METRICI SOFTWARE ȘI INGINERIA CALITĂȚII

RAPORT 2 – REFACTORIZAREA CODULUI SI RESTABILIREA METRICILOR

Student: Drăghici Andreea-Maria
Grupa: IS1.B
Anul de studiu: I
Specializarea: Inginerie Software

Student Management

Plugins and tools used into Intellij IDEA:

- **Plugin to calculate the metrics: MetricsReloaded and MetricsTree**
- **Tool tofind and fix coding issues: SonarLint**
- **Plugin to provides static byte code analysis to look for bugs: SpotBugs**

Compliance with the following SOLID principles:

- ***Separation of Concerns (SoC):*** Separated the classes and interfaces into disting packages to isolate different aspects of the application.
  Target:
    - ✓ Model classes are separated in one package, and parsers/mappers/adapters have their own separate packages.

- ***Dependency Inversion Principle (DIP):*** By creating interfaces for parsers, mappers, and adapters, have created the necessary abstractions to invert dependencies. Classes that use them now depend on interfaces, not concrete implementations.
  Target:
    - ✓ This makes the code more modular and easier.

Compliance with the following design patterns:

- ***Model-View-Controller (MVC) arhitectural pattern:***

  Target:
    - ✓ ***Model:*** This includes my data model and the business logic associated with them.
    - ✓ ***View:*** My fxml files and UI components it is responible for presenting the data to the user and receiving the inputs from user.

- ✓ ***Controller:*** MainViewController acts as a controller. It handles user inputs, updates the model and manipulates the view. It connects the model and the view, ensuring they stay separate.

- ***Factory method creational pattern:*** In the ApplicationFactory class,through the applicationRunner method, this class decides which type of instance of the IApplication interface to create based on the criteria provided.
  Target:
  - ✓ It is decided whether to create a GUIApplication instance or throw an exception.

I used the next metrics:

1. **LOC (lines of code)**
2. **CLOC (lines of comment)**
3. **NCLOC (lines of non-comment)**
4. **NOM (number of methods)**
5. **C (number of classes in each package)**
6. **NOI (number of interfaces)**
7. **NOC (number of direct subclasses of each class that occur in the project)**
8. **NOSC (number of static classes)**
9. **WMC (weighted method complexity)**
10. **BUGS (average bugs per class)**
11. **VIOLATIONS (problems per class / errors or warnings)**

The following values were obtained:

1. LOC (lines of code)

OLD:

- *1020 lines of code in project*
- *174 lines per class (max)*
- *60 lines per method (max)*

! TARGET ! <24 lines per method => seems ok

NOW:

- 3478 lines of code in project
- 366 lines per class (max)
- 23 lines per method (max)

! TARGET ! <24 lines per method => seems ok - Done

2. CLOC (lines of comment)

OLD:

- *19 lines of comment in project*
- *6 lines per class (max)*
- *2 lines per method (max*

! TARGET ! >1 lines per method => seems ok

NOW:

- *1381 lines of comment in project*
- *66 lines per class (max)*
- *6 lines per method (max)*

! TARGET ! >1 lines per method => seems ok - Done

3. NCLOC (lines of non-comment)

OLD:

- *930 lines of non-comment in project*
- *60 lines of non-comment per method (max)*

! TARGET ! not sure if is ok

NOW:

- *1675 lines of non-comment in project*
- *22 lines of non-comment per method (max)*

! TARGET ! not sure what is the normal score / range

4. NOM (number of methods)

OLD:

- *85 methods in project*
- *10 methods per class (max)*

! TARGET ! <20 methods per class => seems ok

NOW:

- *208 methods in project*
- *23 methods per class (max)*

! TARGET ! >=12 methods per class => seems ok – Done

5. C (number of classes and interfaces in each package)

OLD:

- *24 classes in project*
- *6 classes per package (max)*

! TARGET ! not sure if is ok

NOW:

- *53 classes and 9 interfaces in project*
- *14 classes per package (max) and 7 interface per package (max)*

! TARGET ! >= 8 classes / interfaces in each package => seems ok – Done

6. NOI (number of interfaces)

OLD:

- *0 in project*

! TARGET ! not sure if is ok

NOW:

- 9 interfaces in project

! TARGET ! >= 6 interfaces => seems ok – Done

7. NOC (number of direct subclasses of each class that occur in the project)

OLD:

- *0 in project*

! TARGET ! <10 subclasses of each class => seems ok

NOW:

- *0 in project*

! TARGET ! <10 subclasses of each class => seems ok – Done

8. NOSC (number of static classes)

OLD:

- *0 in project*

! TARGET ! not sure if is ok

NOW:

- *0 in project*

! TARGET ! not sure what is the normal score / range

9. WMC (weighted method complexity)

OLD:

- 234 in project
- *38 per class (max)*

! TARGET ! <100 per class => seems ok

NOW:

- 318 in project
- *40 per class (max)*

! TARGET ! <100 per class => seems ok – Done

10. BUGS (average bugs per class)

OLD:

- *2 bugs in project*
- *0.08 per class*

! TARGET ! will have to fix it

NOW:

- *0 bugs in project*
- *0 per class*

! TARGET ! Bugs was fixed

11. VIOLATIONS (problems per class / errors or warnings)

OLD:

- *Warnings = 25 issues with low impact in project*
- *Errors = 15 issues with medium impact in project*
- *Critical Errors = 29 issues with high impact*
- *Total Issues = 69 issues in 13 classes*

! TARGET ! will have to fix it

NOW:

- Warnings = 0
- Errors = 0
- Critical Errors = 0
- Total Issues = 0

! TARGET ! Violations was fixed – Done

References:

1. **MetricsReloaded:**
   - https://blog.jetbrains.com/idea/2014/09/touring-plugins-issue-1/
   - https://plugins.jetbrains.com/plugin/93-metricsreloaded

2. **SonarLint:**
   - https://plugins.jetbrains.com/plugin/7973-sonarlint

3. **MetricsTree:**
   - https://plugins.jetbrains.com/plugin/13959-metricstree
   - https://github.com/b333vv/metricstree

4. **SpotBugs:**
   - https://plugins.jetbrains.com/plugin/14014-spotbugs
   - https://spotbugs.readthedocs.io/en/stable/links.html
   - https://spotbugs.github.io/