**Milestone Two: Software Design and Engineering Enhancement**

Harrison Bergeron

CS 499: Computer Science Capstone

Prof. Brooke Goggin

July 14, 2022

For the software design and engineering component of the project, I included an artifact from CS 250 that is an interactive slide show application written using Java and its built-in Swing GUI library. The application features “next” and “previous” buttons for a user to cycle through a total of five slides about popular destinations for wellness getaways with text captions describing each location. This project was created to practice GUI design with a strong focus on iterative development techniques using the software development life cycle. I selected this artifact to showcase my skill of iterative development using the Agile methodology to incrementally develop code and create deliverables through weeklong sprints. This artifact is the culmination of multiple software development iterations to improve code and add features to the interactive slide show interface. It also demonstrates my ability to solve problems and create software by algorithmically and programmatically building a GUI (CS-499-04). Creating this GUI through code shows that I can design interfaces without the help of visual cues provided by GUI creation tools that hold the developer’s hand but are somewhat inflexible. I enhanced this artifact by addressing a common software industry goal: porting an application to another programming language with a focus on efficiency. I recreated this slideshow application using C++ instead of Java, and the Qt API instead of Swing. The new C++ and Qt GUI mimics the original implementation, but besides simply being a ported version of the program, it improves on design, security, and documentation.

The course objective I intended to meet through creating the enhancement to this artifact is CS-499-04, which stresses software design and engineering techniques. I used innovative skills and techniques for implementing design solutions and accomplishing goals by manually translating code structure and syntax to convert a codebase to a different language and API and improve efficiency. I demonstrated the ability to program solutions to solve logic problems in software by redesigning and refactoring existing code to more concisely create and add new slides to the slide show. The original application replicated code blocks unnecessarily and used loops improperly, and the new implementation grouped repetitive code into reusable functions and removed loops that were not needed in creating slides. I showed that I can address design flaws related to security by validating command line arguments that are supplied to Qt, which prevents users from entering potentially malicious input that could affect how the program executes, and this targets the CS-499-05 course outcome that seeks to mitigate security flaws. I also articulated my ideas in software development by translating lines from the original codebase to the new one to ensure that my new implementation exactly mirrored the original application as I envisioned.

In enhancing this artifact, I gained entirely new experience with the Qt API. While I have plenty of C++ experience, I had never used Qt to develop a GUI. I learned that while some aspects of porting software to other languages and APIs are highly straightforward, there are other aspects that are more complicated and require more creative solutions. For example, mimicking the CardLayout class in Java’s Swing API was as simple as using the Qt analog QStackedLayout. Aside from the class name and syntax for declaring the layout variable, the translation was direct and easy to implement. On the other hand, adding actions to buttons using Qt was more in depth and not as straightforward. In Java and Swing, actions were added by instantiating anonymous listener classes with custom actionPerformed() methods. In C++ and Qt though, actions were added by creating “slots” in header files that would respond to “signals” emitted by Qt widgets, and signals had to be connected to slots using a separate middleman function. At times, research and creativity were required to translate Java code to its C++ equivalent. Another challenge I faced when enhancing this artifact was deploying the completed executable. I had previously deployed the Java implementation simply by packing the compiled Java bytecode into a .jar file that could be executed by double clicking the icon. The same process was not as easy to accomplish in C++. I had to ensure that shared libraries that were not included with the output executable were copied into the final product so that the software would function properly. In doing this, I needed to conduct more research about how to use the “windeployqt” executable through the command line to collect the required binaries and dependencies for deploying the final executable.

A picture containing application

Description automatically generated*Left: Original Java Swing Application — Right: Ported C++ Qt Application*