CS-499 Milestone Two  
Enhancement One: Software Design and Engineering

Shawn Henly

05/29/25

### **Brief Description of the Artifact**

The artifact chosen for this enhancement is a Java class named AppointmentService.java, originally created in CS-320: Software Testing, Automation, and Quality Assurance. This class manages contact and appointment records using in-memory HashMap data structures, and it includes basic methods to add, delete, and retrieve Appointment and Contact objects by their unique IDs. At the time of creation, the implementation was minimal but functional. The logic was contained in a single monolithic class with no formal error handling, abstraction, or use of design patterns. Its original goal was to support test-driven development through straightforward logic and testable structure.

### **Justification for Inclusion in the ePortfolio**

I selected this artifact because it represents my technical progression in backend software design and engineering. In its original form, the code lacked scalability, maintainability, and adherence to clean software architecture. The tightly coupled structure offered no modular separation of concerns and left no room for extension or robust error handling.

Enhancing this artifact allowed me to showcase my evolution from writing procedural code to embracing modular, interface-driven development using real-world software design principles. The improvements I made illustrate my understanding of production-ready practices and show that I can take foundational code and elevate it to a more professional, scalable, and testable standard.

### **Key Components of the Enhancement**

* **Interface-based design**: I separated the appointment and contact functionality into two clear interfaces, IAppointmentService and IContactService. This improves abstraction and allows for easier mocking and testing in future expansions.
* **Custom exception handling**: I introduced AppointmentException, a custom runtime exception, to more clearly manage and identify invalid operations.
* **Logging integration**: I integrated SLF4J with Logback to track key events such as object creation and deletion. This simulates real-world observability and prepares the class for deployment into logging-sensitive environments.
* **Application of SOLID principles**: I applied the Single Responsibility Principle by clearly isolating appointment and contact responsibilities. I also applied the Dependency Inversion Principle to rely on abstractions over concrete classes.
* **Input validation and error protection**: I added null checks and duplicate record checks to prevent runtime failures and maintain data consistency.

### **Course Outcomes Achieved**

This enhancement supports several of the program's core learning outcomes. First, it aligns with **Outcome 3**, which involves designing and evaluating computing solutions using algorithmic principles and computer science practices. By refactoring the original monolithic codebase into a more modular structure with interface-driven services and formal error handling, I demonstrated my ability to evaluate trade-offs between simplicity and scalability. These changes led to a more maintainable and extensible solution, showcasing thoughtful design aligned with professional standards.

Additionally, the project fulfills **Outcome 4**, which emphasizes the use of well-founded and innovative tools and techniques in computing. The integration of SLF4J for logging added a layer of traceability and observability to the codebase—skills commonly required in real-world development environments. Beyond tooling, applying architectural patterns like interface segregation and exception-driven design illustrates my readiness to work within industry best practices.

Finally, this enhancement supports **Outcome 5**, which is centered around developing a security-conscious mindset. I introduced proactive validation to prevent null inputs and duplicate records, and I implemented meaningful exception logic to catch and report invalid operations. These additions help ensure that the system can handle user or system errors gracefully, increasing both its resilience and reliability.

Overall, I am confident that this enhancement satisfies the outcome coverage outlined in Module One, and no changes to that plan are necessary at this time.

### **Reflection on the Enhancement Process**

The enhancement process for this artifact deepened my understanding of clean architecture and maintainable software design. Refactoring the original logic into a more modular structure was more complex than I anticipated. Designing interfaces that made the class more flexible without introducing unnecessary complexity required several iterations.

I found integrating a logging framework particularly eye-opening. This was the first time I worked with SLF4J and Logback in depth, and learning to provide meaningful logs while maintaining clarity in code was a key growth point. Exception handling was another challenge—I learned that catching errors gracefully while still logging or surfacing useful information is far more effective than silent failure.

This experience helped me shift from "just making the code work" to thinking more like a software engineer responsible for quality, structure, and scalability. It was an important step toward the type of backend development I aim to do professionally.