Assignment 2 Group 23a

The aim of this report is to inform about the refactoring metrics used to assess the code quality of the project, explain how different techniques were applied to improve it and evaluate the produced results after the improvement.

For this assignment, MetricsTree was chosen to aid us in the process of identifying the problematic classes. There were several metrics considered. The first had to do with coupling between objects, RFC (Response For A Class), cyclomatic complexity, and lines of code. Metrics such as maintainability index, NOM (Number of Methods), NOC (Number of Children), and NOA (Number of Attributes) have not been taken into account. The reason was that they were not very helpful when it came down to finding troublesome classes. The maintainability index did not produce sensible results in the current situation since it was very high for some very simple things and it was not reasonable to use it as a guideline. NOM was not reliable as well because of the use of Lombok annotations for getters and setters, which skewed the results. When it comes to NOA, it was not relevant in many cases due to the fact that it pointed to static constants in the classes, which make the code more readable and maintainable. As for NOC, it is not very helpful since it flags everything with a score of more than 1. If this statistic is used, then it would render the whole class hierarchy useless which is undesired. The above-mentioned metrics were deeply looked into and taken into consideration whenever sensible. However, there were many cases when they were ignored, due to the arguments presented.

One of the first classes identified as problematic was the `HoaService`. Some of the immediately obvious smells were a god class, bloater, and complex code. In the beginning, it had high coupling, as the coupling between objects was 23 (10 above suggested), access to foreign data was 9 (4 above suggested) and RFC was 84 which was 40 points above the suggested amount. Together with that, the number of methods was on the high end. For the mentioned reasons, a decision to split it into new separate services was made. The original HoaService was kept, which deals with the creation, joining, and leaving. We extracted the methods that had to tackle the users' membership into the `HoaMembershipService`. Those are the cases when the person has been a board member, or if they have been in the HOA for enough years to become a board member, etc. Finally, the methods dealing with reporting users have been extracted and put in a separate service, namely `HoaReportService`. After splitting the methods into their own classes and running MetricsTree again, all of the results have significantly improved. The two new classes were all in the green, which meant they had no issues according to the scores under consideration. And the issues in the main `HoaService` class have decreased significantly. Coupling between objects(CBO) dropped to 14 (decreased by 9) Access to foreign data(AFD) decreased to 6 (down by 3) and RFC to 51 (down by 33!). The results from the refactoring procedure were satisfying enough.

After that, a closer look at one of the methods inside the `HoaService` -

`joinHoa()` produced concerning results on the MetricsTree. It had high LOC (34) and high CC (5). Firstly, to reduce the CC, a part of the method was extracted outside into a method `canJoin()`, which checks if a user can technically join as long as there is a Hoa and the user is in its region. A check in the method whether the user is not already a part of the HOA was also conducted. However, this functionality has already been implemented in the new `HoaMembershipService`. Because of the before-mentioned fact, some more logic has been extracted which led to a reduction in the complexity of the method. It all produced satisfying results as now the method had a maintainable, easy-to-read, and understandable body. The CC went down to 3 and LOC came within the limits of MetricsTree and is now 29, where a part of the lines come from the use of the builder pattern and are with the sole purpose of readability.

During the refactoring process, the class `RestConfig` was removed because it was exactly the same as the `RestTemplateConfig`. This seemed to be an issue due to separate branches and merges, but it is now removed and fixed.

After this, a closer look at the `HoaController` class showed that Non-Commented Source Code lines and Response for Class metrics are still a bit high, which lead to further refactorings in the class and its methods. It was found that the class is unnecessarily big, containing endpoints that would be more appropriate to be put in a separate controller. It is the same case as with the `HoaService` - all the reporting functionality seemed out of place, and for that reason, it has been moved to a new class - `ReportController`. With that change, the `HoaController` is no longer a god class, and it has a much clearer responsibility of managing basic endpoints.

This second look into the class also led to finding similarities/code duplication in two of its methods, namely `getNoticeBoard()` and `getHoaHistory()`. Both of these methods contained generic code for token retrieval and made an http request to another microservice, with the only difference being the url. The solution was to extract the generic token retrieval in `HttpServletRequestExtensions` class - `getBearerToken()` and the similar http request to be put in a new service, that will handle the integration - `getActivities()` (`ActivityService`). The problem with the urls being different due to slight patch change/parameter was resolved by creating an enum `Time` in the `ActivityService` class symbolising the parameter. By using an enum, no further code duplication and checks were needed to implement the inter-microservice communication in both methods. Also extracting that communication code made sense, as the controller methods had too many responsibilities. After these changes, it was noticed that the method `allowedToGetNoticeboard(...)`, used by both controller methods, was taking an entire HOA object as its parameter instead of a single hoa id. The method and its parameter list were refactored, improving both the CND and LND metrics, as well as the maintainability. As a result of all these changes, the `HoaController` has less code duplication and follows the single responsibility principle. The NCSS has been decreased from 42 to 26 and the RFC from 35 to 24. Also metrics such as WMC and NOM have been slightly improved in the process.

Another look into the `HoaController` class and the new `ReportController` class led to the realisation that there were two methods with high Cyclomatic Complexity (4 and 5) and high LOC (27 and 30). The reason for these statistics was the switch statements in the methods `report()` and `joinHoa()`. This was because each method made a check for an enum such that it knows what kind of `HttpStatus` to return. The solution to this problem was to put a label on each enumerator with the `HttpStatus` it corresponds to. This reduced the CC of the methods to 1 and 2 respectively and the LOC to 14 and 17 respectively for each class. After the changes, the class methods became a lot more readable. The changes made gave positive results to the overall maintainability of the `HoaController`.

Looking at the `ElectionController` (in the hoa module) and `ActivityController` (inside the `activities-and-elections` module) classes showed that both these classes had a high Weighted Methods Per Class score, this indicated that there were too many methods in the class with too high a cyclomatic complexity. In both of these classes, this was reduced substantially by changing the code style to something more Spring-like, namely, using exceptions combined with `@ResponseStatus` annotations for returning error codes to the client which used to be returned using a lot of separate branches (leading to a higher cyclomatic complexity) with their own return statements inside of them. After the changes, the WMPC metric went from 13 and 12 to 10 and 11 respectively. This resulted in the MetricsTree metric going from yellow to green for both classes.

Then a look into another microservice, namely activity-and-voting, showed that the method `handleEndOfProposal()` in `ProposalProcedureService` was with a Cyclomatic Complexity of 6 and Lines of Code of 39. This method seemed quite problematic. A logical decomposition of it showed that a significant part of it can be extracted into two separate methods - `checkValidityOfActivity()` and `handleAcceptedProposal()`. This reduced the CC of the method to 3 and the two new methods have a CC of 3 and 3 and the LOC of the three methods is now 20 (here again most of the lines come from JavaDoc and the use of builder pattern), 10 and 10 respectively for each method. These results produced are satisfying enough since CC and LOC have been improved from red to yellow and green.

Table of changes for Classes:

Fixed\Metric	RFC	CBO	AFD	WMPC
HoaService	84->51	23->14	9->6	-------------
HoaController	35->24	------------	------------	20->11
ElectionsController	22->20	------------	------------	13->10
ActivitiesController	32->25	------------	------------	12->11

Rest Configure class was removed, and the HoaServiceTest was refactored to reduce RFC.

Table of changes for Methods:

Method\Metric	CC	LOC
joinHoa() - HoaService	5 -> 3	34 -> 29
joinHoa() - HoaController	5 -> 2	30 -> 17
report() - HoaController	4 -> 1	27 -> 14
getNoticeBoard()	----------	24 -> 15
getHoaHistory()	----------	24 -> 15
handleEndOfProposal()	6->3	39->19