Brewery Problem Metrics and Peer Review Analysis

1. Overview

This report summarizes the structural metrics of the *Brewery Problem* project across all packages, based on five key indicators: **WMC**, **DIT**, **NOC**, **CBO**, and **RFC**. It highlights system-level patterns, potential design bottlenecks, and priorities for refactoring.

2. Aggregated Metrics

Package	WMC	DIT	NOC	CBO	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. Key Insights

- **Complexity (WMC):**
  Moderate overall (215). `plant` and `production` account for ~50% of total complexity, showing heavy logic concentration in process layers.
- **Inheritance (DIT/NOC):**
  Shallow hierarchy (DIT ≤ 5, NOC = 6). Most subclasses stem from `Vat` and `Sensor`; further subclassing should serve distinct behavior only.

- **Coupling (CBO):**
  Highest in `app` and `plant` (15–16). Suggests strong orchestration and data interlinking—decouple via service interfaces and dependency injection.
- **Responsiveness (RFC):**
  Balanced exposure. Keep RFC ≤ 12 per class, especially in `production` where orchestration is denser.

4. Package Observations

- **app:** Central controller; simplify by splitting orchestration tasks.
- **inventory:** Well-contained and cohesive—keep structure.
- **plant:** Most complex; consider submodules (e.g., `vats` , `registry`) to lower cognitive load.
- **production:** Slightly high RFC; apply event-driven delegation.
- **recipes:** Lightweight and stable; maintain as data-only module.
- **services:** Slight coupling spikes; refactor through clearer port–adapter boundaries.

5. Improvement Focus

1. **Delegate orchestration** from `BrewerySystem` to sub-services.
2. **Introduce domain ports** (`MonitoringPort` , `SchedulerPort`) to lower inter-package CBO.
3. **Flatten hierarchies**—merge simple subclasses into composition-based structures.
4. **Tighten APIs** by limiting public methods and keeping data modules immutable.

6. Conclusion

The architecture is structurally sound with **moderate complexity and clean layering**, but **coupling hotspots** exist in orchestration-heavy modules. Minor modular refactoring and clearer boundaries will significantly improve maintainability without altering the overall design.

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