Recitation 6 Metric Discussion of Peer Review
Erdun E
October 09, 2024

CS 5010 Programming Design Paradigm
Recitation 6 Metric Discussion of Peer Review

1 Metrics

WMC	DIT	NOC	CBO	RFC	LCOM
54	13	0	15	74	Low (except medium in `ProductionController`)

Table 1: Summary of Metrics

1.1 WMC (Weighted Methods per Class)

Overall, my system got 54 points. The ProductionController class had the most points (9), which makes sense because it does a lot of things like creating and managing batches.

Other classes like Batch, Vat, and Recipe didn't have as many methods, so their WMC scores were lower. This means most of my classes aren't too complicated, which is good.

1.2 DIT (Depth of Inheritance Tree)

Each class has a DIT score of 1 and a total score of 12. All classes inherit directly from Object and do not extend other user-defined classes.

Although the system is not too much in terms of inheritance, this simplicity helps to avoid the problems associated with complex hierarchies, which can make debugging and maintenance more difficult. the DIT is kept uniform throughout the system, which is one of the reasons for the simplicity of its design.

1.3 NOC (Number of Children)

No classes in the system have any subclasses, NOC score of 0. The lack of inheritance indicates that the system avoids hierarchical complexity. This design choice simplifies maintenance, as changes in one class will not cascade through a chain of subclasses.

1.4 CBO (Coupling Between Objects)

The system has a CBO score of 15, reflecting the level of dependency between classes. BreweryControlSystem and ProductionController show the highest coupling, as they coordinate multiple subsystems and perform a wide range of operations. This is expected, given their roles as orchestrators of other classes.

On the other hand, classes like Ingredient, Vat, and Pipe are more self-contained, showing no dependencies. While high CBO can increase system complexity, the current level of coupling is necessary to ensure communication between the core systems (production, inventory, and recipe management).

1.5 RFC (Response for a Class)

The RFC score is 74, which means a moderate to high level of complexity in terms of method invocations. Classes like BreweryControlSystem and ProductionController have higher RFC values due to the number of methods they call both internally and externally. These classes are responsible for coordinating multiple operations, which justifies their high RFC.

The remaining classes, such as RecipeLibrary and InventorySystem, have lower RFC scores, indicating that their functionality is more focused and self-contained.

1.6 LCOM (Lack of Cohesion of Methods)

Most classes exhibit low LCOM and high cohesion, meaning the methods work closely together on related data. For example, classes like Batch, Vat, and Recipe have methods that all operate on the same data attributes, such as batchID, capacity, and ingredients.

The only exception is the ProductionController, which has medium LCOM (moderate cohesion) due to its broader range of responsibilities, such as batch creation, monitoring, and cleaning. This indicates that ProductionController may benefit from being split into smaller, more focused classes.

2 Summary

In the peer review, we compared metrics WMC, DIT, NOC, CBO, RFC, and LCOM. My code's WMC is slightly lower than theirs, which indicates that my class has a lower average complexity. Like other systems, my DIT and NOC scores are low due to a simpler design without inheritance.

However, my CBO and RFC scores are also slightly lower than my peers, which shows that my system has fewer external dependencies and less complexity in method calls. My LCOM scores show that most of my classes are highly cohesive but similar to the others, ProductionController is less cohesive due to the fact that it has to handle multiple tasks.

3 Surprises

The biggest surprise was the LCOM result for ProductionController, which had lower cohesion due to multiple responsibilities. This was a common issue across the group. Additionally, some peers had higher DIT by using inheritance, which I hadn't considered. Lastly, there was variation in RFC, with some peers having higher scores in their controllers, indicating more external method calls.

4 Changes

Based on the peer review, I plan to:

- Refactor ProductionController into smaller, more cohesive classes to reduce complexity and improve cohesion.
- Reduce coupling between systems by introducing interfaces to improve modularity and lower CBO.
- Explore inheritance to reuse common functionality, which could simplify the system and reduce code duplication.