Andrew Torrez

October 15th, 2025

CS-499-10454

University Course Planner

Artifact 1: Algorithms and Data Structures

**Artifact Description**

The UniversityCoursePlanner is a C++ command-line application originally developed for CS-300. It allows users to load course data from a CSV file, store it efficiently in data structures, and display course information, including prerequisites. The program includes a simple text-based menu to load data, print all courses, print specific course details, and exit. For my CS-499 Capstone, I enhanced this artifact by improving input validation, parsing accuracy, efficiency, and repository organization. These updates reflect a deeper understanding of C++ data structures and professional software design principles.

**Justification for Inclusion**

I chose to include this artifact because it showcases my growth in **algorithms and data structures**—core skills of a computer scientist. When I first built this program, I relied heavily on external help and did not fully understand the reasoning behind using specific data structures. Through this enhancement, I deepened my knowledge of how and why these structures matter for efficiency, scalability, and maintainability.

The enhanced version uses an std::unordered\_map to store course data for **O(1)** average lookup time, significantly improving efficiency when retrieving courses by their code. I chose unordered\_map over map or vector because lookups are the most frequent operation in this program, and constant-time performance is ideal for that pattern. However, since unordered\_mapdoes not preserve ordering, I later gathered the course codes into a std::vector and used std::sort() before displaying them to maintain an ordered, readable output. This decision demonstrates how I balanced **time complexity trade-offs** between access speed and user-facing clarity.

In addition, I restructured the repository to follow a professional format with dedicated src, include, and data directories, plus a detailed README file. This enhances maintainability, supports future collaboration, and reflects **professional communication practices**. I also implemented **robust parsing** and validation to handle malformed input, ensuring that errors from unexpected commas, missing fields, or inconsistent case do not crash the program. These enhancements directly strengthen the artifact’s reliability, security, and overall usability.

**Achievement of Course Outcomes**

This enhancement primarily demonstrates my mastery of the **Algorithms and Data Structures** outcome. Through the use of an std::unordered\_map for fast O(1) lookups and an auxiliary sorted std::vector for ordered output, I applied and reasoned through performance trade-offs and algorithmic efficiency. These enhancements reflect a clear understanding of how appropriate data structure selection impacts scalability, readability, and usability.

In addition, this artifact also supports **Software Engineering and Design** outcomes through modular code refactoring, improved naming conventions, and documentation, as well as **Security and Validation** outcomes through robust input parsing and error handling. Together, these improvements highlight my ability to design efficient, reliable, and professional-quality software systems.

**Reflection on the Process (Learning and Challenges)**

During the enhancement process, I learned the importance of thoughtful **design trade-offs** and writing code that balances performance with clarity. Originally, I focused only on functionality—getting the data to display correctly. Through this enhancement, I learned to evaluate time complexity, data structure selection, and user experience as interconnected factors.

I also deepened my understanding of **secure input handling**. The original code crashed with unexpected input or malformed CSV data. I resolved this by validating column counts, trimming whitespace, and enforcing case consistency. These defensive programming strategies helped me think more like a professional developer, focusing on reliability and preventing potential vulnerabilities.

A key challenge was **maintaining readability while optimizing performance**. Splitting larger functions into smaller, purpose-driven helper functions improved testability and debugging while keeping the logic clear. Additionally, documenting this logic for others to follow made me appreciate the value of communication and collaboration in real-world projects.

**Updates vs. Module One Plan**

Yes, I achieved all the outcomes I originally planned in Module One. My enhancements strengthened both algorithmic efficiency and robustness while introducing improved security practices and repository organization. One addition beyond my initial plan was refining documentation and folder structure to make the project more collaborative and professional. This improvement supports long-term scalability and aligns with modern software engineering practices.