Assignment 4

Asteroids

Created by group OP27-G59 for the course

CSE2115 Software Engineering Methods

of the Computer Science curriculum

at the Delft University of Technology

Group members:

Onur Gökmen
Joseph Catlett
Irtaza Hashmi
Helena Westermann
Ceren Uğurlu

TABLE OF CONTENTS

2

Refactoring

1. The tool we used for computing code metrics for our project was CodeMR. Our first code metrics include **complexity, lines of code, lack of cohesion, size and weighted method count**.

Complexity: Implies being difficult to understand and describes the interactions between a number of entities. Higher levels of complexity in software increase the risk of unintentionally interfering with interactions and so increases the chance of introducing defects when making changes.

Lines of Code:
Related Quality Attributes: Size
The number of all nonempty, non-commented lines of the body of the class. CLOC is a measure of the size and also indirectly related to the class complexity.

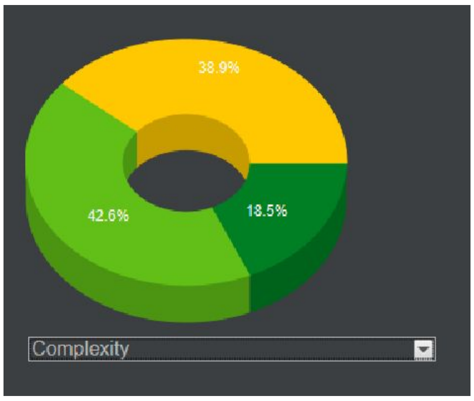
Lack of Cohesion: Measure how well the methods of a class are related to each other. High cohesion (low lack of cohesion) tend to be preferable because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability. In contrast, low cohesion is associated with undesirable traits such as being difficult to maintain, test, reuse, or even understand.

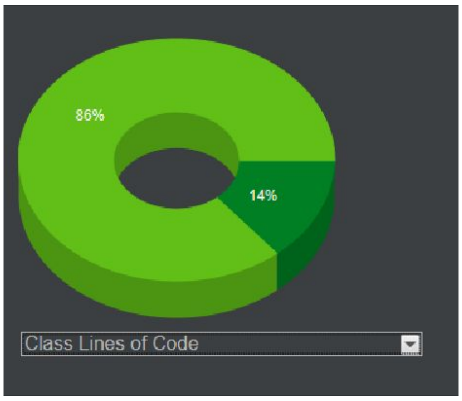
Size: Size is one of the oldest and most common forms of software measurement. Measured by the number of lines or methods in the code. A very high count might indicate that a class or method is trying to do too much work and should be split up. It might also indicate that the class might be hard to maintain.

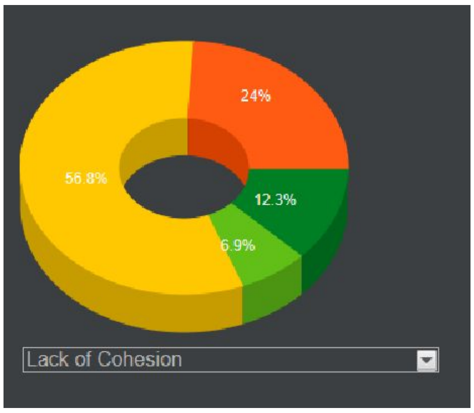
Weighted Method Count: Related Quality Attributes: Complexity, Size
The weighted sum of all class' methods represents the McCabe complexity of a class. It is equal to the number of methods if the complexity is taken as 1 for each method. The number of methods and complexity can be used to predict development, maintaining and testing effort estimation. In inheritance if the base class has a high number of method, it affects its' child classes and all methods are

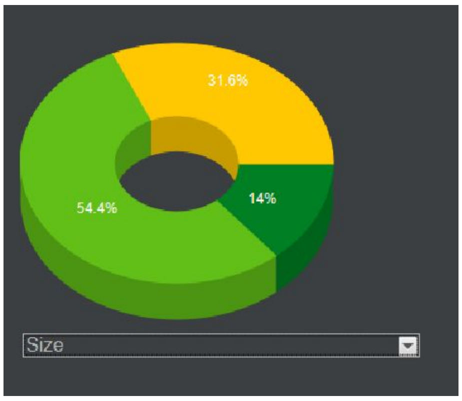
represented in sub-classes. If the number of methods is high, that class possibly domain-specific. Therefore they are less reusable. Also, these classes tend to more change and defect prone.

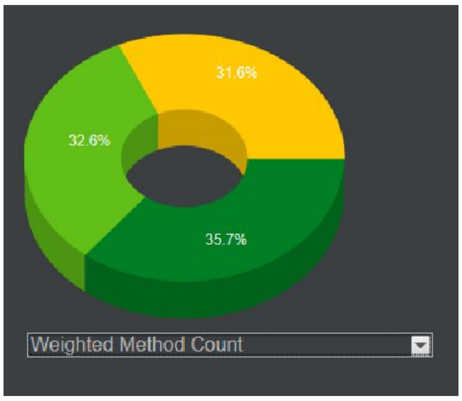
In our opinion, we should definitely improve cohesion as the pie chart depicts that some classes are not that cohesive (24% of the classes are red). In addition, we can improve the complexity, size and the weighted method count for the classes as the figures seem to be a little too high for some classes (yellow subsection of the piechart). Thankfully, those pie charts are not red, which would have been a serious concern.











General Information

Total lines of code: 1747

Number of classes: 26

Number of packages: 5

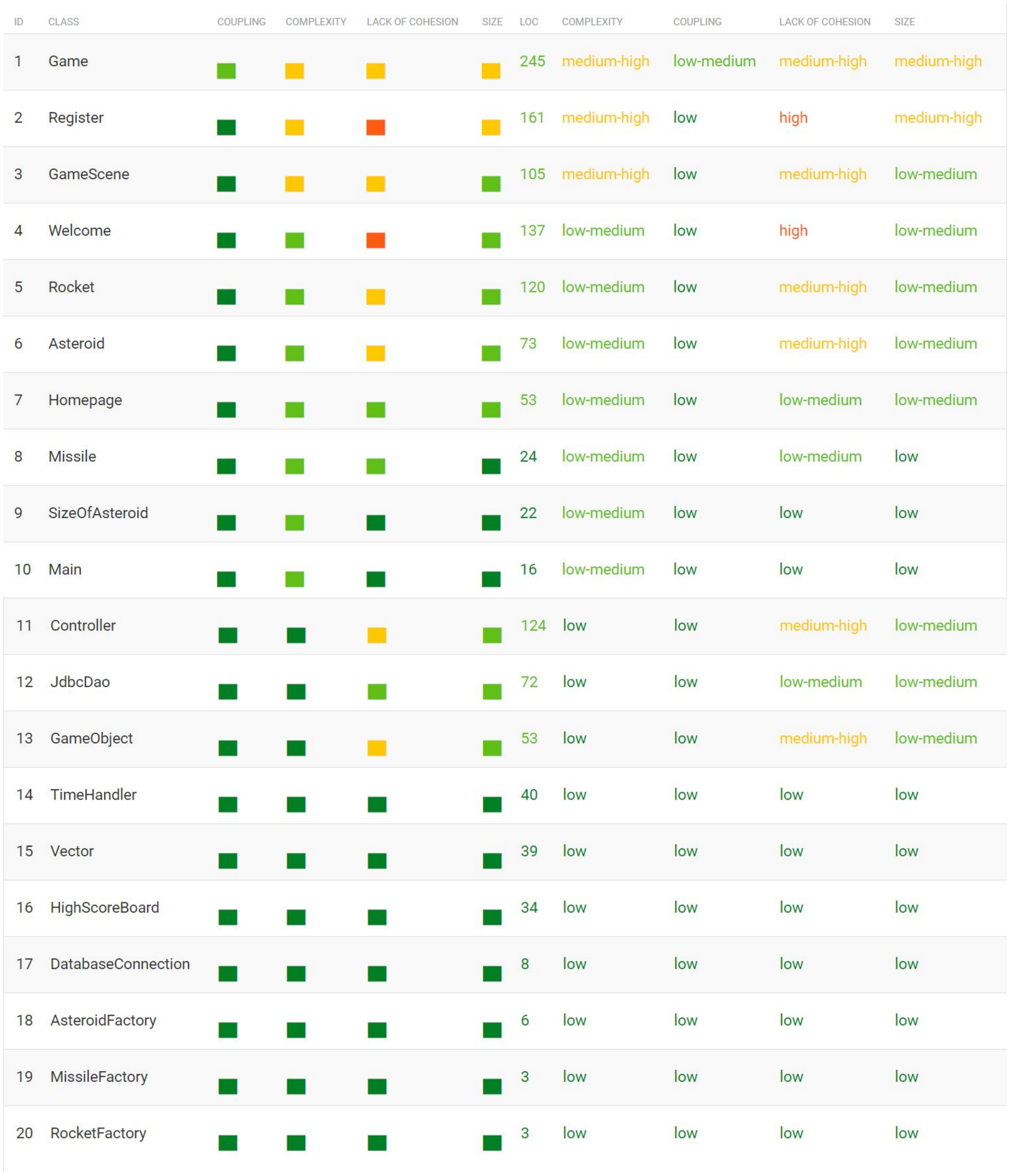
Number of external packages: 50

Number of external classes: 233

Number of problematic classes: 2

Number of highly problematic classes: 0

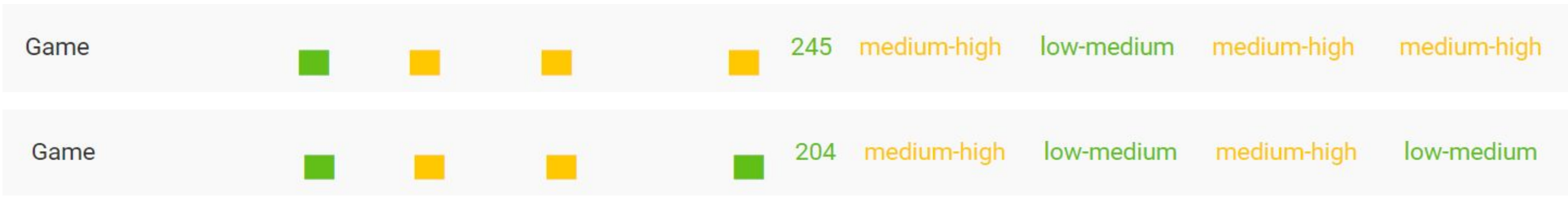
Here you can see the status of classes **before refactoring**:



ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	Game	🟩	🟨	🟨	🟨	245	medium-high	low-medium	medium-high	medium-high
2	Register	🟩	🟨	🟧	🟨	161	medium-high	low	high	medium-high
3	GameScene	🟩	🟨	🟨	🟩	105	medium-high	low	medium-high	low-medium
4	Welcome	🟩	🟩	🟧	🟩	137	low-medium	low	high	low-medium
5	Rocket	🟩	🟩	🟨	🟩	120	low-medium	low	medium-high	low-medium
6	Asteroid	🟩	🟩	🟨	🟩	73	low-medium	low	medium-high	low-medium
7	Homepage	🟩	🟩	🟩	🟩	53	low-medium	low	low-medium	low-medium
8	Missile	🟩	🟩	🟩	🟩	24	low-medium	low	low-medium	low
9	SizeOfAsteroid	🟩	🟩	🟩	🟩	22	low-medium	low	low	low
10	Main	🟩	🟩	🟩	🟩	16	low-medium	low	low	low
11	Controller	🟩	🟩	🟨	🟩	124	low	low	medium-high	low-medium
12	JdbcDao	🟩	🟩	🟩	🟩	72	low	low	low-medium	low-medium
13	GameObject	🟩	🟩	🟨	🟩	53	low	low	medium-high	low-medium
14	TimeHandler	🟩	🟩	🟩	🟩	40	low	low	low	low
15	Vector	🟩	🟩	🟩	🟩	39	low	low	low	low
16	HighScoreBoard	🟩	🟩	🟩	🟩	34	low	low	low	low
17	DatabaseConnection	🟩	🟩	🟩	🟩	8	low	low	low	low
18	AsteroidFactory	🟩	🟩	🟩	🟩	6	low	low	low	low
19	MissileFactory	🟩	🟩	🟩	🟩	3	low	low	low	low
20	RocketFactory	🟩	🟩	🟩	🟩	3	low	low	low	low

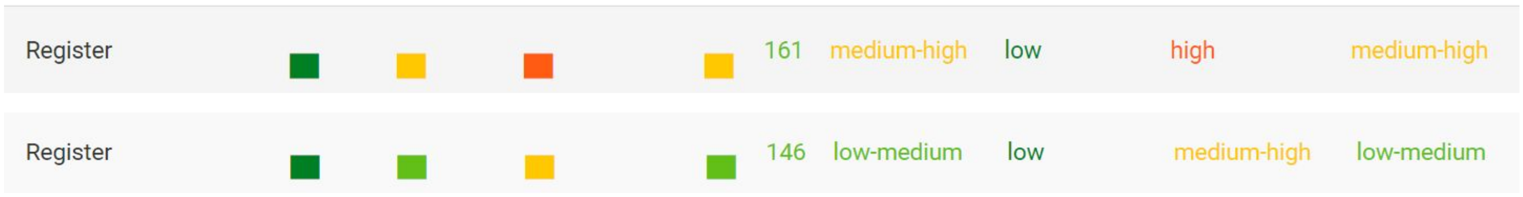
2. We improved the code quality of **at least 5 classes** and **at least 5 of their methods** by applying proper code refactoring as taught in the lectures. You can see the previous and current statuses in the following diagrams below:

1) **Game Class:** The class's previous size metric was medium-high. It is improved to low-medium through refactoring. We split the Game class into 2 classes different classes, Game and GameLogic, hence improving the class-level code. In addition, we split the long methods which were doing too much work into shorter methods to improve method-level code.



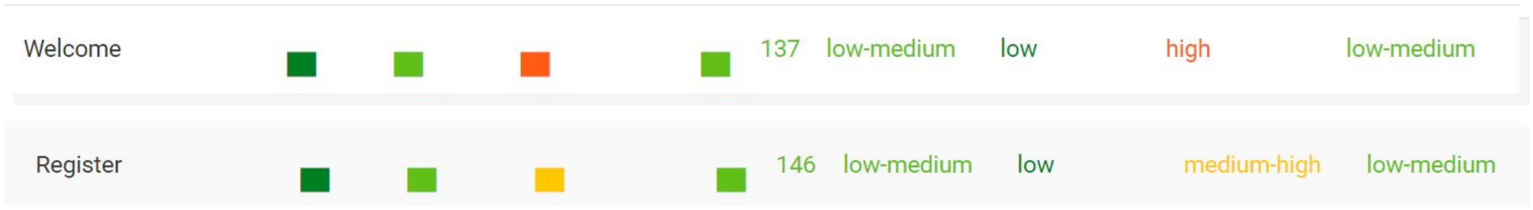
Game	🟩	🟨	🟨	🟨	245	medium-high	low-medium	medium-high	medium-high
Game	🟩	🟨	🟨	🟩	204	medium-high	low-medium	medium-high	low-medium

2) **Register Class:** The class's complexity was improved from medium-high to low-medium. In addition, the lack of cohesion was also improved from high to medium-high, hence improving class-level code in both cases. Finally, size metric was improved from medium-high to low-medium, which improved the method-level code. This was done by refactoring and removing different methods in the Register class.



Register	🟩	🟨	🟧	🟨	161	medium-high	low	high	medium-high
Register	🟩	🟩	🟨	🟩	146	low-medium	low	medium-high	low-medium

3) **Welcome Class:** Its lack of cohesion was improved from high to medium-high, hence improving class-level code. We established this improvement by removing unnecessary methods. In addition, we refactored methods in order to increase the level of cohesion (a method doesn't do too much), hence improving method-level code.



Welcome	🟩	🟩	🟧	🟩	137	low-medium	low	high	low-medium
Register	🟩	🟩	🟨	🟩	146	low-medium	low	medium-high	low-medium

4) **JdbcDao Class**: Its cohesion was improved from low-medium to low, improving the class-level code. We established this improvement by refactoring and removing different methods.

JdbcDao	🟩	🟩	🟩	🟩	72	low	low	low-medium	low-medium
JdbcDao	🟩	🟩	🟩	🟩	64	low	low	low	low-medium

5) **Asteroid Class**: Its previous size metric was low-medium and improved to low, hence improving class-level and method-level code. In addition, the cohesion was improved from medium-high to low-medium through refactoring, hence improving method-level code. We also refactored the Asteroid class. The Asteroid class was split into 2 other classes, Asteroid and AsteroidConstants, hence improving

Asteroid	🟩	🟩	🟨	🟩	73	low-medium	low	medium-high	low-medium
Asteroid	🟩	🟩	🟩	🟩	46	low-medium	low-medium	low-medium	low

class-level code.