Metric Analysis

1. What were the metrics for the codebase? What did these initial measurements tell you about the system?
2. How did you use these measurements to guide your refactoring?
3. How did your refactoring affect the metrics? Did your refactoring improve the metrics? In all areas? In some areas? What contributed to these results?

---

1. McCabe Cyclomatic Complexity

1.1 Measurements tell about the system

- Indicate the complexity of a program.
- Quantitative measure of the number of linearly independent paths through a program's source code.
- Our maximum cyclomatic complexity was set to 10.
- Functions that violated our constrain are (along with their respective cyclomatic complexity):
    1. Lane.java(Lane)
        1. getScore - 38
        2. run - 19
        3. receivePinsetterEvent - 12
    2. LaneView.java(LaneView)
        1. receiveLaneEvent - 19
    3. LaneStatusView.java(LaneStatusView)
        1. actionPerformed - 11
    4. AddPartyView.java(AddPartyView)
        1. actionPerformed - 11
- Inference: the highest cyclometic complexity is of `getScore()` method in `Lane.java` and has a big scope of reduced complexity.

1.2 Guide our refactoring

- Split up method into simpler components.
- Complex loops can be made into separate functions.
- Complex conditional branches can be made into separate functions.
- Use smaller methods.

1.3 Refactoring affect metrics

TODO

---

2. Number of parameters

2.1 Measurements tell about the system

- Our maximum number of parameters was set to 5.
- It should be kept low for simpler understanding of code.
- Functions that violated our constrain are (along with their respective number of parameters):
    1. LaneEvent.java(LaneEvent)
        1. LaneEvent - 9
- Inference: There was just one method that had higher than 5 parameters.

2.2 Guide our refactoring

- We can pass an object instead of high number of parameters.
- We can split the method if the number of parameters can be partioned well with the resulting methods.

2.3 Refactoring affect metrics

TODO

---

3. Nested block depth

3.1 Measurements tell about the system

- Our maximum nested block depth was set to 5.
- It should be kept low for simpler understanding of code.
- Functions that violated our constrain are (along with their respective nested block depth):
    1. Lane.java(Lane)
        1. run - 7
        2. getScore - 7
- Inference: Lane class has methods which violate our constrain and needs improvement.

3.2 Guide our refactoring

- Split up method into simpler components.
- Put deeper blocks into meaningful functions.

3.3 Refactoring affect metrics

TODO

---

4. Lack of Cohesion of Methods

4.1 Measurements tell about the system

- Our limit for LCOM was set to 0.85

- It should be kept low, although it can go high for other reasons too like using getters and setters in java.
- Classes that violated our constrain are (along with their respective LCOM):
    1. LaneEvent - 0.91
    2. NewPatronView - 0.894
    3. LaneView - 0.88
    4. Lane - 0.851
- Inference
    1. LaneEvent has high LCOM because of getters and dosen't need to be split.
    2. NewPatronView has high LCOM because of getters and dosen't need to be split.
    3. LaneView does nees splitting.
    4. Lane view does have one liner functoins, may need splitting.

4.2 Guide our refactoring

- Split up method into simpler components.

4.3 Refactoring affect metrics

TODO

---

5. Method lines of code

5.1 Measurements tell about the system

- Our limit for LCOM was set to 100.0
- It should generally not be high.
- Functions with more lines of code are more bug prone.
- Classes that violated our constrain are: None
- Inference: The code given is strong in this metric.

5.2 Guide our refactoring

- The code given is strong in this metric.
- We should make sure that this is maintained after refactoring.
- If metric constrains are not met, then split funciton into smaller functions.

5.3 Refactoring affect metrics

TODO

---

6. Depth of inheritance tree

6.1 Measurements tell about the system

- Our limit for DIT was set to 5.0
- It should typically be kept between 2 to 5.
- If there is a majority of DIT values below 2, it may represent poor exploitation of the advantages of OO design and inheritance.
- It is recommended a maximum DIT value of 5 since deeper trees constitute greater design complexity as more methods and classes are involved.
- Since the code is relatively small, no lower limit was kept since not much scope of inheritance is there in relatively small code.
- Classes that violated our constrain are: None
- Inference: The code given is strong in this metric.

6.2 Guide our refactoring

- The code given is strong in this metric.
- We should make sure that this is maintained after refactoring.
- If metric constrains are not met, then split funciton into smaller functions.

6.3 Refactoring affect metrics

TODO

---

7. Number of methods

7.1 Measurements tell about the system

- Our limit for number of methods was set to 7.
- Classes that violated our constrain are (along with their respective number):
    1. Lane - 17
    2. LaneEvent - 11
    3. ControlDesk - 11
    4. LaneEventInterface - 9
- Inference
    1. Lane has the highest number of methods per class and should be reduced
    2. LaneEvent has high number of methods due to getters and setters and hence can be ignored.
    3. Control desk has high number and should be reduced.
    4. LaneEventInterface is interface and has getters and setters and hence can be ignored.

7.2 Guide our refactoring

- Split up class into simpler classes with fewer methods.
- Try to merge smaller methods if possible and not violating other metrics.
- Remove dead code and unused methods.

7.3 Refactoring affect metrics

TODO

---

8. Number of classes

8.1 Measurements tell about the system

- Our limit for number of methods was set to 30.
- The number of classes in given package is 29.
- Classes that violated our constrain are: None
- Inference: The code given is strong in this metric.

8.2 Guide our refactoring

- The code given is strong in this metric.
- We should make sure that this is maintained after refactoring.
- If metric constrains are not met, then split package into smaller packages.

8.3 Refactoring affect metrics

TODO

References

1. Points 1, 2, 3, 4
2. Point 5
3. Point 6
4. Point 7
5. Point 8
