# **CS 499 Milestone Three Narrative**

## **Artifact Description**

The artifact I selected for this milestone is my Contact Management System from CS‑320: Software Testing, Automation, and Quality Assurance, originally developed in Spring 2023. It is a Java application (with unit tests in JUnit) that provides Create, Read, Update, and Delete operations for Contacts, Tasks, and Appointments, storing records in in‑memory collections and enforcing field‐length and format constraints.

## **Justification for Inclusion**

I chose this artifact because it highlights my proficiency with core data structures and algorithmic optimization, skills vital for any computer science professional. In my enhancement, I replaced the original ArrayList‑based storage with HashMap<String, T> lookups to reduce CRUD time complexity from O(n) to O(1). I also integrated a lightweight benchmarking framework (using System.nanoTime()) to measure performance across increasing dataset sizes. These improvements demonstrate my ability to evaluate and apply the right algorithmic principles and measure their real‐world impact.

## **Course Outcomes Reflection**

In Module One, I set out to demonstrate the outcome “Design and evaluate computing solutions using algorithmic principles while managing trade‑offs.” By selecting and justifying the use of hash maps, and by balancing memory overhead against lookup speed, I have fully met that goal. I also leveraged innovative tools and techniques to build a benchmark suite, aligning with the outcome of using well‑founded, innovative computing practices. Remaining outcomes to address include “Design, develop, and deliver professional‑quality communications,” which I will strengthen by polishing my benchmark reports, and “Employ strategies for building collaborative environments,” which I will support by soliciting further peer reviews on my code.

## **Process Reflection**

Working through this enhancement taught me the practical nuances of algorithmic trade‑offs: unordered\_map in Java gives average‑case O(1) performance but uses more memory than lists, and its worst‑case behavior depends on hash distribution. Crafting reliable benchmarks was challenging due to variability in JVM warmup and garbage collection; I overcame this by discarding outlier runs and averaging multiple iterations. I also learned to structure timing code cleanly so it doesn’t pollute business logic. Overall, this process deepened my skills in data‐structure selection, performance measurement, and systematic problem‑solving—key competencies for my future career in software engineering.