Brewery Problem Metrics and Peer Review Analysis

1. Overview

This report evaluates the current design quality of the *Brewery Problem* project based on five key software metrics — **Weighted Methods per Class (WMC)**, **Depth of Inheritance Tree (DIT)**, **Number of Children (NOC)**, **Coupling Between Objects (CBO)**, and **Response for a Class (RFC)**.
The goal is to identify structural strengths and weaknesses in the system before final implementation and to compare findings with peer designs.

2. Representative Metrics Analysis

Class	WMC	CBO	RFC	Design Insight
BrewerySystem (App Layer)	18	8	12	Acts as the orchestrator coordinating multiple subsystems; higher complexity and coupling are expected for a controller class.
Inventory (Inventory Layer)	7	3	6	Low WMC and CBO confirm focused responsibility and easy testability.
Vat (Plant Layer)	9	0	5	Abstract superclass reused by multiple vat types; good abstraction with shallow hierarchy.
TransferOrder (Production Layer)	8	4	7	Handles cross-component interactions; moderate coupling acceptable due to orchestration role.
MonitoringService (Service Layer)	8	3	7	Moderate complexity; demonstrates clean separation between domain logic and instrumentation.

Summary of Metric Trends

- **WMC:** Most classes stay below 10, showing low cyclomatic complexity and small, cohesive methods.
- **DIT/NOC:** The overall inheritance depth remains shallow (DIT ≤ 2). Only `Vat` and `Sensor` serve as base classes, ensuring extensibility without over-engineering.
- **CBO:** Average coupling ≈ 3–5, largely confined to necessary orchestration points (`BrewerySystem` , `TransferOrder`).
- **RFC:** Public interfaces remain concise (5–10), supporting predictable behavior and ease of unit testing.
- **Cohesion:** Each package maintains strong internal cohesion — plant classes manage physical equipment, recipe classes define data, and services handle logic orchestration.

3. Comparative Discussion

When comparing results with peers, the group observed consistent design patterns but differing levels of centralization:

- Some implementations placed **business logic directly in `Main`**, leading to inflated WMC and CBO values.
- Others used **additional helper classes**, which lowered class-level complexity but slightly increased overall coupling.
- My version emphasized **dependency injection and clean layering**, leading to more balanced metrics overall.

4. Surprising Observations

A key surprise was that even with minimal logic implemented, the **BrewerySystem** already had higher coupling (CBO = 8).
This revealed that architectural dependencies are established early through constructor design, not just through code volume.
Another insight was that data-focused packages (`recipes` , `inventory`) consistently produced the lowest WMC values across all submissions, showing strong alignment with object-oriented design principles.

5. Planned Improvements

Following the peer review, several refinement directions emerged:

- **Refactor orchestration logic:** Delegate some `BrewerySystem` tasks to helper services to reduce CBO and improve separation of concerns.
- **Enhance cohesion tracking:** Continue keeping single-responsibility per class while avoiding hidden dependencies between plant and production layers.
- **Quantitative validation:** Use IntelliJ or SonarQube metrics later to validate WMC and CBO changes after adding complete logic.

6. Conclusion

The current *Brewery Problem* design demonstrates **low complexity, shallow inheritance, and strong cohesion** across all packages.
Coupling remains moderate only where inter-layer coordination is necessary.
Peer comparisons confirmed that the architecture balances readability, extensibility, and modularity effectively.
Future iterations will focus on refining orchestration logic while maintaining this structural clarity.

Author: Yue Wu
Course: CS5010 – Programming Design Paradigm
Date: October, 16, 2025