Team Controller
Northern Arizona University
Flagstaff, Arizona
October 20th, 2023

Zachary Parham (Team Lead): zjp29@nau.edu
Italo Santos (Mentor): ids37@nau.edu
Bradley Essegian: bbe24@nau.edu
Brandon Udall: bcu8@nau.edu
Dylan Motz: djm658@nau.edu

Tech Feasibility

Northrop Grumman

Weapon System Support Software

Harlan Mitchell

Laurel Enstrom

Table Of Contents

Table Of Contents..... 2
1.0 Introduction..... 3
2.0 Technological Challenges..... 4
2.1 Programming language..... 4
2.2 Graphical User Interface..... 4
2.3 Serial Communication..... 4
2.4 Installer..... 4
3.0 Technological Analysis..... 4
3.1 Programming Language..... 5
3.1.1 Desired Characteristics..... 5
3.1.2 Alternatives..... 5
3.1.3 Analysis..... 6
3.1.4 Chosen Approach..... 8
3.1.5 Feasibility..... 9
3.2 Graphical User Interface..... 9
3.2.1 Desired Characteristics..... 9
3.2.2 Alternatives..... 9
3.2.3 Analysis..... 10
3.2.4 Chosen Approach..... 11
3.2.5 Feasibility..... 12
3.3 Serial Communication..... 12
3.3.1 Desired Characteristics..... 12
3.3.2 Alternatives..... 13
3.3.3 Analysis..... 14
3.3.4 Chosen Approach..... 16
3.3.5 Feasibility..... 16
3.4 System Installer..... 17
3.4.1 Desired Characteristics..... 17
3.4.2 Alternatives..... 18
3.4.3 Analysis..... 19
3.4.4 Chosen Approach..... 21
3.4.5 Proving Feasibility..... 22
4.0 Integration..... 23
5.0 Conclusion..... 24
6.0 References..... 25

1.0 Introduction

Northrop Grumman is a multi-billion dollar defense contractor whose equipment stock the hangars and garages of the United State's military. A cornerstone in the contracting space, they have built everything from the B-2 Stealth Bomber to the James Webb Space Telescope [1]. It's no surprise that these huge projects create hundreds of thousands of diagnostic data. In today's climate, ensuring that our military armament systems are functioning optimally is not just a matter of national security; it is a global concern.

Our clients are Harlan Mitchell, the Senior Systems Engineer Manager, and Laurel Enstrom, the Principal Systems Engineer, for the armaments sector at Northrop Grumman. Along with pioneering new weapon systems, our clients must also endure a cumbersome and resource-intensive approach for problem diagnosis with their products. This results in higher expenses, longer downtimes, and hinders the overall operational efficiency of Northrop Grumman. Getting the aforementioned diagnostic data is difficult and expensive for our clients. Currently, when a problem arises, the standard procedure is to dispatch an engineer to the customer's location. Not only is this time consuming but also resource intensive, oftentimes requiring multiple trips to come up with a solution for their customer. They must physically examine the armament systems and bring an engineering tool to understand and then rectify the issue. The development of a new diagnostic tool for end users is essential. However, the complexity and critical nature of armament systems present unique challenges that must be addressed.

To revolutionize the process of diagnosing and resolving issues with military defense systems, Team Controller and Northrop Grumman present a new solution—a simple, easy to use, and secure desktop application that implements a graphical user interface (GUI) to display and download diagnostic information. This product shall be able to read diagnostic data via serial communication, directly interfacing with the weapon controller. This is similar to how an OBD2 scanner functions in the automotive industry. The main objective of our solution is to provide the end-users with the ability to easily collect diagnostic data without the need for dispatching engineers to their location. Team Controller has identified the key design challenges for this project to be: a desktop application, which is a GUI, with serial communication, that requires a system installer. This document will address our solution to these challenges and analyze potential technologies that will aid in the implementation of this software.

2.0 Technological Challenges

There are four main challenges associated with this project that will need to be addressed for successful development of the weapon system support software.

2.1 Programming language

The programming language the team uses will be a pivotal factor in the success of this project. The desktop application created using this language will need to be OS agnostic and able to run on both Windows versions 10 and 11.

2.2 Graphical User Interface

The graphical user interface is where Northrop Grumman can navigate to the specific code type and categories produced by the weapon. The graphical user interface must be simple to use and easy to understand as well as being professional enough to show clients of Northrop Grumman.

2.3 Serial Communication

This is how the desktop application will communicate with the weapon controller and read its diagnostic data. The application must decipher which communication protocol is in use at which point the controller will send the diagnostic data to be displayed and read by Northrop Grumman's clients.

2.4 Installer

The application will have a simple installer that must act without admin privileges or time consuming setups. The installer must work with as little IT support as possible to allow the quick analysis of the controller data.

3.0 Technological Analysis

Each of the above technological challenges will be dove into to reveal the nuances and potential solutions to each. The team will begin with the programming language, transition to reading the controller's serial input, and finally the application installer. For each of the listed challenges, the team will first discuss the desired characteristics of the ideal solution. Next, we will introduce some alternative solutions and evaluate each in comparison to the ideal solution. Finally, we will rate and explain the chosen approach.

3.1 Programming Language

The desktop application and more importantly, the programming language, is the base platform for this project. All functionalities discussed in the challenges will need to be implemented here.

It is vital that the team select the programming language that will accomplish all the challenges listed above.

3.1.1 Desired Characteristics

The following section will discuss the characteristics the team looks for when choosing a programming language. For each of the characteristics, there will be a point system to clearly and fairly choose a dominant language. The point system is out of 5 points, and any deductions will be shown in **Table 3.1**.

- **OS Independent:** The selected programming language must be able to run on different versions of Windows, namely 10 and 11. A stretch goal is to be able to have the code run on Linux operating systems.

- **Speed:** Due to the large amount of data that this project will be dealing with, the program must be fast to show the diagnostic data in real time. The team does not know the specifications of the computers this software will run on. That being said, the software should be efficient enough to run anywhere.

- **Modularity:** To prevent Team Controller from getting tied up in regulatory documents and courses, Northrop Grumman decided to generalize the project. Upon handoff, the weapon system support software must be modular to assist in-house software engineers in adding new functionalities.

- **Comfortability:** To aid in the development process of the weapon system support software, the programmers must be able to code reliably in the given programming language.

3.1.2 Alternatives

All three languages listed below were specifically mentioned in the client generated project overview document.

- **C:** Created around the 1970's by Dennis Ritchie, C still stands out as a mainstay of computer programming. C has been used in thousands of applications such as in the Linux Kernel to embedded systems like motor vehicles [2].

- **C++:** This programming language introduces object-oriented programming to the powerful C. Bjarne Stroustrup released C++ in 1985 as an expansion to the capabilities of C [2].

- **Python:** Python was released in 1991 by Guido van Rossum. This language is widely used for its fast data processing power. Python also has simple GUI libraries [3].

3.1.3 Analysis

In this section, we will discuss the advantages and disadvantages of each of the chosen languages above.

C

- **OS Independent:** C is a compiled language, meaning its executable will run on almost any operating system [4]. For our purposes, C's compilation process can be targeted for both Linux and Windows operating systems. Using preprocessor directives such as *#ifdef* and *#ifndef* will ensure the most functionality is available for both operating systems.
    - Score: 5 out of 5 points

- **Speed:** C's low-level memory management creates an ideal speed situation. Given the vast amount of data that the controller generates, the program should be able to content efficiently with it.
    - Score: 5 out of 5 points

- **Modularity:** C is not a very modular language and loses points in this section because it would be cumbersome to add new functionalities to the code. While structs do exist in C, they do not exhibit full object-oriented features.
    - Score: 3 out of 5 points

- **Comfortability:** All team members have extensive experience with C. They are all familiar with the syntax and structure and have multiple years of experience working directly with C.
    - Score: 5 out of 5 points

Overall, C is a great language for this project. However, the lack of object oriented features means it ranks lower than the rest of the languages on this list. All team members are familiar with C and its syntax as the team have spent at least two and a half years learning and working with this language.

C++

- **OS Independent:** Like C, C++ is a compiled language. This means that the compilation process can be directed for each target operating system [4].
    - Score: 5 out of 5 points

- **Speed:** Being based off of C, C++ enjoys the benefits of being quick to run and compile. The differences between the two languages are so close they are negligible. The ratings for both languages in the fast category are the same.
    - 4 out of 5 points

- **Modularity:** Modularity is the key difference between C and C++. While C has limited abstraction capabilities, C++ has the ability to employ classes and structs. This makes the modularity of C++ a clear advantage when compared to C.
    - Score: 5 out of 5 points

- **Comfortability:** C and C++ syntax are basically the same. The basic functionalities are defined using the same structure such as loops and functions. The only difference is when using classes and object-oriented features. However, the team had taken multiple classes where object-oriented design was emphasized so this difference is not an issue.
    - Score: 5 out of 5 points

C++ is a great language. It expands the best aspects of C and turns them into something more efficient and easier to read. The modularity is a huge part of the project as the team will turn the software over to inhouse developers at Northrop Grumman. The team is comfortable with both C syntax and object oriented features.

Python

- **OS Independence:** While Python doesn't have the same advantages of C and C++ compilation process, Python is an interpreted language. This means that it can be run on different operating systems as long as there is an interpreter present [5]. This is a downside as we do not yet know the permissions and allowances of the targeted computers and IT systems.
    - Score: 4 out of 5 points

- **Speed:** Python is widely used in the context of big data and data science. Python's speed is a big plus here. However, when compared to C/C++'s speed in context the amount of data this project will be handling, Python's speed loses out.
    - Score: 4 out of 5 points

- **Modularity:** Python is an object-oriented programming language. This means classes and abstractions are available for this project.
    - Score: 5 out of 5 points

- **Comfortability:** This section is where Python's rating goes down. All team members have some experience with Python, however, it was the first language that we learned in an academic setting. The content in that class was the bare basics to allow us to learn the fundamentals of programming.
    - Score: 3 out of 5 points

Python is a great programming language for big data computations. However, the comfortability and speed when compared to C/C++ reduces its score.

3.1.4 Chosen Approach

The programming language that Team Controller has chosen is C++. This selection is shown in **Table 3.1.** Overall, the team is most familiar with the syntax and structure of C but we needed to include object-oriented programming features such as classes, abstraction, inheritance etc. This aspect of C++ would make the overall project more readable and easier to understand.

	OS Independence	Speed	Modularity	Comfortability	Average score
C	5	5	3	5	4.5
C++	5	4	5	5	4.75
Python	4	4	5	3	4

Table 3.1

C scores five points in the operating system independence category because of portability with its compilation processes. The speed category is also scored at five due to its low level memory management, however, C's modularity is where the language lags behind its competitors as the language does not offer object oriented features. The team is very well versed in C so comfortability is scored at a five.

As mentioned above, C++ is the clear winner of this comparison. The deductions taken when looking at C are not taken here due to C++'s object oriented nature. C++ scored a 4 in the speed section to account for the small speed difference between C and C++.

Python scored four points in the operating system independence category due to the fact that an interpreter would be required to run the code. Python's score for speed is due in part to the language's lack of low level memory management. Python's comfortability was a huge deciding factor with the team, which is why it scored a three in that category.

3.1.5 Feasibility

As mentioned previously, all team members have years of experience with C++. To prove that this language is correct for this project, the team will create a demo project to show our clients. This will serve as a guiding system to check in with our clients and make sure we are on the correct path.

3.2 Graphical User Interface

The GUI will be the front-end to our project. It is important to choose the correct library to use. The library will also need to be understandable and have all the required features to meet the clients needs.

3.2.1 Desired Characteristics

For the GUI it will need the following characteristics:

- **Simplicity:** The graphical user interface library will need to be easy to understand and quick to learn.

- **Compatibility:** The GUI library must be able to work with both windows 10/11 and it needs to work with C++. A stretch goal is for the software to run on linux and having a library work on both linux and windows is ideal.

- **Aesthetics:** The GUI should be easy to create, look nice, and professional. The GUI library should provide a framework to make the GUI look aesthetically pleasing.

3.2.2 Alternatives

- **Windows Form:** A windows based IDE that includes many different language support. Windows Form has a feature that allows developers to create a windows GUI utilizing a C++ based front-end [16].

- **QT:** A cross platform IDE that includes multiple languages to help developers create simple and easy desktop softwares, mobile, and embedded platforms [17]. The front-end

library that QT provides makes front-end development easy to learn and has proven to help developers complete tasks faster [17].

- **wxWidgets:** A cross platform C++ library that helps developers create front-end applications with C++ [15]. With this library once the developer creates the front-end it will compile and work on all platforms.

3.2.3 Analysis

This section will discuss the advantages and disadvantages of each of the GUI library/tool stated above in **section 3.4.2**. Each GUI tool will be given a score on each of the characteristics and an overall score will be calculated.

Windows Form

- **Simplicity:** Windows Form utilizes the window form application to create a simple and easy front-end. With Windows Form the developer is able to visually drag and drop buttons and many more features onto a blank windows form.
    - Score: 4 out of 5 points

- **Compatibility:** Since Windows Form utilizes the window form application it uses a windows based library for the front-end. This means it will work just fine for windows however for the stretch goal it will not be compatible with linux.
    - Score: 3 out of 5 points

- **Aesthetics:** Since Windows Form features a drag and drop style front-end development it makes it easier for the developer to create a nicer/professional looking graphical user interface.
    - Score: 5 out of 5 points

Overall Windows Form provides a great framework for creating a GUI. It is simple and easy to use. However the only downside is that it's not compatible with linux so achieving that stretch goal wouldn't be possible.

Qt

- **Simplicity:** Since QT is the newest out of the options here its IDE is much more user friendly and easy to create a GUI. Just like Windows Form, QT features a drag and drop style front-end tool and this makes it very easy to learn.
    - Score: 5 out of 5 points

- **Compatibility:** Unlike Windows Form, QT is cross platform meaning it will work on all platforms. This means that it will accomplish our goal to work on both versions of windows and our stretch goal of linux.
    - Score: 5 out of 5 points

- **Aesthetics:** Just like Windows Form, QT features drag and drop making it simple and easy to create a nice/professional looking GUI.
    - Score: 5 out of 5 points

Overall QT provides a great framework for creating a GUI. It is very simple to use and provides some newer features. It is also cross platform so it makes accomplishing both our goals possible and QT makes it easy to create an aesthetically pleasing GUI.

wxWidgets

- **Simplicity:** Since wxWidgets is just a library and doesn't have an IDE it is harder to learn and design. For example the developer has to code all the GUI. This makes it difficult for the developer to visualize the front-end without continuous testing. Also it will take more time to develop the front-end.
    - Score: 2 out of 5 points

- **Compatibility:** wxWidgets is also cross platform just like QT and this means that it will accomplish both our goals.
    - Score: 5 out of 5 points

- **Aesthetics:** wxWidgets doesn't provide drag and drop like QT or Windows Form so it will take more time to create a nicer/professional looking graphical user interface.
    - Score: 2 out of 5 points

Overall wxWidgets is a great library for developing graphical user interfaces. However since it doesn't provide drag and drop functionality like Windows Form and QT it will take the team more time to learn and develop the front-end. The only benefit to using wxWidgets is its cross platform.

3.2.4 Chosen Approach

The chosen GUI that Team Controller has chosen is QT. It is simple, easy to learn, and provides all the needed functionality for the graphical user interface. The QT IDE is a great development tool and the drag and drop functionality makes it much nicer to visualize the development process of the GUI.

	Simplicity	Compatibility	Aesthetics	Average Score
Windows Form	4	3	5	4
QT	5	5	5	5
wxWidgets	2	5	2	3

Table 3.2

3.2.5 Feasibility

The team will learn and utilize QT's GUI design tool. As a team we will all take time to learn the QT IDE and all its libraries and create a demo GUI for our clients.

3.3 Serial Communication

Serial communication is one of the most common forms of reading and writing data between two devices. Our product must be able to conduct serial communication via a hardwired connection. The protocols we will be using are ambiguous between rs485, rs422, and rs232. A method to identify which protocol the specific hardware is using is essential to ensuring the end user is getting the data as expected. We will also need a method to actually read (rx) the information being transmitted in the physical cable. No writing (tx) to the controller will be necessary as we are just collecting diagnostic data.

3.3.1 Desired Characteristics

For this challenge, our team needs to research libraries or methods that provide a means of conducting communication with the controller. There are hundreds of different serial libraries, but something lightweight may be best since we only need to identify the protocol and read data. The following are characteristics on our wishlist for technologies/libraries to use in our code:

- **OS-agnostic:** Ensuring that our program can run on multiple operating systems without major modifications requires cross-platform libraries. Some libraries may only work for one operating system. Mainly, it must be compatible with both Windows 10 and 11. However, since our stretch goal is to have it ported to Linux, this should be a major consideration. The lowest possible score of 0 means it is not compatible with any of the operating systems listed above. The highest possible score of 5 means it is compatible with almost any operating system.

- **Multi-Protocol Compatibility:** Since military armament systems may use many different communication protocols, the library should have support for various serial interfaces. RS-422 is most commonly used at Northrop Grumman, but it should also have support for RS-485, RS-232, and more. The lowest possible score of 0 means it is not compatible with any of the serial protocols listed above. The highest possible score of 5 means it is compatible with all three ports listed above, and more.

- **Licensing:** Not all libraries will be open source or free for commercial use. For the purpose of this assignment, we need something that is free of charge but still offers comprehensive documentation and usage guidelines. This will ensure that the developers for Team Controller can easily integrate the library into the diagnostic tool. The lowest possible score of 0 means it is not free, and provides very little support. The highest possible score of 5 means that it has extensive documentation as well as an open source license.

- **Lightweight:** The overall functionality of the library should be lightweight since we do not need much utility out of it. The library will mainly be used for (1) connecting to the ambiguous serial port on the hardware running the product, and then (2) reading the data being transmitted. The lowest possible score of 0 means it does not accomplish either of those tasks while substantial in size. The highest possible score of 5 means that it does everything we need without being substantial in size.

3.3.2 Alternatives

- **Qt Serial Port:** This library is part of the Qt framework that was mentioned in section 3.2.2. The most important feature is that it is widely known for its cross-platform capabilities and ability to integrate with the GUI framework. It also abstracts the serial protocols to be able to work with all serial devices [6].

- **Boost.Asio:** This library is popular in C++ for its ability to process several methods of I/O communication asynchronously. In terms of serial communication, it does provide cross-platform support for various operating systems and serial interfaces.

- **Serial:** This library has a much smaller community developed by William Woodall and John Harrison. It has cross-platform capabilities while providing the most basic functionality (open, close, read, write) [7]. It is lightweight and has no dependencies.

- **Generic iostream:** This would involve leveraging C++'s standard I/O streams for serial input with the controller. This wouldn't rely on any third party libraries and is the most minimalistic approach to perform very basic serial communication.

3.3.3 Analysis

Each of these alternatives have their own pros and cons and are all viable options for us to use in our product. Each approach will be analyzed and given a score based on the four desired characteristics from Section 3.3.1.

Qt Serial Port

- **OS-agnostic:** This entire framework is known for its cross-platform capabilities for many versions of Windows and Linux. This is exactly what we are looking for in terms of portability.
    - Score: 5 out of 5 points

- **Multi-Protocol Compatibility:** This library provides support for various serial protocols, most notably RS-422, RS-485, and RS-232 [6]. It is designed to make it easier for the developer to work with serial devices regardless of the specific protocol being used by methods of abstraction.
    - Score: 5 out of 5 points

- **Licensing:** Qt's framework is freely available for use/change/distribution under the GPLv3 License [8]. This means we can use the library to its full extent with much more flexibility and with a larger community behind it.
    - Score: 5 out of 5 points

- **Lightweight:** The entire Qt framework is quite comprehensive, but Qt Serial Port is a portable library which makes it reasonably lightweight. Integrating with the Qt GUI framework may be a bit more intensive but not very complex.
    - Score: 3 out of 5 points

Boost.Asio

- **OS-agnostic:** The library was designed for cross-platform use in most situations. In serial communication, it abstracts (most) platform-specific details and should suit our needs for the purposes of this project.
    - Score: 4 out of 5 points

- **Multi-Protocol Compatibility:** Although this library is definitely capable of handling multiple serial protocols, it may require more manual configuration and be less user friendly for our developers [9].
    - Score: 4 out of 5 points

- **Licensing:** The Boost framework is freely available for use/change/distribution under their Boost Software License [9]. This means we can use the library to its full extent with much more flexibility.
    - Score: 5 out of 5 points

- **Lightweight:** This library was designed to be efficient and lightweight. Because of this, it may be more complex to set up and integrate with less features.
    - Score: 4 out of 5 points

Serial

- **OS-agnostic:** The library was designed for cross-platform use specifically in Linux and most versions of Windows.
    - Score: 5 out of 5 points

- **Multi-Protocol Compatibility:** Serial was designed to be used for very basic serial communication, particularly RS-232 [7]. Like Boost.Asio, it does have support for other serial protocols but may require more manual configuration.
    - Score: 3 out of 5 points

- **Licensing:** The software is freely available for use/change/distribution under the MIT License. This means we can use the library to its full extent with much more flexibility.
    - Score: 5 out of 5 points

- **Lightweight:** This library was designed to be super efficient/lightweight, and also very easy to use for the simplest of tasks. However, it is a port from another Python library and lacks some useful documentation.
    - Score: 4 out of 5 points

Generic C++ iostream

- **OS-agnostic:** Although C++ in nature is quite cross-compatible, without a library it lacks basic serial communication support that may require manual configuration depending on the OS.
    - Score: 3 out of 5 points

- **Multi-Protocol Compatibility:** In terms of serial protocols, technically this is the most compatible option but again may require much more manual configuration depending on the specific port being used.
    - Score: 3 out of 5 points

- **Licensing:** iostream is a built-in library that's standard with C++. It is freely available without any extra dependencies or requirements.
    - Score: 5 out of 5 points

- **Lightweight:** This is the most minimalistic approach for serial communication, only requiring the built-in libraries. However, creating our own code for open/close/read methods may end up being quite complex.
    - Score: 4 out of 5 points

3.3.4 Chosen Approach

The alternative that Team Controller has chosen is the library Qt Serial Port. The corresponding points for each approach is reflected in Table 3.3. Overall, using Qt's framework will prove to be useful not only for serial communication all around but also for interfacing with the GUI. Although it loses some points for its size, most of the libraries we will need are portable and are lightweight themselves. Other than that, it is perfectly compatible with all operating systems we're concerned with as well as all serial protocols we could possibly need. It is also completely free for use with extensive documentation and usage guidelines.

	OS-Agnostic	Protocol Compatibility	Licensing	Lightweight	Avg. Score
Qt Serial Port	5	5	5	3	4.5
Boost.Asio	4	4	5	4	4.25
Serial	5	3	5	4	4.25
iostream	3	3	5	4	3.75

Table 3.3

3.3.5 Feasibility

Using Qt Serial Port, we will be able to implement serial communication into our simple user interface effortlessly. Demos of very simple open, close, and read methods of abstracted data will be shown in our tech demo and MVPs. From there Qt gives the option to scale our system to fit future goals and requirements

3.4 System Installer

The system installer should be a file which will be downloaded and run by the user. The installer is responsible for collecting all the user preferences we need in order to run the application. Then the installer will set up our application's environment including creating directory paths for our system to work in and adding our applications files to their designated directories. The installer is also responsible for deleting the application if the user decides to do so.

3.4.1 Desired Characteristics

When deciding on a technology to use for our installer. We will prioritize the following characteristics (listed most important to least important):

Free to Use

For the purposes of this project we do not have access to any paid services. Therefore the installer technology we choose MUST be free.

Simple

Since ownership of this product will eventually be handed off to Northrop Grumman. We should be thinking ahead to try to reduce the complexity involved with modifying or updating the system. Additionally choosing a simple technology will help the team in the short term by saving time in the development phase for the installer. Choosing a simple installer will also help keep the user from getting confused or overwhelmed when setting up the system.

High Speed

Ideally our application should be able to install in under a minute, though the shortest amount of time we can achieve is preferable.

The installer technology we choose should at a minimum be free, then the other characteristics will be considered and compared between the various free technologies. Unfortunately a separate installer will have to be developed if we want to meet our stretch goal of porting the project to linux. As it is not possible to create a cross compatible installer as of right now.

3.4.2 Alternatives

Options for our Windows Installer:

- **Inno Setup** is a compiler for .exe installer files. It was developed by a well known software developer Jordan Russel, who is responsible for developing many windows utilities. Its main use case is for distributing applications that are meant to run in windows environments. Inno Setup has a convenient GUI for naming your program, providing the version, the directory structure, etc. Then once all the data is input you can compile your installer and you're ready to distribute the application. Additional features include built in uninstaller, multiple language support and file compression and encryption.

- **WiX Toolset** is an open source project which has had a wide variety of developers over the years. It is a direct alternative to Inno Setup and it has the same use case, except it can support the creation of .msi files. WiX is unique because it is a VS Code extension meaning we could potentially gain easy access to the installer portion of the project through our existing work environment. Another advantage to WiX is the fact that it has a large support infrastructure which we can use to debug and study the technology. It also provides the basics of an installer including custom environment setup version control features and uninstaller.

- **NSIS** is also an open source project, it was launched in 2000 and has a very large user base. It also makes msi installer files for windows operating systems. This framework is extremely similar to the 2 previously mentioned. It provides mostly the same features with a slightly different user interface.

Options for our Linux Installer:

- **Debian** is an open source software package installer developed by the linux community. A .deb file contains the software for your project, the necessary directories, and the

metadata (things like version, name, authors, etc.). The main distinction between .deb installers and other linux installers is that this file type specializes in Debian-based distributions of linux. Since Linux is open source there are many different variations of the operating system, Debian is a family of similar distributions.

- **RPM** is essentially the same thing as DEB except it is for Red Hat-based linux distributions. The same logic applies, Red Hat refers to a family of similar linux distributions. This is the family most related to the standard linux operating system which makes it appealing.

- A **tar.gz** compressed application is a simple file type. To create one just navigate to your project's main directory and execute a tar command to compress and zip all sub files and directories into a single tar.gz file. This file can then be downloaded and unzipped on a different machine. The benefit of this is that it should work on most if not all distributions of linux and it is easy to create. The negatives are that this file does not include much of the metadata that other installer files include, it is also harder to set up on the host system because it requires the user to enter a specific command to the terminal to extract the program.

3.4.3 Analysis

To compare these technologies and choose the one that best suits our needs, more specific research will be needed. Since installers require an existing project in order to be tested, we will need to use a sample C++ program during the testing phase.

Inno Setup

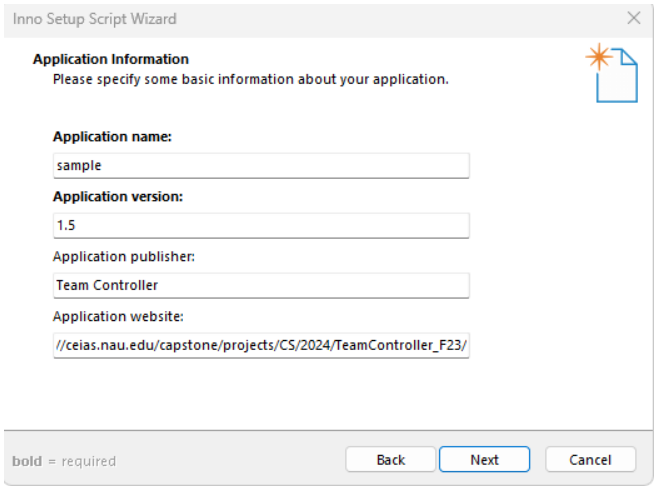

Figure 3.4.1

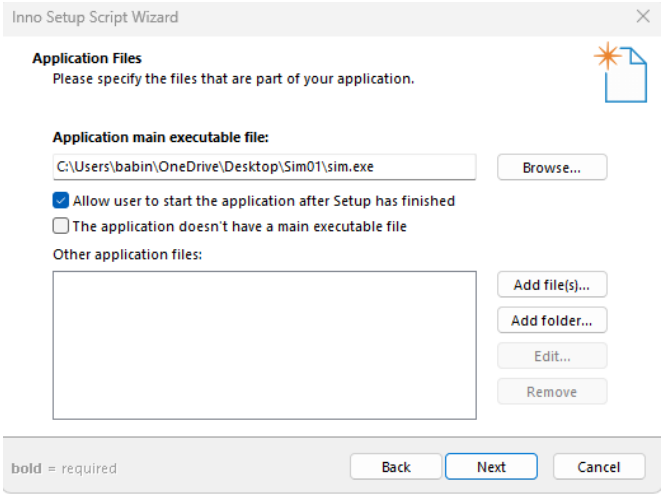

Table 3.4.2

Included above are some screenshots of the Inno Setup GUI. Everything was straight forward, it asked for details, files, settings related to our project then it turned all of this data into a script which is then compiled to generate our installer file. The file was tested, it asked for an installation path and set everything up nicely. The uninstaller was also tested and worked as expected. Overall this was a straightforward framework and we could definitely use this in our project.

WiX Toolset

This product was underwhelming. There was no user friendly GUI and in order to build the installer file you must manually create XML files to compile into the installer. Since this process is rather complicated, we decided to cut this test short as it has already proven to be a worse product for our purposes than Inno Setup was. Overall I cannot see us using this product for our project.

NSIS

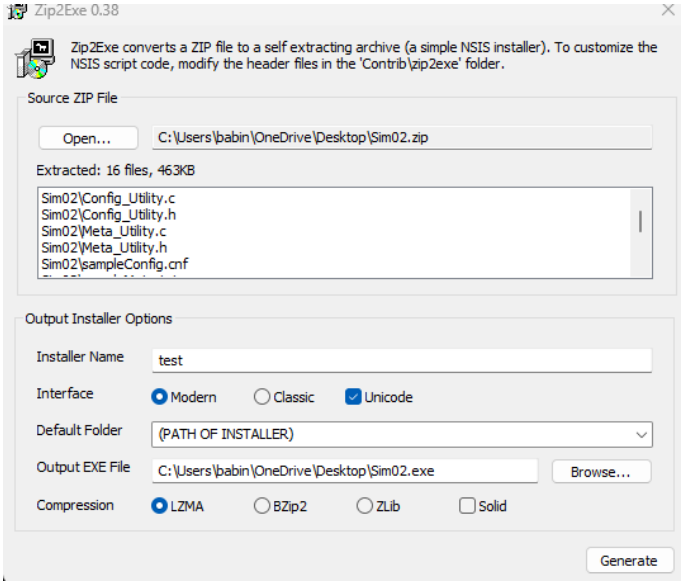

Figure 3.4.3

NSIS provided a nice GUI which asks for a zip file containing project code. A few other options are able to be modified but none of them are really too critical. After generating the installer it ran just as expected. This product is definitely an option for our team.

3.4.4 Chosen Approach

After considering these options we can immediately narrow down our choices for the windows installer to NSIS and Inno Setup. Both options produce .exe installers. The GUIs for each are different, there are less steps for the NSIS installer which could be a positive but in this case that is a drawback since that means less customization. With that being said, Team Controller believes that Inno Setup will be the best bet for this project considering the fact that it meets our needs while remaining incredibly simple to understand and use.

	Speed	Simplicity	Comfortability	Average score
Inno Setup	4	5	5	4.67
WiX	5	3	2	3.33
NSIS	5	4	4	4.3

Table 3.4

For our Linux installer we are quite limited with our options. We will have to go with RPM due to the fact that it covers the family of linux distributions that we have to accommodate in order to meet our stretch goal. In addition to implementing our linux installer using RPM we should also provide a tar.gz file for installing our application on non-red hat distributions of Linux as the cost of creating a tar.gz file is negligible.

3.4.5 Proving Feasibility

To show that this technology will work for our project, we need to show that it is capable of installing a program that contains all the functionality needed to complete this project. Both the windows installer and the linux installer must be capable of handling programs with GUI and serial communication modules.

4.0 Integration

The software application isn't too complicated due to the fact it is all done locally so there isn't any need for a server or database. As seen in **Figure 4.0**, the weapon controller will plug into a laptop and the product will gather all the data utilizing QT's serial protocol library. The program must also be able to process the three different types of serial protocol that Northrop Grumman requires. Once the data is processed by the software's back-end it then will output the data onto a log file for later use. Both our back-end and front-end will utilize C++ since all the libraries the team is going to use are in C++. Then all the data will be displayed on the graphical user interface and allow for the user to interact with it. Then the user has an option to download the log file if needed. Finally the program will be using the Inno Setup to create an .exe file for windows and .rpm for linux.

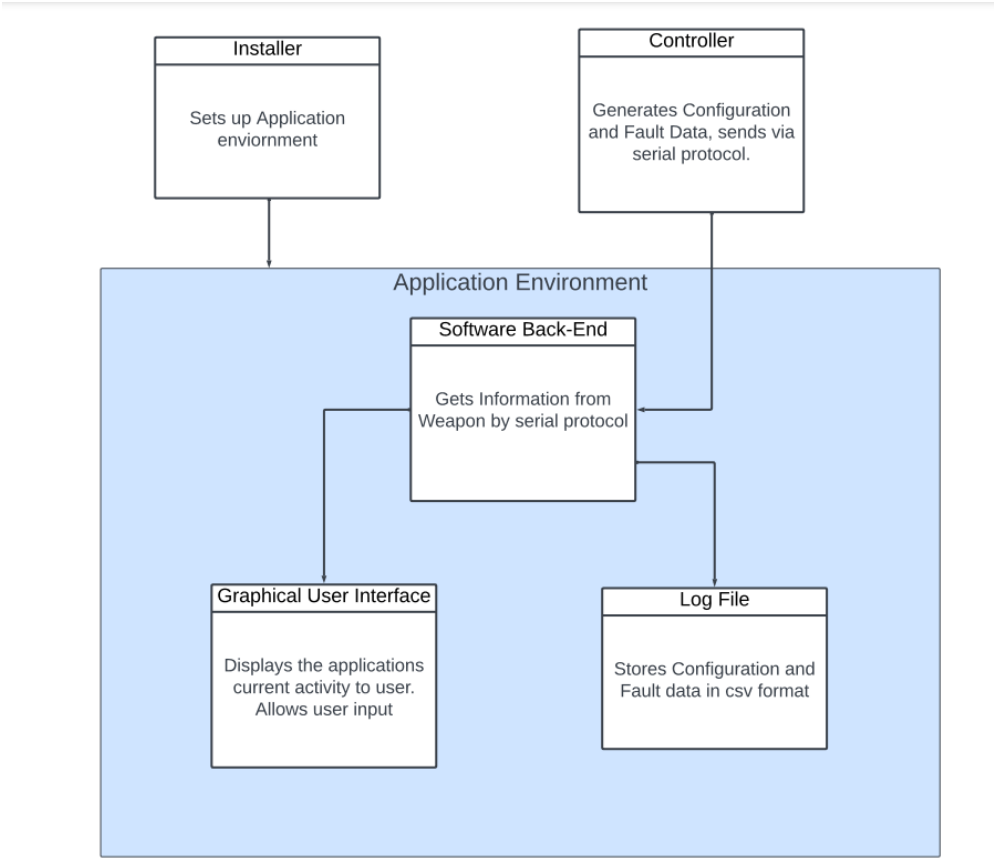

Figure 4.0 System Diagram

5.0 Conclusion

Diagnostic systems are a necessary part of maintenance of any mechanical system. With the implementation of this project, multi-billion dollar military equipment can be serviced effectively with a lower cost to Northrop Grumman.

In summary, Team Controller will use C++ as the base programming language, the Qt Serial Port library to assist in reading data from the weapon's controller, and the Qt GUI library to create a simple and modern graphical user interface. The team will also utilize the Inno Setup installer software to help create installers for windows. For the team's stretch goal to port to Linux, the team will use RPM package manager. We have also ensured that all our technologies have cross-platform capabilities to ensure that porting in the future is as easy and effective as possible.

The next steps for the team will be to create and refine small demonstrations to prove our technologies' feasibility. Afterwards, the team will begin building the requirements document by connecting with our clients and listening carefully to their requests for what this software should do.

6.0 References

[1] "James Webb Space Telescope, Built in Partnership with Northrop Grumman, Reveals New View of the Universe," *Northrop Grumman Newsroom*. https://news.northropgrumman.com/news/releases/james-webb-space-telescope-built-in-partnership-with-northrop-grumman-reveals-new-view-of-the-universe

[2] "C vs C++ – What's The Difference?," *freeCodeCamp.org*, Nov. 04, 2021. https://www.freecodecamp.org/news/c-vs-cpp-whats-the-difference/

[3] K. Ostrowska, "A Brief History of Python," *LearnPython.com*, Jun. 20, 2022. https://learnpython.com/blog/history-of-python/

[4] "Is C Platform Independent? Full Explanation | Developer Pitstop," *developerpitstop.com*, Apr. 18, 2023. https://developerpitstop.com/is-c-platform-independent/

[5] T. Statler, "Is Python Platform Independent?," *Comp Sci Central*, Feb. 28, 2021. https://compscicentral.com/is-python-platform-independent/

[6] "QT Serial port," Qt Documentation, https://doc.qt.io/qt-6/qtserialport-index.html (accessed Oct. 20, 2023).

[7] W. Woodall and J. Harrison, "Cross-platform, Serial Port Library," GitHub, https://github.com/wjwwood/serial (accessed Oct. 20, 2023).

[8] "Qt licensing," Choose the Right License for Your Development Needs, https://www.qt.io/licensing (accessed Oct. 20, 2023).

[9] C. M. Kohlhoff, "Serial Ports," Boost C++ Libraries, https://www.boost.org/doc/libs/1_76_0/doc/html/boost_asio/overview/serial_ports.html (accessed Oct. 20, 2023).

[10] "20 Best Installers for Windows Programs as of 2023." *Slant*, 14 Dec. 2015, www.slant.co/topics/4794/~installers-for-windows-programs/

[11] "Inno Setup," *jrsoftware.org*. https://jrsoftware.org/isinfo.php

[12] "Get started with WiX | WiX Toolset," *wixtoolset.org*. https://wixtoolset.org/docs/intro/ (accessed Oct. 20, 2023).

[13] "NSIS Users Manual," *nsis.sourceforge.io*. https://nsis.sourceforge.io/Docs/

[14] "How to install software in Linux (RPM/DEB systems) :: TutsWiki Beta," *tutswiki.com*. https://tutswiki.com/install-software-linux-yum-rpm-apt-dpkg/ (accessed Oct. 20, 2023).

[15] “Cross-Platform GUI Programming with wxWidgets - wxWidgets,” *www.wxwidgets.org*. https://www.wxwidgets.org/docs/book/

[16] “How to Create a C++ GUI Application Using Visual Studio? | Simplilearn,” *Simplilearn.com*. https://simplilearn.com/tutorials/cpp-tutorial/cpp-gui#conclusion (accessed Oct. 20, 2023).

[17] “Qt Designer Manual,” *doc.qt.io*. https://doc.qt.io/qt-6/qtdesigner-manual.html