Team Controller
Northern Arizona University
Flagstaff, Arizona
May 9th, 2024

Zachary Parham (Team Lead): zjp29@nau.edu
Tayyaba Shaheen (Mentor): ts2434@nau.edu
Bradley Essegian: bbe24@nau.edu
Brandon Udall: bcu8@nau.edu
Dylan Motz: djm658@nau.edu

Final Report

Northrop Grumman

Weapon System Support Software

Harlan Mitchell

Laurel Enstrom

1.0 - Table of Contents

2.0 - Introduction

2.1 - Our Clients and the Problem

Our clients work for Northrop Grumman, one of the largest aerospace and defense contractors in the world. With an annual revenue of $30 billion, Northrop Grumman has led the development of many revolutionary projects for our United States military. Part of working with these advanced weapon systems is the ability to diagnose problems that occur with them.

These systems produce a massive amount of complex data that is not easy to work with. When one of Northrop Grumman's customers runs into some type of problem or issue, they will dispatch an engineer with a tool to collect this data. From there they will work to resolve the issue; traveling back and forth as many times as necessary.

The apparent problem with this is that the end user lacks the ability to properly diagnose problems with their products independently. The existing tool that Northrop Grumman has is intended for engineers only and contains complex or insignificant data that is not important to the end user. Furthermore it is very expensive to dispatch engineers as it costs time, resources and most importantly money to do so.

2.2 - The solution

Our proposed solution is a program running a graphical user interface (GUI) which can read in data directly from Northrop Grumman's weapon controllers and display the information in real time. Our solution relies on RS422 serial protocol to communicate with the weapon controller where it will extract data and render it into one of our 4 key pages based on the category of information it falls under. Our four pages along with their purposes are listed below.

- **Events page:** Takes the form of a text log and displays events and errors which represent notable points in time in which something happened, examples of this would be a firing event or a weapon jam error.
- **Status page:** Represents the state of the weapon which can be measured at any point in time. Some examples include trigger status (i.e. are the trigger(s) of the weapon engaged, indicating an attempt at firing, or disengaged) firing mode (i.e. the weapons current mode of operation, is it on safe, semi-automatic, fully-automatic or burst).
- **Electrical page:** Lists the various subcomponents of the weapon and their measured amps and voltages.
- **Connection page:** Gives the user the option to select and configure their serial port in order to be able to effectively communicate with the weapon controller. Among the

adjustable settings are baud rate which represents the rate of data transmission over the serial port and flow control which defines the behavior the communicating serial ports will use in order to decide whose turn it is to transmit data at any given point in time.

We have also provided 2 additional pages other than the 4 essential pages mentioned previously. The first is a notifications system which displays notifications involving the programs back end processes. One such example would be if an unrecognized or improperly formatted message is received by our application, it will render a notification for the user. Another example would be session statistics which are calculated and displayed in the notification after a disconnect occurs with the weapon controller. Clicking on the notifications page will display all of the notifications generated that session using red to denote errors and green to denote routine operations.

Finally we have added a user settings page where the user can adjust the behavior of various parts of our application for example toggling color coding in the events page or modifying the timeout value which represents the number of seconds our program will wait before disconnecting if the controller is not responding. All of these settings as well as the connection settings are saved and loaded cross session meaning if you launch the application the next day it will remember your preferred settings from the previous session.

Our application is designed to be simple so it can be used by both engineers and customers of Northrop Grumman. It is designed to be easily installable on devices running windows or linux so that it can be obtained, set up and used in a matter of minutes. Furthermore it is designed to be modular and easily modifiable so that Northrop Grumman can further adjust the application based on their specific needs at any point in the future.

By implementing these solutions Northrop Grumman can more effectively diagnose problems with their weapon systems and more importantly, they can give their customers the tools they need to also stay informed on the exact condition of their products.

3.0 - Process Overview

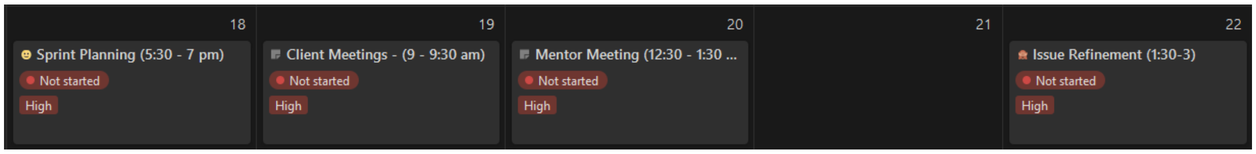
During the development of the Weapon System Support Software, the team had developed a process for organizing the work that needs to be done. More specifically, the team had used an Agile approach, meaning that work was done in sprints. These sprints spanned over two weeks and held four different meetings, which will be discussed later in this section.

3.1 - Meetings

During the course of the second semester of the capstone project where implementation started, the team used four different meetings to organize the development. These meetings were:

- **Sprint planning** - These meetings occurred every week on Mondays. Sprint planning meetings were used to add issues to the current sprint or plan the upcoming sprint;
- **Client Meetings** - Client meetings were every week on Tuesday mornings. These meetings usually consisted of the team demonstrating the most recent updates to the application or asking questions on development. Having these meetings occurring twice per sprint proved invaluable to our development process;
- **Mentor Meetings** - Meetings with our mentor were used to guide the completion of in class documentation and deliverables and occurred every Wednesday. Combined with that, the team also updated the mentor with major improvements in the application as well as any questions we had.
- **Issue Refinement** - Issue refinement meetings occurred every other Friday. These meetings were used to narrow down the scope of issues, break them into smaller issues or include specific goals the issue would meet once resolved.

These meetings served as an organizational tool that would prepare the team for the current or upcoming sprint. It was very important to limit the number of meetings to the absolute minimum to not waste any time.


18	19	20	21	22
Sprint Planning (5:30 - 7 pm)	Client Meetings - (9 - 9:30 am)	Mentor Meeting (12:30 - 1:30 ...		Issue Refinement (1:30-3)
Not started	Not started	Not started		Not started
High	High	High		High

Screenshot 1: Screenshot showing the weekly meetings

3.2 - Development and Version Control

When planning for the development of the application, the team identified that an external tool responsible for tracking issues and development would lead to friction. Because of this, the team opted to use the github issue tracker. Using this issue tracker that was integrated with the version control system proved to be the best solution.

For the version control software, the team chose Git and *GitHub.com* to host the repository. The most appropriate branch structure for Agile development can be seen below. The team started with the ***main*** branch, which contains the finished and approved version of the MVP developed in the previous sprint. Branched off of the main branch is the individual ***sprint_x*** branch, which as the name suggests, holds all current development work performed by the team during the sprint. For individual issues the team branched off of ***sprint_x*** and developed single issues or kept a running branch for the team member's commits.

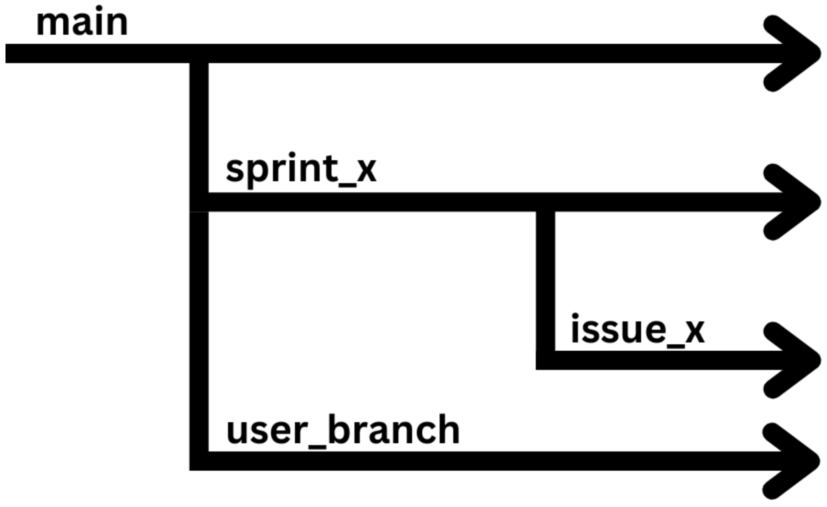


Diagram 1: GitHub branch structure

4.0 - Requirements

4.1 - Requirement Acquisition

In order to define the requirements for this project we met weekly with our clients where we would discuss different aspects of the project and assess our clients needs in relation to the tools we had available to us. We would then propose a list of requirements for approval. These requirements were then reviewed by our clients before feedback was given and recorded by our team. We would then consider this feedback and come to the next meeting with a revised list of requirements and repeat the process as many times as necessary until the clients fully approved of the requirements we came up with. We found that this iterative process was effective and thorough in defining the goals of the project in a manner that everyone approved of and could understand.

4.2 - Requirements Breakdown

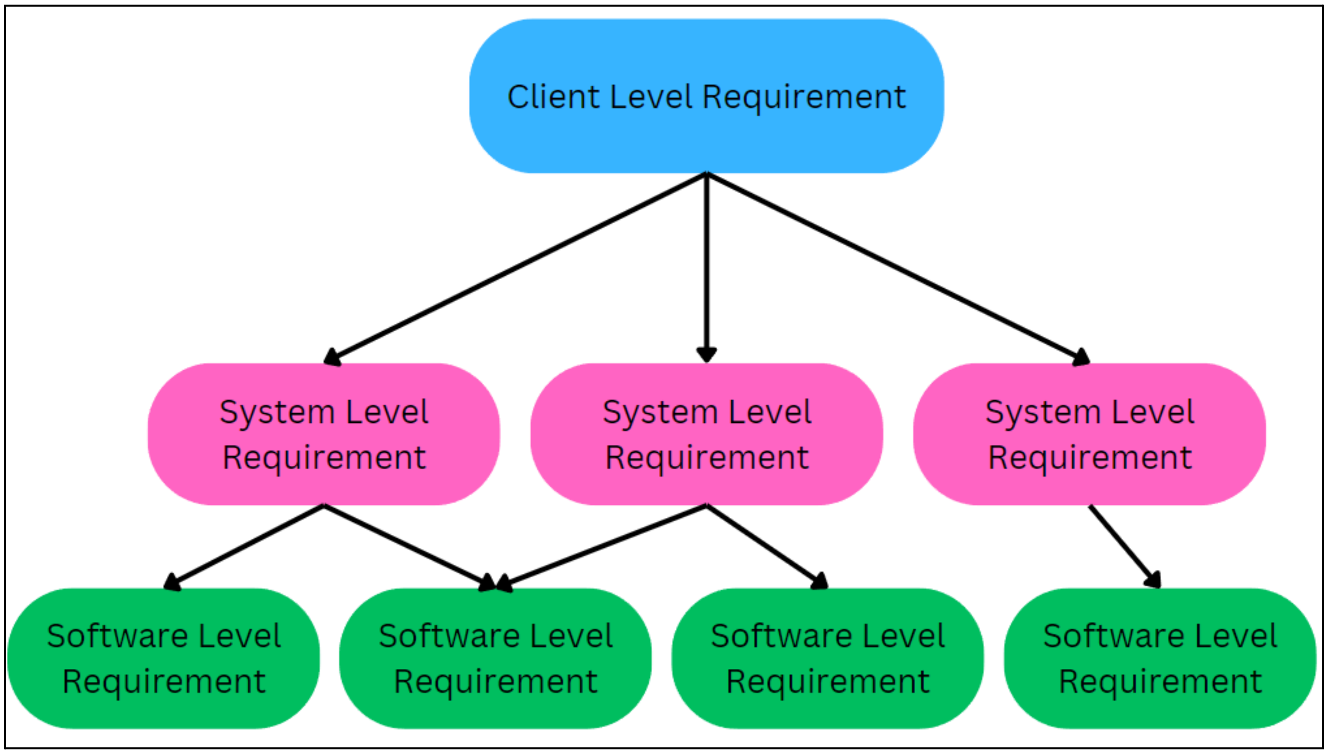

Client Level Requirement
System Level Requirement
System Level Requirement
System Level Requirement
Software Level Requirement
Software Level Requirement
Software Level Requirement
Software Level Requirement

Diagram 2: Requirements Breakdown

For this project we decided to divide our requirements into 3 categories. The first level is the **client level requirements**. These represent things our clients directly requested. For example, *"you must use RS422 Serial Protocol for your communications with the weapon controller."* Is an example of one of our client level requirements.

The next level is the **system level requirements**, these represent verifiable operations involved with implementing the client level requirement. A single client level requirement will have one or more system level requirements that if we can prove are satisfied by our application, we can prove the client level requirement is satisfied.

The final level is the **software level requirements**. These represent modular sub-processes that will be used to implement the system level requirements. A software level requirement can be thought of as a function within a program. A single software level requirement can be associated with 1 or more system level requirements from any client level requirement.

4.3 - Requirements Traceability Matrix

Here is the traceability matrix we designed for our clients which defines all of our requirements from each category. For better readability, the software level requirements are limited to just the name within the matrix as many are used more than once. However they are defined at the bottom of this section. For definitions of specific terms, please refer to the glossary in section 10.0.

Client requirement	System requirements	Software requirements
CR01 The data display module shall be a desktop application.	**R01** The data display module shall be an .exe file.	NA
	R02 The data display module shall display a GUI.	SR15 - SR19
CR02 The data display module shall read input data via RS422 serial protocol from the controller simulator.	**R03** The data display module shall be capable of serializing / deserializing messages received via an RS422 serial port.	SR01 - SR05 SR08
	R04 The controller simulator shall be capable of serializing / deserializing messages received via an RS422 serial port.	SR01 - SR05 SR08
CR03 The data display module shall have the ability to write event data into a log file	**R05** The data display module shall be capable of generating a log file including all known events when requested by the user	SR11 SR15
	R06 The user shall be able to determine if a log file will be	SR11 SR20

	automatically generated after a session.	
	R07 The user shall be able to determine how many auto saved log files will be kept before overwrites occur on the oldest autosaved file.	SR20
CR04 The data display module shall display all weapon status information directly to the application's window for the duration of a session.	**R08** The controller simulator shall send status updates through the designated serial port every 250 milliseconds (2 seconds).	SR01 SR05 SR14
CR05 The controller simulator shall send event updates to the data display module.	**R09** The controller simulator shall send event updates through the serial port at most 100 milliseconds after they are generated. second after they are generated.	SR01 SR05 SR12 SR13
CR06 The data display module shall not require admin rights to install, set up, or use.	**R10** The data display module shall not require admin rights to install setup or use	NA
CR07 The data display module shall include filtering options to filter events and errors	**R11** The data display module shall have the capability to display **only errors** to the Events tab of the GUI	SR13
	R12 The data display module shall have the capability to display **only cleared errors** to the events tab of the GUI.	SR21

	R13 The data display module shall have the capability to display **only active errors** to the events tab of the GUI.	SR21
	R14 The data display module shall have the capability to display only **non-error events** to the events tab of the GUI.	SR21
CR08 The system and its environment shall be installed via an installer file.	**R15** The system and its environment shall be installed via an installer file	NA
CR09 The system shall be portable on Windows 10 or 11	**R16** The system shall be portable on Windows 10 or 11	NA
CR10G The system should be portable on Debian linux distributions	**R17G** The system should be portable on Debian linux distributions	NA

Table 1: Traceability Matrix

4.4 - Software Level Requirements

Serial Communication:

- **SR01 -** The software shall be capable of generating *serialized* versions of given *status data* and *event data.*

- **SR02 -** The software shall be capable of generating *status data* given *serialized* status data.
- **SR03** - The software shall be capable of generating an *event string* given a *serialized event string*.
- **SR04** - The software shall be capable of generating *electrical data* given *serialized electrical data.*
- **SR05** - The software shall be capable of sending *serialized data* through a *serial port*.
- **SR06** - The software shall be configurable to fill one of the the following roles during *handshake protocols*
    a. send the first contact message every 5 seconds until a response is received
    b. listen for the first contact message, then respond.
- **SR07** - *Handshake protocols* shall be implemented using the Boost.Asio serial library
- **SR08** - The software shall be able to listen for and record serialized bit strings from a given *serial port*.
- **SR09** - The software shall be able to pause serial communication
- **SR10** - The software shall be able to resume serial communication
- **SR11** - The software shall be capable of storing all event strings received via serial communication until a new session is started or the program ends.

Controller Simulator:

- **SR12 -** The software shall be capable of generating *event strings* with random *parameters* given a collection of *event messages*.
    - **Note:** See terminology section for definitions of event string, event message and parameters
- **SR13 -** The software shall be capable of reading *event strings* from the *command line interface*.
- **SR14 -** The software shall be capable of generating randomized *status data*.

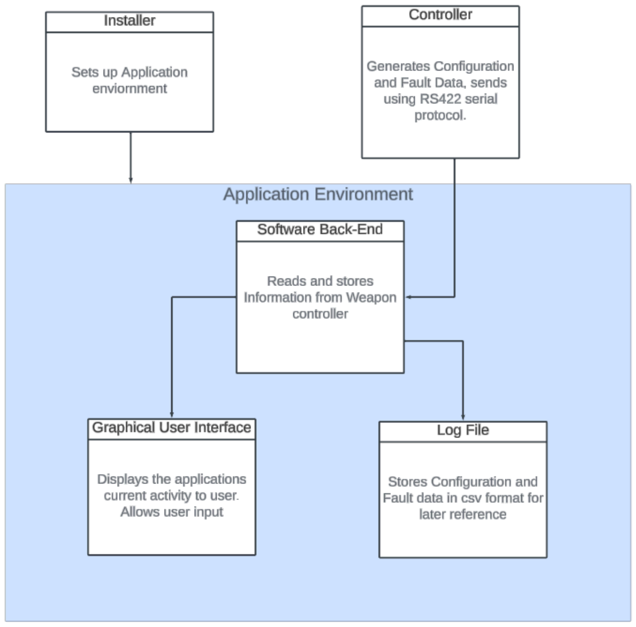
Data Display Module:

- **SR15 -** The software shall be capable of writing an event log file in csv format, given a collection of events upon user request.

- **SR16 -** The software shall be capable of opening the events page when the events button is pushed.
- **SR17 -** The software shall be capable of opening the status page when the status button is pushed.
- **SR18 -** The software shall be capable of opening the electrical page when the electrical button is pushed.
- **SR19 -** The software shall be capable of opening the connection settings page when the connection settings button is pushed.
- **SR20** - The software shall allow the user to input how many auto saved log files they want to be kept before overwrites occur on the oldest autosaved file.
- **SR21** - The software shall allow the user to input what filter they want on the event page out of the following options.
    a. Only errors
    b. Only cleared errors
    c. Only active errors
    d. Non error events

5.0 - Architecture and Implementation

For the program's architecture, we have designed a simple diagram (Figure 1) that breaks down the system into several components as seen below. Similar to a model-view-controller architecture pattern, each component is a separate piece in which the back-end acts as a liaison between them all. The top of the architectural diagram starts with the whole system being packaged and installed via a third party installer, known as Inno Setup. This is used to push the product to production when on a stable release. There is also the Controller component, which is what supplies our program with the information that needs to be interpreted and manipulated. Within the actual meat of the application, there are three main components that all interface with each other: the backend, the GUI, and the log file management. The software backend is essentially the brains of the Data Display Module (DDM), also known as the GUI, and as well as the brains for the log file generation.


Installer
Sets up Application enviornment
Controller
Generates Configuration and Fault Data, sends using RS422 serial protocol.
Application Environment
Software Back-End
Reads and stores Information from Weapon controller
Graphical User Interface
Displays the applications current activity to user. Allows user input
Log File
Stores Configuration and Fault data in csv format for later reference

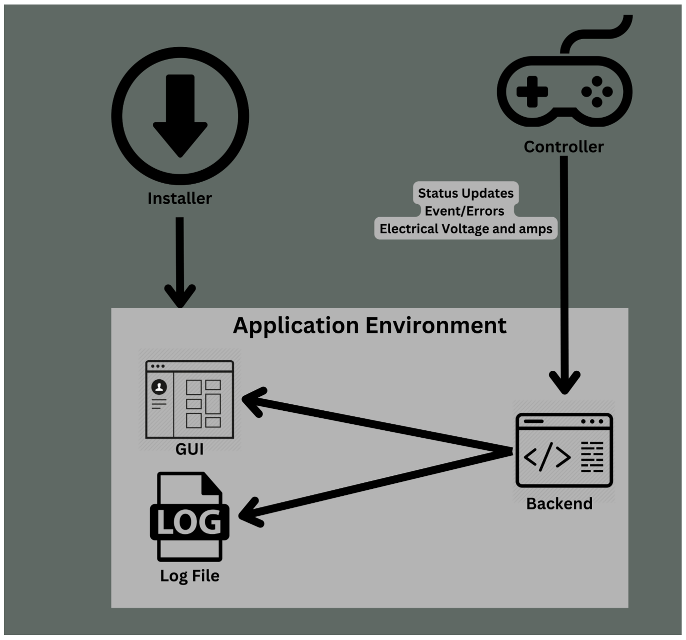
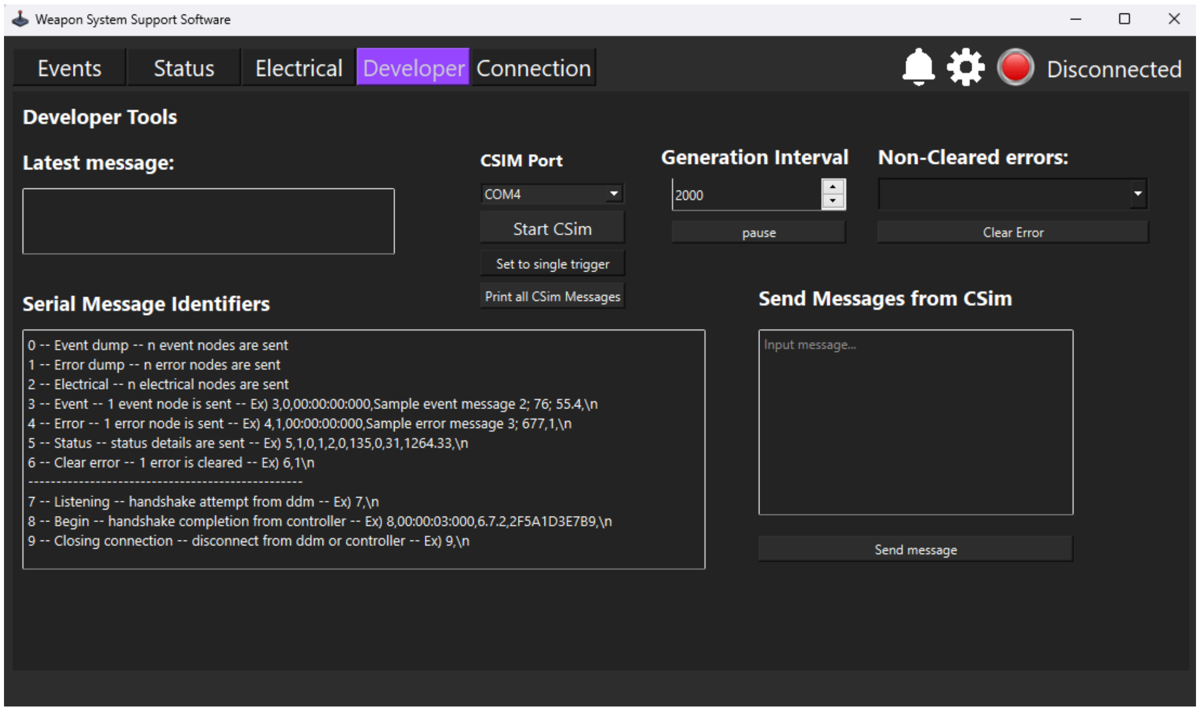


Diagram 3: Architecture Diagram

Diagram 4: Graphical Diagram

5.1 - External Controller Component

The weapon controller is responsible for supplying our program with the data it needs to update its GUI and log files. The controller's purpose is to manage the weapon and record any relevant weapon data found during operation. This data is then sent via RS422 serial communication to our program's back end. Because our team was unable to get our hands on a real controller demo, we have developed a simulator that should act and perform similarly to the real thing. A Northrop Grumman engineer can manage the simulator in the "Developer" tab of our program. This entire page and all of its subcomponents can be completely disabled (without compiling) by modifying the DEV_MODE preprocessor directive in the source code.



```
0 -- Event dump -- n event nodes are sent
1 -- Error dump -- n error nodes are sent
2 -- Electrical -- n electrical nodes are sent
3 -- Event -- 1 event node is sent -- Ex) 3,0,00:00:00:000,Sample event message 2; 76; 55.4,\n
4 -- Error -- 1 error node is sent -- Ex) 4,1,00:00:00:000,Sample error message 3; 677,1,\n
5 -- Status -- status details are sent -- Ex) 5,1,0,1,2,0,135,0,31,1264.33,\n
6 -- Clear error -- 1 error is cleared -- Ex) 6,1\n
-------------------------------------------------
7 -- Listening -- handshake attempt from ddm -- Ex) 7,\n
8 -- Begin -- handshake completion from controller -- Ex) 8,00:00:03:000,6.7.2,2F5A1D3E7B9,\n
9 -- Closing connection -- disconnect from ddm or controller -- Ex) 9,\n
```

Screenshot 2: Developer Page

5.2 - Software Backend

The back-end component's main responsibility is to be able to create a connection with an attached serial device and read messages sent through that port. Furthermore, our software must figure out how to properly interpret those serial messages—which could be separated into errors, events, status updates, or electrical subcomponents. Data is stored in their respective class objects and frequently updated as serial communication occurs.

5.2.1 - Events

The "Events" class in our software is designed to manage and store significant information about events and errors received from the (simulated) controller. It consists of two linked lists structures: an EventNode and ErrorNode. The EventNode structure stores data related to non-error events, such as a unique identifier, a timestamp, the event message, and a pointer to the next node in the list. The ErrorNode structure inherits all this from EventNode, and extends the functionality to handle error events. It includes two additional fields which are the error's current status (cleared or active) and the location of the error in the automatic log file (for use later on). Also, it is a separate data structure so it also has its own pointer to the next node in the error linked list. Besides the typical utility functions you would find for a linked list, it also has utilities for logging event data to files and loading data from log files. All of the information in these linked lists is processed and displayed in the “Events” tab of our GUI.

5.2.2 - Status

As a significant part of our requirements, we must also store general information about the weapon system as these events/errors are occurring. The “Status” class includes member variables that represent different aspects of the system’s status; including whether the system is armed, the status of either of the two triggers, the overall controller state, the firing mode, feed position, burst length, firing rate, controller version, and finally the CRC version. It also stores other relevant counters such as the total amount of errors and events, which is also in the Events class. All of this information is displayed in real time (as we receive the messages) in the “Status” tab of our GUI, with custom graphics for the feed position and firing mode.

5.2.3 - Electrical

Last but not least is our “Electrical” class in our software. It is designed to manage and represent the electrical components’ information within the weapon system. Since there can be any number of components depending on the weapon system, we use another linked list data structure (electricalNode) to dynamically load, manipulate, and output electrical data to our GUI. The structure only has a few member variables: a unique identifier, the component name, voltage, current amps, and a pointer to the next node in the linked list. All of the electrical components are dynamically loaded into the “Electrical” tab of our GUI.

5.3 - Log File

The log file component of our software is a crucial mechanism for logging and storing event/error data that has been generated throughout an entire session. This enables the system to capture and persist critical information, facilitating troubleshooting and analysis. Each entry in the log file corresponds to a specific event or error encountered during a session with the controller. This component of our program is seamlessly integrated into our software architecture, allowing the system to automatically write data to the log file during runtime. The capability to retrieve and read data from a log file for analysis is also an important aspect of our architecture. It allows engineers or maintenance technicians to analyze data in a clean and formatted environment, right within the program.

5.4 - Installer

Finally, the installer component in our software is responsible for facilitating the deployment and installation of the application on any target system. It serves as the gateway to access and utilize the software, with little to no setup for the end users. In our implementation, we use the Inno Setup installer, which is a third party tool for creating Windows installers. It also allows us to maintain any software license we or Northrop Grumman may prefer, since all Qt libraries we use have the LGPL license and are dynamically linked (.dll files) with the installer. This separates our executable entirely from the Qt framework and abides by the LGPL license agreement. It is seamless for the end user since the installer includes all the DLL files without requiring users to download the entire framework externally.

6.0 - Testing

Testing the application came in three waves. The first of which was incremental unit testing, then integration testing, and finally usability testing. To test our software the team has employed the use of QTest, a QT-based testing library that has two core testing functions:

- QCOMPARE();
- QVERIFY().

The QCOMPARE function is used to check the accuracy of a value against an expected value while QVERIFY checks a boolean value. QVERIFY can be compared to an assertion in other testing libraries.

6.1 - Unit Testing

Unit testing represents the majority of test cases in the application. The team has defined a “unit” to be a pivotal function. Alongside testing the “correct” functionality, the team has also included bad input tests for functions who take in parameters. These bad input tests fit into one of two categories: Incorrect type or incorrect format. Originally, the primary functions could not handle incorrect inputs or errors. These bad input tests have expanded the capability of the function’s error handling.

An incorrect type test sees the function pass in a variable with an incorrect type. Our functions will recognize the incorrect input and fail. In our tests, we predict this behavior with QVERIFY and ensure the function returns false.

An incorrect format test is similar to an incorrect type, however it is only used for a string data to node function where a string containing a node id, message and data is passed through. These tests ensure that the case of a missing item is handled gracefully.

6.2 - Integration Testing

Our application contains two main sections where functions are interconnected: Log file outputs and serial communication. The log file output involves testing not only the file generation function but also the actual log file. Testing for the serial communication involves a similar approach where the team creates a pair of virtualized ports and sends test messages to and from and analyzes the response from each.

6.3 - Usability Testing

To ensure there are no roadblocks or bugs associated with the GUI, the team is incorporating usability testing. The team created three use cases where a predicted end user might navigate to. These include connecting, viewing data on the events page, disconnecting and downloading a log file and changing the connection and user settings.

The team dispersed the application with a specific checklist to users with an average knowledge of technology, which is quantified as being able to open an application or visit a website with a web browser. The users will then complete the tasks and fill out the check list. These tests aim to uncover any road blocks associated with a user's workflow or any bugs with the GUI.

7.0 - Project Timeline

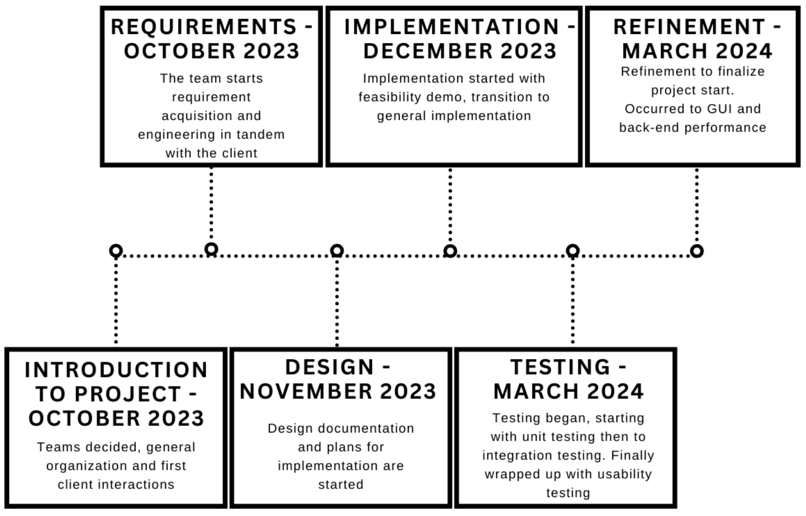

REQUIREMENTS - OCTOBER 2023
The team starts requirement acquisition and engineering in tandem with the client
IMPLEMENTATION - DECEMBER 2023
Implementation started with feasibility demo, transition to general implementation
REFINEMENT - MARCH 2024
Refinement to finalize project start. Occurred to GUI and back-end performance
INTRODUCTION TO PROJECT - OCTOBER 2023
Teams decided, general organization and first client interactions
DESIGN - NOVEMBER 2023
Design documentation and plans for implementation are started
TESTING - MARCH 2024
Testing began, starting with unit testing then to integration testing. Finally wrapped up with usability testing

Diagram 5: Project Stage Overview

After receiving our initial project guidelines from Northrop Grumman, and after garnering the most significant requirements, we began our implementation in December of 2023.

Requirements were one of the most significant aspects of last semester, where we were constantly refining our requirements every single week with the clients until they were accurate and traceable. In this second semester, we primarily focused on our biweekly sprints for implementation and testing.

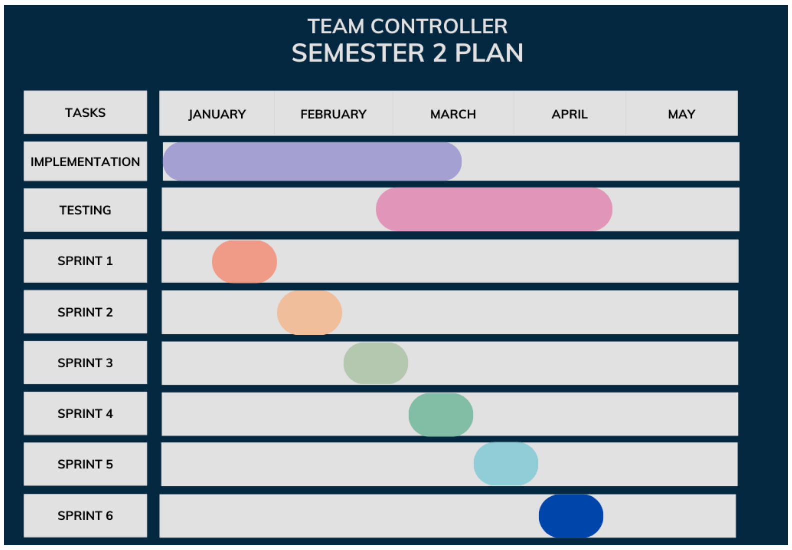


Diagram 6: Project Stage Overview

Each sprint consisted of 20 to 25 issues that were evenly distributed to each member of the team, based on the project requirements we still needed to fulfill. At the end of each biweekly sprint, the clients were given a demonstration of what has changed and what is still to come. Dylan Motz focused heavily on the front-end implementation of things, and making sure that the software was as user friendly as possible. Brandon Udall led the development of much of the controller simulation operations and the serial communication. Finally, Zachary Parham and Bradley Essegian set up most of the data structures and unit tests for the entire system. Currently, we are in an impromptu sprint 7 where we are focusing on minor UI/backend refinements.

8.0 - Future Work

For the future we will be handing over our work to Northrop Grumman. The team will be handing over software documentation, source code files, the installer, and the guides to jumpstart Northrop's software engineers. These guides will include how to use QT since the software is built around using QT and also a guide on how to use the Inno setup. Northrop's engineers will be continuing our work and they will have access to the actual weapon controller. So since our team didn't have access to the actual weapon controller we had to simulate it. Therefore we couldn't make an accurate program that would work with the weapon controller. So the team made the program modular and simple to understand so when Northrop's engineers take over there should only be a few functions that need to be changed. We also pointed out these functions that will probably need to be changed and made all the code easily readable. Other than that a few features that could be added is light mode since our program only has dark mode and maybe instead of a log file they could output the logs to another file type like a microsoft word document. Overall this software will help anyone at Northrop Grumman diagnose weapon data and save time and resources.

9.0 - Conclusion

Testing weapons deals with lots of complex data that only engineers would understand and this means an engineer needs to be dispatched to testing sites to test these weapons. This takes away a lot of time and resources from other engineering tasks. Our goal is to make an easy to use desktop application that any person can diagnose weapon data. The software we built accomplishes the following:

- Easily diagnose/filter weapon data
- The software displays the following
  - Weapon events/errors
  - Status of the weapons
  - Electrical components
- Modular and simple framework

This software will help Northrop Grumman save time and resources since the engineers won't have to travel as much. Also since the software's framework was simple and modular when Northrop's engineers took over it shouldn't take much time for it to work with the weapon controller. Overall the team is thankful to our clients for sponsoring this project and we have learned lots of valuable skills both from class and our clients that have made us better software engineers.

10.0 - Glossary

System - All files developed for the purposes of satisfying the client level requirements.

- **Data display module** - displays status updates and events to the user
- **Controller simulator** - Generates status updates and events then sends via RS422 serial protocol to the data display module.

- **Event Log file** - Will be generated by the data display module to contain 1 or more event strings encountered during a session.
- **Installer** - will deploy the project in the customers system and perform environment setup and initialization. Will record initial user preferences and take them into account during installation. I.e. “Do you want a shortcut on your desktop?”

Environment - The directory our system will be placed in and all of its contents.

Serialized data - A string of 1s and 0s which can be translated to traditional data such as strings and integers. Only serialized data can be sent through serial ports

Serial Port - The physical hardware port which can send and receive serialized data.

Status data - (See class diagram for specifics) General data pertaining to the weapon which can be measured at any point in time during session.

Event String - A string of the format “<time> <event message> <param 1> <param 2> <param 3>” generated by the controller simulator to simulate the occurrence of a weapon related event. The parameters can be NULL, but the event message must be specific text outlining what the event is. Ex. “[00:12:41] Weapon overheat 237 200” where 237 represents measured barrel temp and 200 represents max recommended barrel temp in degrees celsius.

Session - The time measured from the moment the controller sim is connected to the data display module to the moment the controller sim is disconnected from the data display module.

Handshake Protocols - A necessary set of agreements between two devices before they perform serial communication.
Electrical data - A data structure containing float values for current and voltage and a name for the component.
Command Line Interface - method of interacting with programs on a computer by inputting text commands /data/ prompts.

Automatic Log File Generation - The ability for the system to automatically generate an event log file after a session has ended.

Appendix A: Development Environment and Toolchain

Hardware

- Any computer able to run windows 10/11 and linux will be able to run our software.

Toolchain

- **QT**: Used to edit both the backend code and front end design

  Libraries:

  - Qt Serial Bus
  - Qt Serial Port

- **com0com**: Used to create virtualized ports to test and run the software on one device

Setup

- QT: Install the QT Community Edition found at this link here: https://www.qt.io/download-qt-installer
  1. Sign in with an account or create a new one
  2. Accept the terms of service
  3. Make sure to install all the required packages/libraries and beyond the defaults select the following options
     a. Under Qt 6.7.0
        i. Select MinGW 11.2.0 64-bit
     b. Libraries
        i. Qt Serial Bus
        ii. Qt Serial Port
     c. Developer and Designer Tools
        i. Select MinGW 11.2.0 64-bit
        ii. CMake 3.27.7
        iii. Ninja 1.10.2

- **com0com**:Install com0com for the virtualized ports and the link can be found here:https://sourceforge.net/projects/com0com/files/com0com/2.2.2.0/com0com-2.2.2.0-x64-fre-signed.zip/download
    1. Follow the installer setup
    2. On the Choose Components
        a. Uncheck the box labeled **CNCA0<->CNCB0**
    3. Then finish up the installer
    4. At the end of the installer select
        a. Launch Setup Command Prompt
    5. Once command prompt is launched use the following commands
        a. install PortName=COM9 PortName=COM10
        b. list
            i. This is to make sure the ports are set up correctly

Production Cycle

- QT
    1. Launch QT
    2. Once QT is open select Open Project
    3. Find the project in your folders and select the CMake files tied to the project
    4. Now you are able to edit/run the program
    5. To run the program click the green arrow on the bottom left
    6. In the program make sure on both connection page and developer page you are using the ports you setup using com0com