Brewery Problem Metrics and Peer Review Analysis

1. Overview

This document evaluates the structural quality of the *Brewery Problem* system based on aggregated class metrics across all packages.
Metrics analyzed include **WMC**, **DIT**, **NOC**, **CBO**, and **RFC**, focusing on complexity, inheritance depth, coupling, and class responsiveness.
The purpose is to assess how the architecture scales across modules and where structural refinements are most beneficial.

2. Aggregated Metrics Overview

Package	Total WMC	Total DIT	Total NOC	Total CBO	Total RFC
brewery.app	21	2	0	15	20
brewery.inventory	31	2	0	9	18
brewery.plant	63	5	5	16	34
brewery.production	47	3	1	13	29
brewery.recipes	28	2	0	7	16
brewery.services	25	2	0	11	21
Total	**215**	**16**	**6**	**71**	**138**

3. System-Level Interpretation

Overall Complexity (WMC)

Total WMC = 215 indicates a medium-complex system where logic is moderately distributed.
The *plant* and *production* packages contribute nearly half of the overall complexity (WMC ≈ 110), confirming that manufacturing and orchestration dominate the logic density.
Future implementations should avoid embedding too much process coordination within these two domains.

Inheritance Structure (DIT / NOC)

The cumulative DIT = 16 and NOC = 6 suggest a **shallow-to-medium hierarchy**, mainly from the `Vat` and `Sensor` abstractions in `plant`.
This hierarchy level is healthy—deep enough for extensibility but not excessive.
However, further subclass growth should be justified by distinct behavior rather than by data-only differentiation.

Coupling Between Packages (CBO)

CBO totals show *app* and *plant* as the most coupled modules (15 and 16 respectively).
This aligns with expectations— `BrewerySystem` orchestrates dependencies and `Plant` serves as a physical model layer.
Still, refactoring could decouple shared concerns:

- Introduce **Ports/Interfaces** for scheduling and monitoring dependencies.
- Let `BrewerySystem` delegate coordination tasks to sub-services.

Response For Class (RFC)

Total RFC = 138 shows a balanced exposure of methods across modules.
`Plant` and `Production` account for nearly half, which is reasonable since they encapsulate process operations.
Keeping RFC ≤ 10–12 per class in these areas would maintain manageable complexity during expansion.

4. Package-Level Observations

- **brewery.app** – Central orchestration point with the highest relative CBO/WMC ratio. Simplify by pushing procedural coordination into smaller service classes.
- **brewery.inventory** – Efficient, low-coupling and cohesive. Structure should be retained.
- **brewery.plant** – Dominant in WMC and RFC; strong indicator of concentrated logic. Consider splitting subpackages by equipment type (`vats` , `flow` , `registry`) to reduce cognitive load.
- **brewery.production** – Balanced but trending toward high RFC. Apply Domain Events to reduce direct dependencies on `plant` .
- **brewery.recipes** – Compact and data-oriented. Metrics confirm proper use of value objects.
- **brewery.services** – Moderate complexity with service logic well-isolated. Slightly high coupling suggests potential for port–adapter refactoring.

5. Improvement Focus (Based on Data)

1. **Refactor Core Orchestration**
   Move orchestration responsibilities from `BrewerySystem` into task-specific services, aiming for WMC < 10 per class.
2. **Reduce Cross-Package Dependencies**
   Establish domain interfaces (`MonitoringPort` , `TransferSchedulerPort`) to minimize CBO across packages.
3. **Flatten or Combine Minor Hierarchies**
   `Vat` subclasses could consolidate through composition, reducing NOC proliferation.
4. **Method-Level Optimization**
   Classes with RFC > 12 should hide internal helpers and provide minimal public APIs to stabilize system responsiveness.
5. **Maintain Low Complexity in Data Packages**
   The `inventory` and `recipes` modules exemplify high cohesion; keep their isolation from service orchestration.

6. Conclusion

The metric totals indicate a **well-layered but tightly coupled core**, typical for mid-stage architectural builds.
Complexity remains manageable and inheritance depth is controlled, yet orchestration and process

modules are clear coupling hotspots.
A moderate refactor focusing on **service delegation and interface decoupling** will substantially enhance maintainability and scalability while preserving the current clean layering.

Author: Yue Wu
Course: CS5010 – Programming Design Paradigm
Date: October 20, 2025