CS 499 – Computer Science Capstone

5-2 Milestone Four: Enhancement Three: Databases

Malgorzata Debska

Southern New Hampshire University

Prof. Wasim Alim

July 25, 2025

Category One: Software Engineering and Design

The artifact I selected from CS-410: Software Reverse Engineering is an in-depth project focused on analyzing and deconstructing a software program using both static and dynamic analysis techniques. This project required me to reverse engineer a compiled application by interpreting low-level code and reconstructing its high-level logic. I used tools to disassemble and examine the executable, which allowed me to understand how the software functioned without access to the original source code. Through this process, I learned to think critically and carefully dissect software in a way that exposed its logic, functions, and potential vulnerabilities.

I decided to include this project in my ePortfolio because it represents a unique and highly relevant skill set in the world of software engineering. Being able to reverse engineer a program and analyze its behavior, especially without having access to the source code, is an important ability in fields like cybersecurity and legacy software maintenance. This artifact shows how I can look at software from a forensic perspective, figure out what it does and how it works, and communicate those findings in a clear, technical manner.

In preparing the project for my ePortfolio, I enhanced the original version by expanding the documentation and making the structure of the report easier to follow. I reorganized the findings so that the conclusions were grouped by function, rather than just listed in the order I discovered them. I also went back into the decompiled code to add inline comments that explained what each block of code was doing, which made the report much more accessible to readers who may not be familiar with reverse engineering. Working on this artifact helped me meet several key course outcomes, especially in terms of critical thinking, technical documentation, and applying systematic problem-solving techniques. Improving the clarity and structure of my analysis, I also

showed my ability to communicate complex information in a way that's easier to understand and useful to others. These enhancements made the project more professional and polished, which is exactly what I wanted for my portfolio. Looking back on the process, I realized how much I've grown since I first worked on this project. Initially, I struggled to understand some of the lower-level binary patterns and how they relate to the software's higher-level logic. But by sticking with it and using the tools and knowledge I gained throughout the course, I built up the confidence to work through those challenges. I also came to appreciate how important clear documentation is, especially in technical fields where others may need to understand and build upon your work.

Category Two: Algorithms and Data Structures

The second artifact I selected for my ePortfolio is a web-based CRUD (Create, Read, Update, Delete) application originally developed in CS-340: Client-Server Development. This project was created using Python, Flask, and MongoDB, with a Dash web interface to visualize animal shelter data. The application provides interactive features for users to view and filter animals suited for rescue tasks, such as water or mountain rescue. The initial version was completed during the course, but I enhanced and restructured it to better demonstrate my evolving technical skills.

I chose to include this artifact because it reflects my growth as a full-stack developer, especially in applying algorithms and data structure concepts in real-world scenarios. While the original version functioned correctly, its logic for querying and filtering data was tightly coupled to the event callbacks and lacked modularity. In the enhanced version, I made several improvements,

particularly to the algorithms used for filtering and the data structures used to represent conditions and operations.

Key Enhancements

1. **Modular Algorithm Design**:

   In the original version, MongoDB queries for filtering animals based on rescue type (e.g., water, mountain, disaster rescue) were directly embedded within the event handler callbacks. This approach made the logic harder to maintain and reuse. In the enhanced version, I modularized these queries by creating dedicated functions like query_water_rescue(), query_mountain_rescue(), and query_disaster_rescue(). Each function returns a structured MongoDB query in the form of a Python dictionary. This change improved the algorithm's clarity and reusability.

2. **Efficient Use of Data Structures**:

   I used Python's dictionary and list structures to represent complex query conditions with logical operators like $and, $in, $gte, and $lte. For example, breeds eligible for water rescue are stored as a list within a dictionary condition using the $in operator. This approach makes the code more flexible and aligns with best practices for managing data-driven logic.

3. **Code Readability and Maintainability**:

   I added structured inline comments, improved function naming conventions, and introduced visual separators to make the code easier to follow. These improvements help others (and my future self) understand the logic more quickly and make future updates less error prone.

4. **UI and Layout Enhancements**:

   Although the main focus was on algorithm and data structure improvements, I also enhanced the UI consistency by refining layout structure and visual hierarchy. I reorganized the layout components and updated the placement of interactive elements like maps, dropdowns, and data tables to improve user experience.

This artifact demonstrates my skills in creating modular, maintainable code and in designing custom algorithms that solve a practical problem, filtering animal data for specific rescue tasks. The enhancements I made clearly show my ability to transform procedural logic into modular algorithms, while effectively using Python's built-in data structures to represent complex, nested conditions. At the beginning of the course, I planned to meet the course outcome related to designing and analyzing algorithms and data structures for efficiency and maintainability. This enhancement meets that outcome by:

- Refactoring procedural logic into discrete algorithmic functions,
- Using structured data (dicts and lists) for query construction,
- Supporting scalable, maintainable code practices.

At this time, I have no updates to my outcome-coverage plans, as this enhancement fully supports my original goal.

Enhancing this artifact taught me the value of writing modular code and how critical it is to separate logic from implementation details. By extracting logic into standalone functions, I now better understand how to organize software components for readability, reuse, and testing.

A key challenge I faced was determining how to restructure the query logic without breaking existing functionality. Testing each enhancement incrementally helped ensure correctness. I also

had to ensure that the logic accurately filtered animals by age, sex, and breed requiring careful use of MongoDB's query syntax.

Ultimately, this enhancement strengthened both my technical and problem-solving skills. It transformed a working project into a more professional and maintainable application that demonstrates my ability to work with algorithms, data structures, and full-stack web development.

Category Three: Databases

For the final category, I selected a contact management system that I initially created in CS-405. It tests the behavior of a dynamic collection using the std::vector class and the Google Test framework. This test suite is designed to verify operations like adding elements, resizing the collection, clearing values, checking capacity, and handling exceptions. Although not tied to a formal database system, the project simulates record-based operations similar to basic CRUD functionality found in contact or data management systems, making it relevant to the Databases category.

I selected this artifact for my ePortfolio because it demonstrates my skills in object-oriented programming, testing, and defensive programming all critical competencies in software development. The original version of the test suite (test.cpp) functioned correctly but lacked clarity, structure, and coverage in certain key areas. Specifically, it had minimal inline

documentation, limited exception handling, and did not fully test edge cases like exceeding maximum capacity or accessing out-of-bound elements.

As part of this milestone, I significantly improved the artifact by enhancing the codebase in NEWtest.cpp. To strengthen the original code, I made several numbers of key improvements:

- I added clear and detailed inline documentation throughout the test suite to explain the purpose and logic of each function and test case.
- I expanded exception handling by writing tests that explicitly check for errors, such as accessing invalid indices (std::out_of_range) and attempting to reserve memory beyond the collection's maximum capacity (std::length_error).
- I incorporated defensive programming practices, such as validating preconditions, confirming expected outcomes after each operation, and designing tests that handle edge cases gracefully.
- To increase test coverage and scalability, I introduced parameterized tests that automatically run multiple scenarios using different input sizes (0, 1, 5, and 10).
- I improved the readability of the code by refining test case names to clearly reflect their purpose, such as ResizeIncreasesCollectionSize and AssignValuesToCollection.
- I reinforced proper memory management by using std::unique_ptr for the collection, along with structured setup and teardown methods to ensure a clean test environment for each case.

These enhancements significantly improved the clarity, maintainability, and reliability of the code. They also reflect my understanding of software engineering best practices and my ability to apply those practices in a meaningful way.

This artifact directly supports the course outcomes I set out to achieve in Module One. I applied modular design, test-driven development principles, and robust error handling. The enhancements I made helped me go beyond simply writing tests I learned to think critically about edge cases, failure conditions, and the long-term sustainability of the code. At this point, I do not have any updates to my outcome-coverage plans, as this artifact fully demonstrates the skills I intended to strengthen.

Reflecting on the enhancement process, I gained a deeper understanding of how valuable thorough testing and thoughtful design can be in ensuring software quality. One of the challenges I faced was determining how to best simulate database-like operations within a C++ environment that doesn't use a database engine. I focused on core behaviors like adding entries, resizing collections, and validating boundaries, and ensured these were covered by meaningful, automated tests.

This process helped me transition from simply making code “work” to engineering code that is reliable, readable, and resilient. It reinforced the idea that strong testing and defensive design aren’t optional, they are essential to building professional-grade software. Overall, this artifact reflects not just my technical growth but also my evolving mindset as a developer who prioritizes quality, clarity, and long-term maintainability.