Solving the Brewery Problem with Object-Oriented Design

Brianna Dym
b.dym@northeastern.edu
The Roux Institute at Northeastern University
Portland, USA

Julie Jarzemsky
julie.jarzemsky@colorado.edu
University of Colorado Boulder
Boulder, USA

Course Software Engineering & Design
Programming Language Java
Knowledge Unit Object-Oriented Design Paradigms
CS Topics Program Design, Object-Oriented Programming, Coupling, Class Design, Code Review
Resource Type Assignment

SYNOPSIS

This assignment helps students practice designing and implementing the code for small programs. It engages with historical discussions of object-oriented design paradigms within the ACM by requiring students to read the 1993 article "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Models" [4]. In the article, the authors describe two design methodologies they applied toward building a program for managing a brewery's systems, though any production facility can be used in student-facing scenarios. The authors then analyze the strength of the code written using each methodology based on a series of metrics. After reading the article and discussing it in class, students then recreate some or all of the described systems using at least one of the design methodologies outlined. Once complete, students then analyze their code using the metrics outlined in the article and reflect on opportunities to strengthen their program's design in future iterations.

KEYWORDS

Design Paradigms, Programming Practice, Object-Oriented Programming, Software Metrics, Design Patterns

ACM Reference Format:
Brianna Dym and Julie Jarzemsky. December 2025. Solving the Brewery Problem with Object-Oriented Design. In *ACM EngageCSEdu.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3785672

ACM EngageCSEdu, December 2025.

ACM ISBN 979-8-4007-2379-7/2025/12.
https://doi.org/10.1145/3785672

1 ENGAGEMENT HIGHLIGHTS

This assignment provides important context for why object-oriented programming prioritizes certain design philosophies over others, and gives students a historical perspective on how modern object-oriented design paradigms were formed. Students are banned from importing any pre-built data structures and are required to consider how to design each class as a data type responsible for managing a strictly-defined task in their production system. This task promotes a more active method of learning the object-oriented paradigm, encouraging students to research the paradigm and apply the design principles in their own code [3]. The project also introduces students to different metrics used to analyze the quality of a program's design and provides examples to contextualize why those metrics matter. In general, students struggle to switch from a procedural paradigm to an object-oriented paradigm [2]. This assignment seeks to reduce that difficulty through providing a historical context and concrete examples outlined in the 1993 paper [4].

2 ASSIGNMENT DETAILS

2.1 Introducing the Reading

The article this assignment is structured from is a 1993 publication through ACM's SIGSOFT conference [4]. The article describes two approaches to designing an object-oriented system for managing a brewery: data-driven and responsibility-driven. The authors outline self-imposed restrictions on their code and goals for the brewery system. For example, the authors specify that the system should not import any external packages to make their code work, meaning pre-built data structures like hash maps and linked lists need to be be rebuilt for the program. When introducing students to the paper, explain that object-oriented programming was not always a prominent design paradigm in coding. Furthermore, use the article to emphasize the difference between data-driven and responsibility-driven design philosophy. Show students how data-driven design generates objects defined in terms of data while responsibility-driven design generates objects defined in terms of the program's goals.

Walk students through the self-imposed restrictions the authors apply to their code. Explain what the restrictions in

the original problem mean for their own practices as developers. By prohibiting the use of imported libraries, students will have to define their own data structures: linked lists and hash maps are not allowed unless students wish to construct them from scratch. Additionally, make sure that students understand they are to design their production system using a responsibility-driven design philosophy. Once students understand the point of the restrictions, discuss the tasks the brewery system does, highlighting which tasks from the article are required in the assignment.

Finally, discuss the complexity analysis from the reading. The authors use the complexity analysis to emphasize the efficiency of responsibility-driven design in developing the brewery system. Ask students why that might be the case and to compare the metrics between the two design paradigms.

2.2 Reviewing Code with Metrics

To tally the Weighted Methods per Class (WMC) metric, make each line of code 1 point. Every time a semicolon is used to end a statement, this should be regarded as starting a new line of code, including definitions of attributes of a class. Alternatively, students can use WMC calculators that are integrated directly into their development environments.

To tally the Depth of Inheritance Tree, each parent class that a subclass belongs to is assigned 1 point (include imported packages in this analysis). To tally the Number of Children metric, assign each subclass 1 point as well.

To tally coupling between objects, assign 1 point for each method in an object that requires another object to compute its results (for example, a class might use a method that requires data to be passed through from another class). To tally the response for a class, assign 1 point for each method in a class and 1 point each time a method outside of that class is used within the class itself.

Finally, the Cohesion Across Methods metric does not contribute to numerical score, but students should aim for high cohesion. That is, each element of the class should serve a similar goal contained within that class. You might explain the significance of this concept using the following example: if you have a class that is responsible for managing containers and cleaning them, then perhaps that class should not have methods responsible for managing the drinks that a container produces, even though it is used to produce them.

3 RECOMMENDATIONS

We piloted in Object-Oriented Analysis and Design courses at two Universities, Northeastern University and University of Colorado Boulder (CU) and provide recommendations in this section. The reading and programming project can span two to three weeks. Students with a range of previous OO programming experience, from almost no experience to moderate amounts, accomplished the assignment tasks in this time frame. This assignment was used in a course where students had not yet taken data structures to help teach LinkedLists compared to Arrays. Students should have a basic understanding of Java and spent a month engaging in OO principles before this assignment is introduced. This assignment was delivered as a group project in the CU course and as an individual assignment at Northeastern.

At CU, we found it helpful to build a foundation of code and group dynamics during class time. Students created Unified Modeling Language diagrams and received feedback on initial designs in class. They also ran "mob-programming" sessions, where one student types code while others assess the bigger picture and give directions [1]. This approach established healthy team dynamics, offered a refresher on programming language basics and version control, and bolstered confidence within students to complete the program.

Prohibiting imports of external libraries is a potentially sticky requirement for students who frequently use libraries in other courses. Students questioned how they might prompt the user for input to run through scenarios without the use of external libraries. Suggest that students hard-code scenarios to test the program instead of gathering user input or allow java.util.Scanner. In our experience, students benefited from the import restriction, as it required them to consider the design of the system from the data structure level.

For educators seeking to integrate deeper social commentary into the classroom, we encourage modifications to this assignment that modify a "wasteful" production system to be more sustainable and environmentally conscious.

4 MATERIALS

Assignment Handout: Solving the Brewery Problem.docx
Metric Analysis Handout: Metric Analysis Template.docx
Example Code: BreweryProblem.zip

5 ACKNOWLEDGMENTS

We would like to thank the students who worked with us through this assignment as we piloted it.

REFERENCES

[1] [n. d.]. Team programming - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Team_programming#Mob_programming. [Accessed 20-11-2024].

[2] Pamela Catherine Flores, Ismael Sebastian Rivas, and Jenny Gabriela Torres. 2023. Difficulties in Object-Oriented Design and its relationship with Abstraction: A Systematic Review of Literature. In *Proceedings of the 4th European Symposium on Software Engineering*. 1–13.

[3] Said Hadjerrouit. 1999. A constructivist approach to object-oriented design and programming. *ACM Sigcse Bulletin* 31, 3 (1999), 171–174.

[4] Robert C Sharble and Samuel S Cohen. 1993. The object-oriented brewery: A comparison of two object-oriented development methods. *ACM SIGSOFT Software Engineering Notes* 18, 2 (1993), 60–73.