreme
WMC	Weighted Methods Per Class	[0..12)	[12..35)	[35..45)	[45..∞)
CMI	Maintainability Index	[0.0..19.0)	[N/A..N/A)	[N/A..N/A)	[19.0..∞)
CBO	Coupling Between Objects	[0..14)	[14..17)	[17..23)	[23..∞)
NOOM	Number Of Overridden Methods	[0..3)	[3..5)	[5..8)	[8..∞)
NOA	Number Of Attributes	[0..4)	[4..9)	[9..14)	[14..∞)
RFC	Response For A Class	[0..45)	[45..65)	[65..80)	[80..∞)
NOM	Number Of Methods	[0..7)	[7..15)	[15..25)	[25..∞)
NOC	Number Of Children	[0..2)	[2..4)	[4..7)	[7..∞)
NOPA	Number Of Public Attributes	[0..3)	[3..5)	[5..12)	[12..∞)
DIT	Depth Of Inheritance Tree	[0..3)	[3..5)	[5..7)	[7..∞)
ATFD	Access To Foreign Data	[0..6)	[6..8)	[8..10)	[10..∞)
NOAC	Number Of Accessor Methods	[0..4)	[4..7)	[7..13)	[13..∞)
NCSS	Non-Commenting Source Statements	[0..1000)	[1000..1500)	[1500..2000)	[2000..∞)

Image 1.1. Metrics created by MetricsTree before the refactoring.

For computing the project metrics we used the MetricsTree plugin for IntelliJ. From the metrics provided, we prioritized the Chidamber and Kemerer metrics; however, we also considered LOC, NOM and NOA as a metric for code readability. As can be seen in the image 1.1 some of the metrics have high, very high and extreme values. These values are grouped by thresholds, which are globally acceptable values. First, we considered the higher valued metrics to be solved, however even though some of the metrics are high it does not necessarily mean that it needs refactoring.

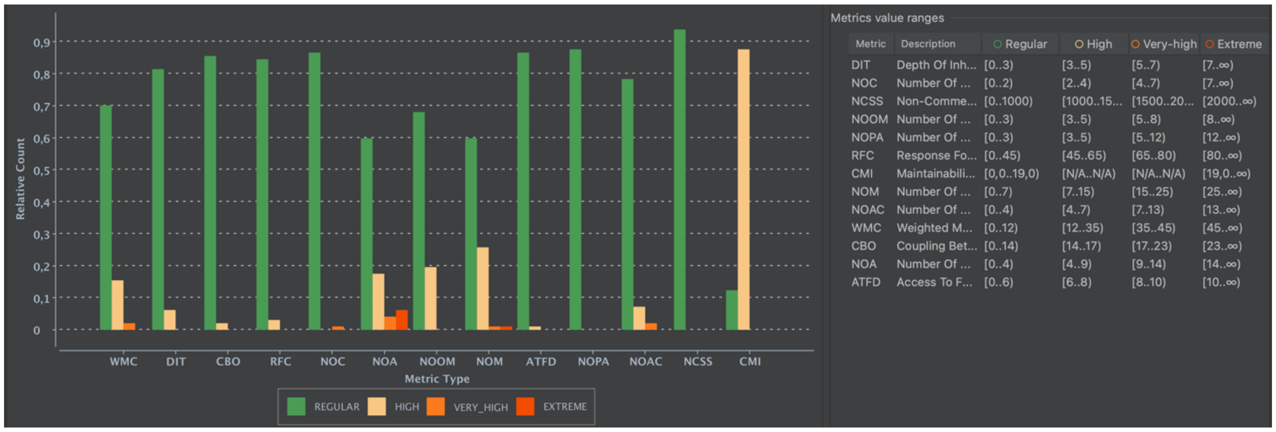


Class	Range	Value
ContractService	[12..35)	33
MessageValidator	[12..35)	30
PostMessageRequestModel	[12..35)	28
PostMessageRequestModel	[12..35)	28
AppUserRequestModel	[12..35)	24
AppUserRequestModel	[12..35)	24
CreateCandidateRequestModel	[12..35)	24
GetInboxOrOutboxResponseModel	[12..35)	20
AppUserResponseModel	[12..35)	19
RegistrationRequestModel	[12..35)	18
AuthenticationRequestModel	[12..35)	18
AppUser	[12..35)	15
AuthenticationResponseModel	[12..35)	14
AppUserService	[12..35)	14
AuthenticationController	[12..35)	13
ContractController	[12..35)	12
Message	[12..35)	12

Image 1.2 Pie chart of metrics, where the classes that cause high values can be seen on the right.

We can also see, in image 1.2, which classes cause the metric values to be high. In the image the metric on the right the NOM values can be checked for the classes. After identifying which classes produce high values, we can check out the classes individually and start refactoring them if it's necessary.

Metrics After Refactoring



Metrics value ranges

Metric	Description	Regular	High	Very-high	Extreme
DIT	Depth Of Inh...	[0..3)	[3..5)	[5..7)	[7..∞)
NOC	Number Of ...	[0..2)	[2..4)	[4..7)	[7..∞)
NCSS	Non-Comme...	[0..1000)	[1000..15...	[1500..20...	[2000..∞)
NOOM	Number Of ...	[0..3)	[3..5)	[5..8)	[8..∞)
NOPA	Number Of ...	[0..3)	[3..5)	[5..12)	[12..∞)
RFC	Response Fo...	[0..45)	[45..65)	[65..80)	[80..∞)
CMI	Maintainabili...	[0,0..19,0)	[N/A..N/A)	[N/A..N/A)	[19,0..∞)
NOM	Number Of ...	[0..7)	[7..15)	[15..25)	[25..∞)
NOAC	Number Of ...	[0..4)	[4..7)	[7..13)	[13..∞)
WMC	Weighted M...	[0..12)	[12..35)	[35..45)	[45..∞)
CBO	Coupling Bet...	[0..14)	[14..17)	[17..23)	[23..∞)
NOA	Number Of ...	[0..4)	[4..9)	[9..14)	[14..∞)
ATFD	Access To F...	[0..6)	[6..8)	[8..10)	[10..∞)

Image 2.1 Metrics created by MetricsTree after the refactoring.

As can be seen in Image 2.1 the metrics for many classes were improved. The extreme values at most cases were fixed. We decided that some of the values, even though they are high, should be kept as it does not make sense to refactor them.

Class Refactoring

Message class refactoring

The code smell detected in the Message class was Data Clump. Extract class refactoring method was utilised. 3 attributes of the Message class (sentAt, readAt, wasRead) were moved to a new Embeddable class called MessageStatus. This resulted inMessage class' NOA (number of attributes) being lowered from 8 to 6, NOM (Number of methods) to 9 from 11 and WMPC (Weighted Methods per Class) being lowered to 9 from 12. The refactoring resulted in smaller, easier to understand Classes.

Contract class refactoring

The code smell detected for Contract class was Data Clump and Long Class. By fixing these code smells the metrics for NOA (Number of Attributes), WMC (Weighted Methods Per Class), NOM (Number of Methods) and LCOM (Lack of Cohesion of Methods) could be reduced. The refactoring methods I used are Extract Class and Move methods. I made two new classes and moved the fields and methods responsible for the relevant functionalities in them. This improves overall readability and maintainability of the class.

Authentication Controller refactoring

The AuthenticationController used to have three different methods to handle registering new users: register, registerCandidate and registerAdmin. All of these methods did the same thing, except send the RegistrationService a different employee role info. With the refactor, there is now only one register method, and the users will supply the role of the new employee when they send a request.

Contract Controller refactoring

The code smell for the contract controller was that it was a bloater and a large class. We decided that dividing the controller into two separate controllers would be beneficial to decrease the coupling of objects and reduce the size of the class. We split the endpoints based on which service they communicate with, improving readability, and allowing for future additions of features to be more organised.

Method Refactoring

Mailboxservice: isMessageValid refactoring

The code smells here were dead code and unnecessary complexity. To address these, I first removed the indispensable method, then extracted the switch statement into a new method, as well as replaced the fall-through switch with a map-switch. By doing so I decreased the CC and LOC of the method, making it simpler and easier to read and maintain.

Contract constructor method refactoring

The code smell for Contract constructor method was Long Parameter List and Long Class. By improving these code smells we reduced the NOPM (Number of Parameters). The way we did it was by the Introduction of Parameter Objects, which is related to the class refactoring. As we created new classes for some attributes, the input parameters for the constructor were less, as two objects are used instead of seven parameters. With this the LOC (Lines of Code) were also decreased. This improves the overall readability of the method, instead of many parameters we change it to one object.

MessageValidator: getInbox/getOutbox refactoring

The code smell found in the methods was Duplicate Code (2 instances of 8 duplicated lines). The duplicate code was responsible for converting data retrieved from database and putting it into a response model. The code was moved into a constructor for GetInboxOrOutboxResponseModel constructor. This will result in a more readable code, that will require less maintenance if changed.

JwtRequestFilter: filterInternal Refactoring

The code smell detected for JwtRequestFilter class was a long method with high cyclomatic complexity in doFilterInternal(). By extracting a part of this method into a separate function setAuthentication(), the LOC (lines of code) was split into two different methods, the CC (cyclomatic complexity) was reduced, as well as the CND (condition nesting depth). This way, the method is more easily maintainable and clearer to understand.

ContractService: propose, accept, terminate refactoring

The code smell detected here was duplicate code and long methods. We extracted the methods isReviewing and sendMessage to avoid duplication in our code and decrease the CC of these methods. While doing this refactoring we also implemented other business logic in these methods, so the metrics do not all show an improvement. (This refactoring was done during the first part of the project)

Class	Code smell	Refactoring	Metric before	Metric after
Message	Data clump	Extract class: move sentAt, readAt, wasRead into a new Embeddable class.	NOA: 8 NOM: 11 WMPC: 12	NOA: 6 NOM: 9 WMPC: 9
Contract	Bloater: Data clump Long class	Extract class into SalaryInfo and ContractAdditions. Some methods were moved to SalaryInfo to increase cohesion and reduce coupling. This reduces data clump in Contract class. Also improves other metrics such as WMC, NOM, LCOM.	NOA: 14 WMC: 34 NOM: 17 LCOM: 5	NOA: 10 WMC: 25 NOM: 13 LCOM: 1
AuthenticationController	Duplicate code	There was a different “register” method for each type of employee, we unified all of them into one.	NOM: 5 WMC: 13	NOM: 3 WMC: 8

ContractController	Bloater: Long class	Extract class, create 2 controllers with related endpoints	CBO: 15	CBO: 14
BaseHandler	Parallel inheritance hierarchies	Does not make sense to refactor as a high NOC does not necessarily indicate a fault in the code, here the inheritance is beneficial because we avoid a lot of code duplication		NOC: 6

Method	Code smell	Refactoring	Metric before	Metric after
MessageValidator :isMessageValid	Dead code, Unnecessary complexity	Remove stub code. Extract switch statement into a new method. Replace fall-through switch with a map-switch combination.	CC:14 LOC:41	CC 2 + 7 LOC: 14 + 12 (Split over 2 methods)
Contract: Contract	Long parameter list Long method	Introduction of Parameter Objects which reduce the number of parameters and by this the lines of code in the method.	NOPM: 11 LOC:31	NOPM: 6 LOC: 26
Mailboxservice:getInbox/getOutbox	Duplicate code (2x8 lines)	Move duplicate code responsible for parsing data into a response model to GetInboxOrOutboxResponseModel constructor	LOC:2x14	LOC:2x6 + 8
JwtRequestFilter: doFilterInternal	Long method	Extract code inside method to external method, which reduces overall complexity of initial method, and makes it easier and less confusing to maintain	CND: 3 CC: 7 LOC: 45	CND: 2 CC: 4 LOC: 23
ContractService:propose, accept, terminate	Duplicate code, Long method	Extract method, sendMessage and isReviewing methods created which avoids code duplication, here the metrics don’t all show improvements because more business logic was implemented	LOC: 60, 38, 29 CC: 6, 4, 4	LOC: 59, 46, 41 CC: 6, 5, 4