**CS 499 Capstone Project**

**Professional Self-Assessment**

Andrew Emilio DiStefano

SNHU

Professor Neil Kalinowski

June 21, 2025

1. **Discuss how completing your coursework throughout the program and developing the ePortfolio has helped you showcase your strengths, shape your professional goals and values, and prepare to enter or become more employable in the computer science field.**

When my time in the SNHU computer science program began, I already had some experience coding. I was skilled in some areas, but I always found that I was lacking understanding of key components required to make a project work as intended. I spent a lot of time looking for answers in the wrong places, and I wasted time learning skills which are either irrelevant to my desired career path or so obsolete that knowing how to navigate them does nothing more than take up valuable real-estate within the mind. I was committed to becoming a software engineer, but I lacked the proper guidance to make my dreams a reality.

SNHU has turned me from a hobbyist coder to a legitimate computer science professional and software engineer with marketable real-world and industry-specific skills. My time in the computer science program has filled in the empty spaces in my understanding of technologies relevant to my planned career trajectory, allowing me to achieve things which I could only have dreamed about before.

I started my educational journey with the lofty (at the time) aspiration of becoming a robotics software engineer specializing in autonomous mobile robots (AMR). I made the decision to pursue this field for a number of reasons including the fact that I had already been working as a recruiter in the robotics industry, and I saw how impressive some of their portfolio projects were. *Surely*, I thought, *if others can do it then so can I*. As a recruiter I was aware that companies looking for robotics software engineers would almost certainly not consider someone without a college degree, and that an associates degree would probably not hold much weight in today's competitive market. I decided to enroll in SNHU’s BSCS with a focus on Software Engineering program. I began to really enjoy what I was doing since I had acquired the appropriate skills and received the appropriate guidance to form an understanding for what exactly I was learning, and how software and hardware components are made to work together to breathe life into our modern world.

The SNHU computer science course titled *Programming Languages or Object Oriented Programming* marked my first time writing *C++*programs with any level of complexity. The material was simple enough to be learned by a novice yet complex enough to really communicate some of the core concepts of how *C++* and OOP can be used to exert much more control over system functionality than is possible when coding in other popular programming languages. This was a great introduction to what will certainly be the most relevant programming language to my planned career path.

In the course titled *Client-Server Development*, I learned how to configure machines to communicate with each other via the internet and/or a local area network. This was the first time that my code had ever interacted with a database. I found the guidance which I received throughout this course to be highly effective at communicating the source material in a way which translates to actual skills. I developed a Python CRUD module which uses PyMongo and MongoDB to perform operations on data sets and to access data quickly when it is needed. The end-result was a complete Python module which allows for the creation, reading, updating, and deleting of data within a non-relational MongoDB database.

The course titled *Data Structures and Algorithms* which was one of the more difficult courses included in the program, and was definitely one of my favorites. This course involved the development of complex data structures which are used in the professional world including hash maps and binary search trees. Putting together an effective binary search tree was a significant milestone for me as a professional, and I believe that this course was what really turned me from a coder to a software engineer. Managing the C++ memory optimization chaos was an excellent opportunity to test my ability and all that I had learned at that point during my time in the program. I was able to complete all of the assignments in that course, and my final grade was almost 100%. One of the most powerful motivation tools for me is the feeling of *eureka* which is felt upon achieving a difficult goal. The motivation which I experienced during my time spent working on the assignments for this course still affects me even today.

The completion of the *MemoryBot* artifact enhancements for this capstone project has undoubtedly been the most rewarding experience for me during my software engineering journey. Using my original Python code as an outline, I was able to rewrite the original program in C++ in a matter of hours. I then began researching data file types to see which would be the optimal choice for storing the question-answer pairs for the conversational agent. I decided to store my data in JSON files due to their compact nature and ease of accessing specific data. JSON files are compatible with virtually all mainstream database options. This means that data from users is always sitting in a format which would be very easy to send to a database as structured data in an incredibly simple form.

Then I opted for the most effective data structure and search algorithm for the purpose of quickly identifying the closest match between user input and vast data stores of questions and answers to be used in conversation. My research led me to the use of an inverted index data structure accompanied by a linear merge count algorithm to effectively and efficiently search among large amounts of data for the string which shares the most tokens (words) in common with an input string. I am rather proud of the finished product, and I even added some functionality which further leverages the inverted index and conversation structure to facilitate conversation with deceased historical figures (both real and fictional).

As is the case with the application’s conversational assistant, the more data is added to the JSON file holding the question-answer (key-value) pairs for the deceased historical figure characters, the more precise and relevant the answers become. These agents have an average of about 200 question-answer pairs in their respective JSON files, and it really is impressive how even with this small amount of data, conversation loops do a decent job of holding tone and staying relevant with responses. My *Benjamin Franklin* character has over 400 pairs in its JSON file (which is still a miniscule amount of data), and there is a significant difference in the quality of responses output by this agent versus the conversations generated by agents with 150-250 pairs. I can only imagine how precise conversations would feel with an agent which has hundreds of thousands of question-answer pairs. The efficiency of the inverted index data structure with the linear merge count algorithm for identifying the most similar response string for an input string means that the system would be able to handle data structures of this size relatively effortlessly.

My security functions also do a great deal to demonstrate my ability to write secure C++ code. All user input in the MemoryBot project is handled by custom functions where what reaches *std::cin* from user input is heavily filtered and limited. Since the program only uses string and int type variables, I focused on security measures relevant to these two data types. I can safely say that any malicious actor would have to try some really unconventional hacking or cracking techniques to perform an input attack on this system. All special characters not used in traditional English language punctuation are filtered out. Strings are allowed a maximum of 500 characters for questions and answers and 30 characters where less input would be needed, and all characters are made lowercase before they are stored (with the exception of the answer values of question-answer pairs stored in JSON). Edits are made before the information enters the program, which means that buffer overflows and command injection attacks are effectively mitigated against. My input function for handling *int*-type input assures that input is stored as the correct data type while also restricting input to a range between 0 and 300. Buffer overflow attacks and input attack types which involve entering the incorrect data type are effectively mitigated against. These measures show a commitment to security while engineering systems written in *C++,* and an ability to construct a system of security functions which are written for specific use-cases. Separate functions are called in the *getSafeStringInput* and the *getSafeIntInput* functions for specific sanitization and validation steps, which shows my ability to write modularized and reusable code. I will certainly be keeping this project on my desktop for whenever I need these security functions in future projects.

I made sure to carefully document each of the functions in the *InvertedIndex* and *Rememberer* classes. I usually have more of an issue with providing too much documentation than not enough. I made sure not to over-comment my code, but to explain the functionality where it may not be apparent at first by simply looking at the code.

My selection of the inverted index as my data structure and the linear merge count algorithm show my ability to evaluate solutions in a specific computing scenario while managing trade-offs. The inverted index is incredibly efficient for string comparison querying, but adding elements to the index has a significantly less impressive time complexity. For this reason, I use the *nlohmann* C++ JSON parsing library (linked and cited in my project submission) to perform CRUD operations on my JSON files rather than doing so through the *InvertedIndex* class. This way I am taking advantage of the advantages of inverted indexing while using other methods to complete tasks where the data structure falls short.

The MemoryBot program’s architecture is so modularized and the functions so reusable that the addition of new entities for conversation is relatively straightforward. To prove this, I added a *TALK TO DEAD PEOPLE* menu where approximations of historical figures (both real and fictional) are brought to life using the same *JSON* question-answer (key-value) pair structure used for conversations with *MemoryBot*. Variables are set for the full name, first name, appropriate filename, and three vectors loaded with output for starting conversations, ending conversations, and telling the user that the system is unsure of the correct answer. In order to add characters to the *TALK TO DEAD PEOPLE* menu, a collaborator would only have to add around 20 strings to specify these variables for each new addition and prompt a free AI service such as *ChatGTP* to produce question-answer (key-value) pairs in JSON format, copying and pasting into the *JSON* file created for the new character. Anyone around the world could use this software to create their own entities for conversation to fascinate friends and family, give students the chance to talk to historical figures, or any number of other use-cases. The user- and developer-friendly nature of this project displays my ability to foster collaborative environments and enable diverse audiences to collaborate.

The completion of my PlantSitter artifact enhancements for this capstone project also garnered a significant sense of achievement. This artifact was developed as a project for the SNHU computer science course focused on embedded systems development. The original artifact involved using simple sensors on a circuit controlled through a Raspberry Pi running a headless version of Ubuntu 24.04. My enhancements not only required the addition of new components to my circuit, but also involved the complete reworking of the original program from a simple state machine to two separate state machines which run simultaneously with functions on a series of threads and asynchronous functions. I was able to get a second LED up and running on my circuit using the same GPIO pins used for the first screen, and I was able to correctly configure my state machine logic to run each machine concurrently. The Python language’s built-in libraries handle a lot of this automatically, but my use of asynchrony and Python “multithreading” make this system rather efficient for a Python program.

Rather than simply printed sensor readings to the circuit’s LCD screen as was the case with the original artifact, the enhanced program writes temperature and humidity readings to a database. This opens up a wide range of possibilities for data analysis and visualization which could enable agricultural production clients to optimize their greenhouse conditions for better harvests. These solutions and techniques deliver value by accomplishing goals specific to the agricultural industry.

The PlantSitter system involves a lot of components which are specific to embedded systems engineering. Seeing an abundance of numerical representations such as model numbers and GPIO pin specifications can be intimidating for people who aren’t used to seeing this type of code. My documentation and commenting throughout this program makes it very clear what each line of code is doing, and someone from a discipline other than embedded systems programming would probably be able to infer how to make simple changes to the system (such as changing the minimum and maximum set temperatures, for example). I think that my dedication to making the application developer-friendly really shows in the finished product. My professional-quality communications are easy to interpret, and my strategies for effective documentation enable collaborative environments which have the capacity to enable diverse audiences in the computer science field.

My strategic arrangement of hardware and software to create the smart greenhouse circuit and the software which runs on it shows my ability to use algorithmic principles to solve computer science problems. Through the use of Python “multithreading” and asynchronous functions, I made the system quite efficient for a Python program. Leaving this project in Python rather than porting to *C++* was a decision made in favor of accessibility over potentially unnecessary efficiency optimization. The Raspberry Pi 4 B has more than enough computing power to easily run a program like this, especially with the current system architecture of PlantSitter. Writing this in C++ would increase efficiency, but the improvement probably would not be very noticeable. This tradeoff was decided based on considering the pros and cons of either implementation and making an informed decision based on prior experience and research.

My constant variable checks and validation measures make sure that the appropriate variable types are being used throughout the system. My functions not only check for inappropriate data types, but they automatically change them as needed to bring down the possibility of unintended behavior (whether caused by accidentally or purposefully malicious input) as low as possible. I made sure to use the appropriate Python functions and parameters to accomplish the goals of the PlantSitter system while allowing for as few unknowns as possible. Much of the security of Python code is due to the nature of the language itself, and the fact that engineers are limited from certain functionality in order to avoid damage to the system as a result of ineffective administration. That being said, the measures taken to make sure that this Python program is as secure as possible demonstrate my ability to maintain a security mindset even when coding in languages where security is usually not as big of an issue.

1. **Summarize and introduce how your artifacts fit together and inform the portfolio as a whole.**

The artifact enhancements made for my capstone project closely align with my goals as a professional. The techniques and skills which are demonstrated through the successful completion of these projects reflect practices used in the robotics industry. As a robotics recruiter, I have personally sought out professionals with these skills for employment with my clients.

Inverted indexes are used in simultaneous localization and mapping (SLAM) algorithms which enable autonomous mobile robot (AMR) systems to navigate through environments where they are unable to communicate with those handling or commanding them. Inverted indexing allows image data to be compared for the association of visually detectable characteristics in an agent’s surroundings with specific locations or areas on a map. The autonomy afforded to intelligent AMRs through the use of SLAM algorithms with inverted indexing has enabled the use of these systems in high-stakes applications such as military operations, search and rescue, and law enforcement. My successful creation of a class containing this data structure and algorithm shows that I am able to understand and implement this data structure.

The MemoryBot project shows potential employers and contracting officers that I am able to achieve impressive functionality on extremely limited hardware. I actually tested the program on the Raspberry Pi 4 which is running my PlantSitter project and everything works flawlessly, even while the PlantSitter program is also running on the system. The program could include four JSON files each with hundreds of thousands of question-answer pairs, the entire program would still take up only a miniscule amount of memory while performing operations in milliseconds. The ability to program for efficiency on limited hardware is incredibly important in the autonomous mobile robotics industry. This is especially the case in unmanned aerial system development since a decrease in hardware requirements leads to a decrease in weight, and therefore an increase in lift power and/or battery life.

My demonstration of secure *C++* coding skills in the development of the MemoryBot project shows my proficiency in this incredibly important aspect of robotics programming. When databases are hacked, people’s information could be stolen or used against them. The hacking of certain robotics and embedded systems, however, could easily result in serious injury or death to humans. Since so much robotics software is written in *C++*, and since *C++* requires constant consideration of security principles more than many other popular coding languages, my enhancement of this artifact has been an excellent opportunity to showcase this important skill.

The functionality of the *PlantSitter* project also certainly involves robotics-adjacent technologies and techniques. The same circuitry principles used in the *PlantSitter* project can be used to create autonomous mobile robots. The PlantSitter system is just an embedded system that takes input data from sensors and engages in the response behaviors deemed appropriate in order to achieve system goals. This is essentially a simple version of what an AMR system is programmed to do, albeit robotics systems are generally programmed with the ability to act on the environment rather than simply interpreting its conditions. The many parallels between robotics development and embedded systems development are enough for a wide range of AMR software development companies to accept professionals with embedded systems development experience, which is a fact which I can attest to as a professional.

Finally, the *TALK TO DEAD PEOPLE* menu included in the *MemoryBot* program shows that I am highly invested in my projects, and that I like to go the extra mile to make my projects extraordinary. I came to the realization that the same principles used in this program could be used to create a highly efficient necromancy simulator capable of searching enormous data sets for the right answers to questions in milliseconds. I decided to test this, and the fact that everything worked as intended shows that I take modularization and object-oriented programming principles very seriously.

The skills showcased in these artifact enhancements come together to display a deep understanding for a wide range of tools and technologies used in the autonomous mobile robotics industry. I am ready to dedicate myself to contributing to the betterment of my organization through the implementation of the experience and skills which I have developed during my time in the SNHU computer science program.