Project Wilbur

Requirements Document

Version 1.0

Authors: Adriel Perez, Alexander ‘Gus’ Siegel,
Nathan Seitz, Taylor Nielsen

Northern Arizona University

CS476: Requirements Engineering
Client: Christopher Mayerl Ph.D., Northern Arizona University,
Mayerl Lab
Mentor: Italo Santos

October 30, 2023

Client Signature: ______________________________

Team Signature: ______________________________

Overview:

This document is the Requirements document where the overall project, solutions, requirements and risks will be described in detail.

Table of Contents

Introduction …………………………………………………… 3

Problem Statement ……………………………………………… 4

Solution Statement ……………………………………………… 5

Project Requirements …………………………………………… 6

Potential Risks ………………………………………………… 13

Project Plan …………………………………………………… 14

Conclusion …………………………………………………… 15

Glossaries and Appendices ……………………………………… 17

1.0 Introduction

The United States is one of very few developed countries that suffers from high maternal mortality and premature birth rates. In 2022, one in ten children were born prematurely [1]. Preterm is defined as being born before the 37th week of pregnancy [1]. Infants born prematurely, often end up spending extended time in the neonatal intensive care unit (NICU), where constant monitoring provides them with constant monitoring to address health concerns before discharge. Before the hospital discharges premature babies into the homes of their parents, the babies must demonstrate a healthy ability to feed [2]. Infant feeding issues are prominent in prematurely born babies leading to issues with latching, suction or creating a vacuum seal with the mouth which can later impact speech and growth development. When interviewed, mothers of preterm infants reported inability to create sucking, swallowing in coordination with breathing as some of the reasons why breastfeeding after leaving the hospital was difficult [3].

One potential solution to infant feeding issues is a robotic nipple feeding device that has different settings making it adjustable to an infant's specific needs. An adjustable device would allow medical professionals the precise control needed to ascertain what settings provide the infant with the best solution to their particular feeding situation. If we can provide medical professionals with a tool that helps alleviate any feeding issues with their patients, perhaps down the line, medical professionals could recommend a specific bottle and nipple combination that allows the baby relief at home and throughout their infancy.

Dr. Mayerl received his Ph.D. for Biological Sciences at Clemson University in 2018 and has since then conducted numerous research programs. Dr. Mayerl received a grant from the National Institutes of Health (NIH) for his research on the impact of sensory intervention on motor output and feeding performance in full term and preterm infants [4].

For our team project, Dr. Mayerl has also applied for a grant with the NIH to help with his research and availability of assets. To further aid his research and provide opportunities for undergraduate and graduate students alike to work on real world research, Dr. Mayerl launched the Mayerl Lab which is located at the Northern Arizona University campus. The Mayerl lab has published a plentitude of peer reviewed publications that range from regional variation in contractile patterns and muscle activity in infant pig feeding to preterm birth impacts.

Dr. Mayerl is currently researching how certain feeding approaches impact piglet health and development. For this type of research Dr. Mayerl gets a group of piglets each spring to use in his research study. Piglets are the selected test subjects for this type of research because they have similar mouth and jaw anatomy and physiology to that of human infants. Each spring, half of the piglets used for research are premature and half are born after full gestation. His on campus lab, Mayerl Lab, uses a team of student researchers, to provide 24 hour care for the animals and to investigate how certain aspects of feeding methods impact a myriad of factors in the piglet, including respiration, heart rate, mouth pressure/suction, and bone positions. Most elements of the research process are sufficiently advanced to collect clear information, but a major problem exists in how the feeding method is delivered and how the milk and nipple properties are changed.

2.0 Problem Statement

In the past, Dr. Mayerl has needed a participant for managing the piglets during the research study, a person to manipulate and hold the bottles used, a person to man the computer and watch the graphs as well as control the manual trigger that denotes the use of the X Ray, and a person to control the X Ray. Our product allows Dr. Mayerl to save not only time but also reduce the amount of people needed to successfully run the research sessions. Having an automated way to control the prototype means he does not need to wait for different nipples to be replaced on the bottle, or wait for a new mixture of milk to be made for feeding. If during the study, the researchers identify three different bottle setups they want to use on the piglet, they can do it with almost no wait time as opposed to needing to make and change out three different bottles. This also means they will not need to adjust or turn off the X Ray in order to do this because there will not be a human participant in the path of the X Ray that they need to protect from harmful rays.

Some of the variables that appear to influence feeding have been identified as flow rate, as it relates to milk viscosity, and nipple stiffness. Dr. Mayerl and his team have performed several studies researching the impact these variables have on piglet feeding behavior [5]. However, to best investigate the impact of these variables, he would be able to dynamically change one or more of these variables while an individual piglet was feeding and record the physiological results in real time. This would provide real-time information on how healthy and premature piglet feeding is impacted by these changes, leading to a clearer understanding of potential feeding solutions. To accomplish this, Dr. Mayerl needs a robotic/pneumatic nipple whose properties can be changed on demand.

Our team is responsible for creating a control software that provides Dr. Mayerl with the ability to change prototype properties quickly and efficiently from his desktop computer. While having a robotic nipple with manual controls that modify properties would be one solution is simply building one is not enough; the robotic nipple needs software to actuate control. As the computer science team on this project, this is our responsibility. By interfacing with the team constructing the prototype and Dr. Mayerl, we plan to develop software that provides control of this robotic nipple and meets requirements. These are the primary problems solved by our software:

- Dynamic control of nipple stiffness.
- On-demand modification to milk viscosity.
- Immediate, discrete setting of number of ducts open.
- Real-time graphical display of properties changed on Dr.Mayerl's PowerLab charting system.
- Grouped and available set of controls for robotic nipple.

3.0 Solution Statement

Our planned solution is to develop control software for this prototype robotic nipple. The system overview appears in Figure 3.1. It will consist of a front end interface hosted on a desktop application, back end software to send the necessary signals to the prototype, and a connection to the Data Acquisition Center (DAC). A user will be able to control the three properties of the robotic nipple all from a single software control interface. The front end software will be a desktop app installed on the control computer. A user of our control software will exclusively interact with this component. Its primary function is providing a graphical interface from which a user can change properties of the robotic nipple.

Working closely with the front end will be what we are calling the back end software, which will be the component of the desktop app responsible for taking inputs from the front end and forwarding them to the microcontroller, through a USB connection. In essence, this component will not be particularly challenging, since it is purely an information exchange layer.

The last major component of our solution is the actual control software. This will be code hosted on the microcontroller, and be responsible for moving the actual hardware integrated into the robotic nipple. While the goal for this component is exceptionally clear, ensuring it can actually achieve that goal may be its own challenge. We anticipate challenges in having discrete control over the properties of the robotic nipple. Achieving consistency and overcoming other issues that come with hardware are two of many potential roadblocks for this portion of the solution.

As an additional, somewhat separate element of our software solution, we will develop a robotic nipple simulator that will act identically to a physical prototype. Instead of being a true prototype, however, this element will serve as a way for us to demonstrate met requirements, without necessarily having a prototype to run our software on. Since our solution and its requirements are highly dependent on some kind of robotic nipple, we determined this component necessary as part of our solution.

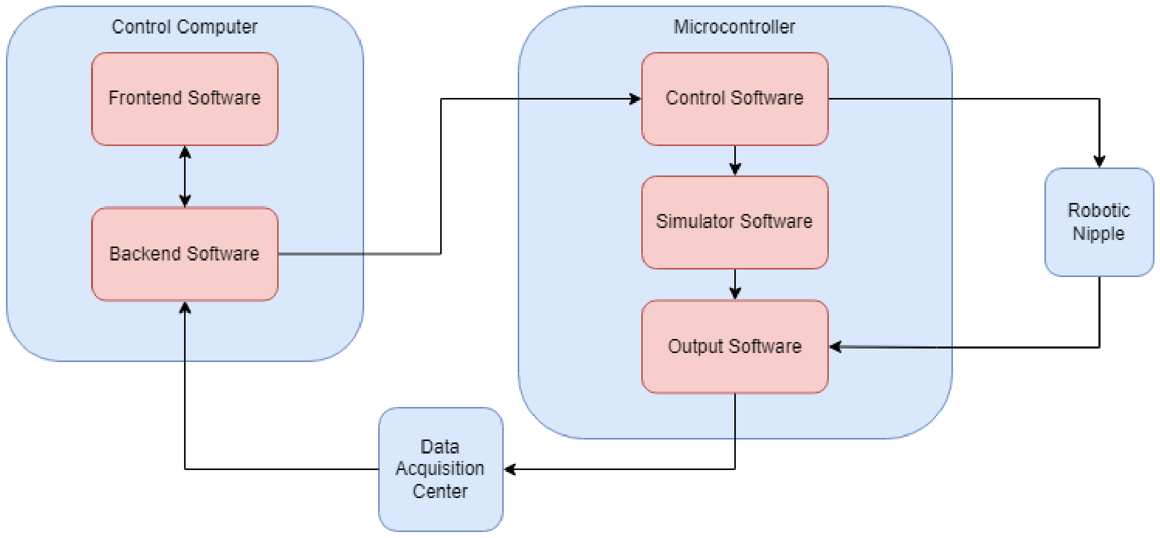


Figure 3.1: Solution Overview Control Software Architecture

The above diagram illustrates the envisioned solution. Marked in blue are hardware components we will be working with, and pink represents software components, developed by our team. Arrows designate flow of data.

4.0 Project Requirements

While creating our design plan for our project, we gathered our requirements from our client as well as suggesting some of our own. The following dissects our functional, non-functional and environmental requirements. Functional requirements are objectives we must have in order to have a successful, complete product. Non-Functional requirements are objectives we can include to improve the use and feel of the product. Environmental requirements are restrictions placed on our team and development plan due to barriers or laws outside of our control.

4.1 Functional Requirement

4.1.1 Front End Functional Requirements

The user interface(UI) and front end functionality will be created in Qt using Visual Studios tool integration for Qt. We will implement the Model-View-ViewModel design architecture for organizing our code, seen in figure 4.1. Model-View-ViewModel is a design architecture for front end programming that helps the developer visualize and easily understand how bindings are created and which code files directly impact other features of the code. For example, for each new window created we want that window to have its own view, viewmodel and model file. Each file has its own required outline and inclusions in order to make the program easy to read and change. Qt is the desired framework for the front end because of its use of the C++ language and high level of online documentation.

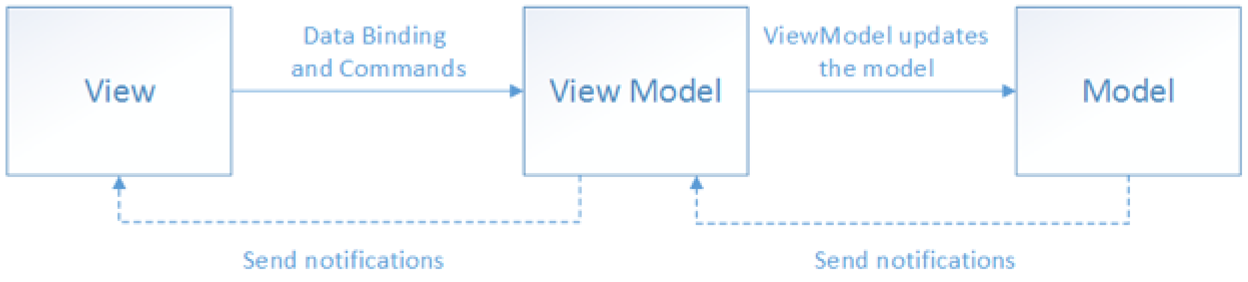
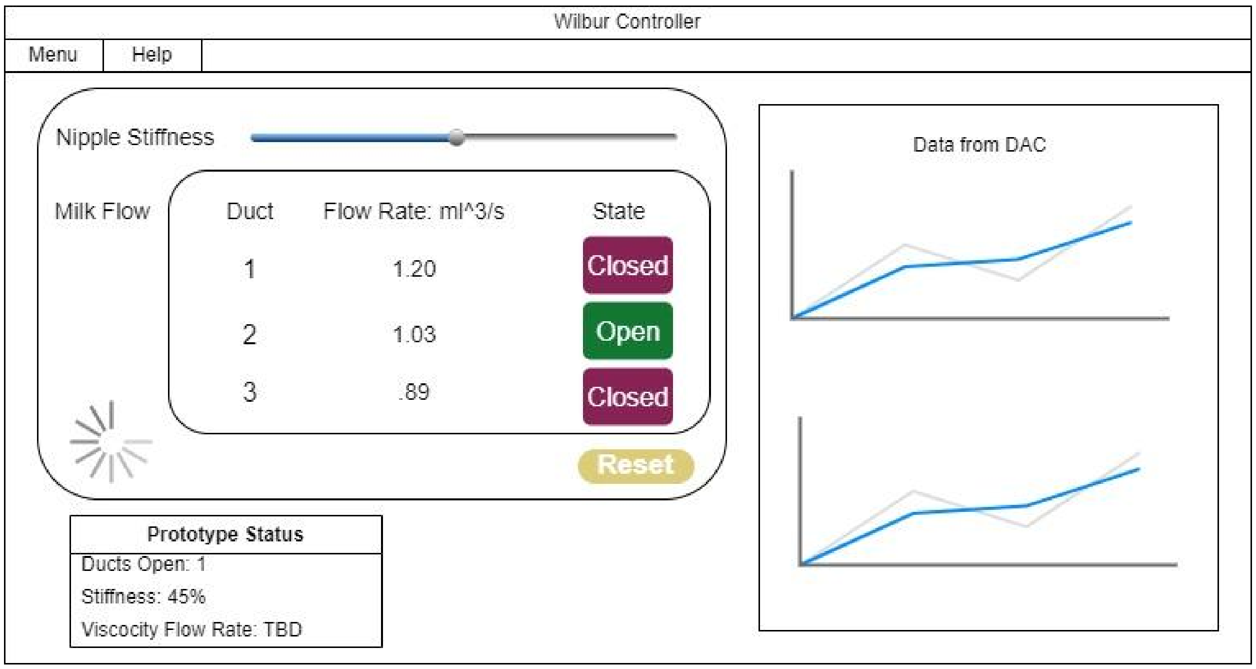


Figure 4.1: Model-View-ViewModel Design Architecture [6].

Below, Figure 4.3 shows a very basic UI Design. We will be implementing a menu bar across the top of the window to provide additional navigation functionality and access to methods like prototype information and help documentation. The help button in the menu bar will call a function that displays our user tutorial and will link to the user documentation.

Our UI will implement tooltips for all grid elements to provide individual instructions or information to the user after a brief period of hovering over the element. The upper left quadrant within the window will be the control system for the actual prototype. We will be able to make changes to the prototype from here by selecting which parameter to change.

The prototype controlled by our software has 3 duct systems. Each system needs individual control. As the ducts are opened and closed, milk viscosity and flow rate will change based on the viscosity in each reservoir currently being released. For us to be able to use viscosity, the user will have to enter the flow rate of each reservoir into the corresponding grid for each duct. We will incorporate a check and balance that will prevent the user from shutting all three ducts at once.



Duct	Flow Rate: ml^3/s	State
1	1.20	Closed
2	1.03	Open
3	.89	Closed

Prototype Status
Ducts Open: 1
Stiffness: 45%
Viscocity Flow Rate: TBD

Figure 4.3: Mockup of Front End User Interface

We will implement exception handling alerts to cover multiple types of errors. This will alert our user to any issues in data being gathered and when troubleshooting or resetting may be required. These exception alerts will be a new small window that pops up with a specific message detailing the error with a confirmation button.

Front End Functional Requirements Review:

- Development Tools and Architecture:
    - UI and front end functionality developed in Qt using Visual Studios tool integration for Qt.
    - Implementation of the Model-View-ViewModel design architecture for code organization and easy expandability.

    - **Use Case:** When a developer needs to add a new feature to the user interface, the Model-View-ViewModel design architecture will facilitate the addition without affecting other components of the system
- Programming Language and Framework:
    - Utilizing Qt framework because of its extensive online documentation.
    - Front end written in C++ to match the hardware language, mitigating compatibility issues.
    - **Use Case:** Since the hardware communicates in C++, developers will be able to ensure seamless integration and minimize compatibility issues between software and hardware by coding in C++.
- Dynamic UI:
    - Graphical user interface window dynamically sized to the monitor it is displayed on.
    - Buttons and graphical displays adjust with window size to maintain proportions and increase visibility on larger displays.
    - **Use Case:** When running the application on different sized monitors, the user interfaces dynamic display will ensure that UI elements maintain correct proportions, enhancing user experience and visibility.
- Menu Bar and Navigation:
    - Menu bar for additional navigation and access to methods (e.g., prototype information, help documentation).
    - The Help button in the menu bar will display tutorial documentation and links to user documentation.
    - **Use Case:** Users can quickly access additional information and elements of the user interface, such as prototype details, help documentation and tutorials through the menu bar.
- Tooltips and Control System:
    - Tooltips for grid elements to provide instructions or information after hovering.
    - Control system in the upper left quadrant for making changes to the prototype parameters.
    - **Use Case:** Users can hover over grid elements to see tooltips specific to that element which help provide useful information or instructions, aiding the users understanding and interaction with the system.
- Duct Systems and Viscosity:
    - Individual control buttons for prototype duct systems.
    - User entry of flow rate for each reservoir in the corresponding grid for each duct.
    - Check and balance to prevent the user from shutting all three ducts simultaneously.

    - **Use Case:** Users will be able to individually control each duct system, the check and balance system will prevent the accidental closing of all three ducts, which would result in the system being unusable.
- Exception Handling:
    - Implementation of exception handling alerts for various types of errors.
    - Popup window with specific error message and confirmation button.
    - **Use Case:** When the user encounters a system error, the exception handling system presents a clear error message in a popup window. This helps the user understand and rectify the issue promptly.
- Simulation Mode:
    - Navigation between physical prototype and simulator.
    - Method for entering simulation mode (e.g., command line arguments or mode selection in the menu bar).
    - Clear indicator when the user is in simulation mode.
    - **Use Case:** Users can seamlessly identify and switch between working with the physical prototype and the simulator, ensuring future developers and users alike can simulate scenarios without affecting the actual hardware.

4.1.2 Back End Functional Requirements

The back end of our software will be the integration point between the front end and the microprocessor. The back end will take inputs from the front end and convert them into inputs that the Arduino can act upon. For example if the front end triggers a function for closing ducts one and three, it is unlikely that the Arduino will easily understand the input regardless of its type so to combat this we will write an intermediate function between the Arduino and the front end to ensure the data from the front end is translated correctly to the Arduino.

The back end will be written in C++ and it will be required to not only que the Arduino to change parameters but also when to trigger signals to the DAC showing that the change has been made on the prototype. If we are unable to signal a change on prototype to the DAC, the user will continue recording data without knowing that the prototype state has changed which will result in that batch of data being inert.

The back end must also be able to take the signal received from the front end and translate it into a state change on the robot. For example, when the button that controls the flow rate is pressed, the back end must be able to read which one of the duct systems was chosen, and open the control chamber. This behavior must be consistent for all buttons.

The back end will have a method to check for prototype errors or malfunctions. This will prompt the front end to throw exception handling errors. At a minimum, exceptions will be thrown if one or more duct valves fail, as well as an exception if the pneumatic pump fails. With each exception, the found failures will be described, and the severity of the failure will also be attached to the exception, whether a minor failure, partial failure, or full failure. We need to implement prototype accuracy monitoring to prevent the researchers from proceeding under the impression that the prototype has changed features correctly when there in fact has been a failure.

Back End Functional Requirements:

- Intermediate Function:
    - Implementation of an intermediate function between the Arduino and front end to ensure correct translation of data.
    - Ensures that commands from the front end are translated for appropriate interpretation by the Arduino.
    - **Use Case:** Users will be able to directly control the hardware by manipulating the settings through the user interface instead of manually adjusting the prototype themselves.
- Programming Language and Queuing:
    - Back end written in C++.
    - Ability to queue Arduino for parameter changes.
    - Trigger signals to the DAC to indicate changes made on the prototype.
    - **Use Case:** Developers can develop code with efficient communication with the Arduino, preventing data conflicts and ensuring a smooth interaction between the software and hardware.
- Translation of Signals:
    - Translates signals from the front end into state changes on the robot.
    - **Use Case:** As the user interacts with the front end, the back end translates these signals into meaningful state changes on the robot.
- Error Checking and Exception Handling:
    - Method to check for prototype errors or malfunctions.
    - Signals front end to throw exception handling errors.
    - Minimum exceptions for duct valve failures and pneumatic pump failures.
    - Describes found failures and attaches severity (minor, partial, full) to each exception.
    - **Use Case:** The back end actively checks for prototype errors or malfunctions. If a failure is detected, the back end signals the front end to throw exception handling errors, allowing the user to address the issues in an immediate fashion.

4.1.3 Microcontroller Software Functional Requirements

Our front end solution must integrate with the Arduino that is controlling the robotic nipple. The Arduino already has a manual control system, but not a remote control system. It is our job to create the connection between the controller and the robotic nipple prototype. Our solution should be able to change any feature on the robot, and do it in a way that supports Dr. Mayerl's research. To do this we are going to run functions that change the metrics of the robotic nipple. These functions are going to control the robot's stiffness, duct system, and viscosity of liquid produced. In theory, these functions are going to be pre-built onto the robot, but we will be verifying the accuracy of the pre-built functions and making any additional modifications that improve speed, timing and signal triggering.

A second goal of the connection to the microcontroller is to have a second connection point to Dr. Mayerl's DAC. That device is what he uses to get data from many of the instruments synchronized during the experiment. The goal of this second connection is to accurately get data from the DAC when a change in the nipple happens. This will be done using output signals. The Arduino should initiate a different signal per prototype change to display multiple signals on one graph or multiple. This will allow our client to coordinate data taken from the piglet xRays and synchronize it with which change. These experiments are happening in real time, and because of that getting accurate data at the point of change is important. To accomplish this goal, we are going to create an encoding system to be able to tell when and what change was made.

Microcontroller Functional Requirements Review:

- Change Features on the robot
    - Stiffness can be controlled
    - Milk duct can be chosen
    - **Use Case:** The User will be able to accurately control features on the robot.
- Interface with DAC
    - Read data with existing hardware
    - **Use Case:** The user will be able to read data from the with the DAC

4.1.4 Simulator Functional Requirements

The simulator effectively acts as our safety net in the event that a robotic nipple prototype is not ready for us to program. Due to this, it is highly necessary that the simulator is as versatile as possible. We want the simulator to work at any level in the system, to ensure we are able to show other components are meeting requirements. This primarily means two things: the simulator must be able to be configured as a purely front end simulator, and it must also be able to be configured as a microcontroller simulator.

A purely front end configured simulator would act as the last layer in the desktop app. Where the desktop backend sends signals to a microcontroller via USB connection, our simulator must be able to act as a virtual "microcontroller", in that it must be able to receive the same signals and output the expected signals.

The microcontroller simulator configuration would be more integrated, in that there would be additional programming on the microcontroller, and it would intercept the signals being sent out of the microcontroller. Instead of the microcontroller sending signals to the hardware components of the robotic nipple, this configuration would have the simulator acting as the robotic nipple's hardware. The simulator would replicate the inputs and outputs of the robotic nipple, without involving the movement of any actual hardware.

Running the system in simulator mode will be designed to be as intentional as possible, to avoid situations where the user unknowingly starts the software in simulation mode, and expects to see hardware movement. Accomplishing this will involve the handling of additional arguments upon program startup, which must be provided through the command line. This will make the simulation mode exceptionally difficult to enable accidentally, while also not being inaccessible for the client, in the event it becomes necessary for the client to run the control software in simulator state.

In both front end and microcontroller configurations, the simulator will have an enableable option to simulate hardware failures. All simulated hardware elements will be capable of being set to fail to varying degrees of severity, determined by additional passed command line arguments. When active, the simulator will always return the specified types of failures whenever the system attempts to modify the state of that hardware component. For example, the simulator might be initialized to simulate the prototype with no duct failures, but toggled to simulate a moderate pneumatic pump failure. If ever the control software calls for a modification of the simulated pneumatic pump, this configuration of the simulator will return a moderate failure. Consequently, since the simulator was not configured to simulate duct valve failures in this example, any modifications to the simulated duct valves will return no errors, and simulate as though all those components are working perfectly. Simulator exception creation is necessary to check if our software correctly handles a hardware failure. Since we cannot guarantee that the hardware will fail while we test it, we must be able to verify our software will properly react by simulating such a failure.

Simulator Functional Requirements Review:

- Configuration
    - Purely front end simulation
    - Microcontroller configuration
    - **Use Case:** Depending on setup configuration, the user can see simulated results of just a front end desktop, or the entire system.
- Hardware Failure Simulation
    - Valve simulated failure
    - Pneumatic pump simulated failure
    - Both types of hardware failure
    - Variability of failure severity
    - **Use Case:** With additional input arguments upon startup, the user can simulate how the front end handles hardware failures of variable severity
- Returned values
    - Assume successful change if not configured otherwise
    - Based on configuration status, return failures
    - **Use Case:** Through the values returned by the simulator, the user can see verification of hardware changes within the time restriction

4.2 Non-Functional Requirements

4.2.1 Front End Non-Functional Requirements

According to the ACM Code of Ethics, developers should design technology that is easily used by all people and avoids exacerbating inequities of people with differing abilities [8]. To remain in accordance with the ACM Code of Ethics, we plan to integrate certain states that enhance the user experience for individuals that suffer from visual impairment(s).

Our front-end will include contrasting colors and our control elements will be large enough for people to see. While developing the front end there will be an emphasis placed on accessibility for color blind users and users with any other visual impairments. We will use high contrast, color blind friendly palettes for diagrams, graphs and themes. All diagrams and graphs will have a legend that includes a brief description of the purpose or function of the image. Buttons and navigation tabs will use common vernacular found in many front end designs and will not use obscure words that are vague and non-descript.

Non-Functional Requirements Review:

- Accessibility Design Principles:
    - Adherence to the ACM Code of Ethics, focusing on designing technology accessible to all people and avoiding exacerbation of inequities.
- Visual Impairment Considerations:
    - Integration of features to enhance the user experience for individuals with visual impairments.
    - Front-end design with contrasting colors for improved visibility.
    - Control elements designed to be sufficiently large for easy visibility.
- Color Blindness Considerations:
    - Emphasis on accessibility for color-blind users during front-end development.
    - Use of high-contrast, color-blind-friendly palettes for diagrams, graphs, and themes.
- Accessibility Features:
    - Inclusion of a legend for all diagrams and graphs, providing a brief description of their purpose or function.
    - Buttons and navigation tabs designed with common vernacular, avoiding obscure words that are vague and non-descriptive.
- User-Friendly Design:
    - Front-end elements designed with user-friendly principles to enhance accessibility for a diverse user base.

4.2.2 Coding Convention Non-Functional Requirements

Each member of the team has taken the data structures course where we were made to code using a set of standard coding conventions that can be easily applied to any other language. We will maintain the coding convention across all pieces of code for the project. The coding convention will help all members easily understand new pieces of code and it will help non-coders, such as our client, understand how to change and manipulate the code after the team has graduated and left the project. Coding conventions will also help create an easier structure for maintainability and expandability for our release versions.

Basic Coding Conventions:

- No single letter variable names.
- Methods, Classes and Variables will be named using camel case.
  - Classes will Start with a capital letter.
  - Methods and Variables will start with lowercase letters and use uppercase letters to designate the start of a new word.
- Constants will be capitalized and use underscores to separate words.
- Variables and method names must be self descriptive.
- Methods must have an implementation description that includes the following categories.
  - Method name:
  - Purpose:
  - Dependencies:
- Developers will avoid redundant code by creating methods to implement one function in multiple places.

4.2.3 Simulator Non-Functional Requirements

The simulator would normally be considered non-functional because it is not part of the core system, but we are considering it functional overall. However, there are non-functional requirements to go with simulating an Arduino connection to our front end. The simulator must accurately simulate the robot, down to timings and sending signals back to our front end. We feel that this must be done because we need a way to be able to test everything about our front end system.

Simulator Non-Functional Requirements

- The simulator’s response time must be the specified time of 3 seconds or less
- The simulator code must adheres to conventions of the entire system
- The connection to the microcontroller must mimic prototype as close as possible

4.2.4 Back-end Non-Functional Requirements

Our back-end needs to meet specific metrics that are non functional. One such metric that the back end needs to meet is the ability to change a parameter on the robot quickly. This means that the robot must make a change and the signal must be sent back to our front end in 1-3 seconds. The transformation of the robot doesn't need to be instant or even very fast because we are dealing with biological data. Most of the data will be gathered in the order of seconds rather than milliseconds. Having a good, consistent modification speed will help Dr. Mayerl get good data back from the robot for his research.

The back end must also be:

- Accurate:
  - Consistent
  - Reset to neutral after use
- Modular:
  - Future-proof
  - Able to use other robots
- Readable
  - Follows the guidelines outlined in 4.2.2

4.3 Environmental Requirements

The prototype and control software is made specifically to study how modifying nipple settings and milk viscosity can improve infant feeding. The prototype itself will need its own power source while the control software could be hosted on either a desktop or a laptop computer. Without some kind of power source our product will be rendered inert.

Our product also requires that the study subjects be hungry when the prototype is in use. If the piglets are not hungry and refuse to engage in some type of latching with the prototype, we will not be able to employ our control software effectively. Our control software can be observed to work without the presence of piglets but it can not serve its purpose without active participants.

When creating a product or prototype that has a medical implementation for humans, one must be very careful about how they plan research and development before proceeding with research studies. The robotic/pneumatic nipple is being designed and created in order to address a specific need in the medical community and address a specific health issue among human infants. The United States Department of Health and Human Services is one of the main governing bodies that dictates how research studies for infants proceed [7].

We have been told a projected date of January for the finished prototype to arrive, however, there could be issues in the completion and delivery of the prototype. If the prototype is not completed on time or is damaged during shipping we will be forced to strictly rely on our simulation system to test our front end and back end code. Our simulator should allow us to accurately complete our side of the control project. Once completed, our client should be able to set up the prototype with our control code, if he does receive it after our project is completed.

Environmental Requirements Review:

- Power Source Requirements:
    - Prototype requires its own power source for functionality.
    - Control software will not be compatible with android or ios and therefore, must be hosted on either a desktop or laptop computer.
- Study Participant Conditions:
    - Study subjects must be present and hungry when the prototype is in use.
    - Active participation of piglets required for effective use of the control software.
- Medical Research and Regulatory Compliance:
    - Research and development planning must adhere to guidelines set by governing bodies, such as the United States Department of Health and Human Services.
- Projected Prototype Arrival Date:
    - Projected completion and delivery of the prototype by January.
    - Potential issues in completion or delivery may necessitate reliance on the simulation system for testing.

5.0 Potential Risks

There are many potential risks to our solution. One of the main risks that we are aware of is that the prototype might be fundamentally different from what we are expecting. To our knowledge, the prototype will come with functionality, and it will be manually controlled via buttons. If the prototype we receive does not have functionality, we will have to add that functionality, which would eat into development time for other parts of the project. This risk has a very low chance of happening

A second, and much more likely risk, is that the robot prototype will not be delivered on time. This will eat into development time, but as long as the prototype works we will be fine. We have access to a second prototype for testing purposes, and as long as there is functionality, this should not be too bad of a risk. The worst case time delay would be a day or two to make sure the controller works as expected with the new prototype. This risk has a fairly high chance of happening.

There is a risk of incompatibility between the software that we are using. One of our stretch goals is to incorporate the splitting of data via a signal on Dr. Mayerl's DAC, which in theory should not be too difficult. The issue could come from the API of the DAC not being able to work with C++, our preferred back end language. This should not be a problem because we can jerry rig something together that will connect the two together. For example, some of our group members have fixed a similar issue by running a python instance inside of C++. There is an extremely low chance that this happens.

Our front end will communicate with the robot via a serial port, but Arduino is not the most reliable device when communicating through the serial port. The way that we are going to communicate over the serial port should be consistent, but there is a chance that the buffer on the Arduino causes reading errors on the robot itself. This should not happen, but if it does it would take about a week to find and fix the problem.

The front end is built with a library that makes the project open source because of licensing. If the client changes his mind about allowing code to be published on Github because he is reaching the end of his research and wants to pursue the for profit market, we would need to change the front end library that we use. QT is open source, if the project it is being used on is open source and any changes that are made to QT are sent back to the main library. If this project becomes for profit, we lose the license and would have to remake the front end using a new library. Seeing how this project is a research project, this is extremely unlikely to happen. If it were to, it would add months to our timeline. Table 5.1 shows each risk along with their chance of happening, severity, and expected time delay.

	Chance of Happening	**Severity**	**Time delay**
Misunderstand the level of functionality of the robot	3/10	9/10	Month(s)
Robot is delivered to us late	6/10	2/10	Day(s)
Incompatible software	2/10	5/10	Week(s)
Serialization error	2/10	6/10	Week(s)
Licensing Concerns with QT	1/10	9/10	Month(s)

Table 5.1: Chart describing the potential risks, how severe and the estimated time lost fixing them.

6.0 Project Plan

To ensure that the team maximizes efficiency when developing our product, an Agile method of development will be adopted since it allows adaptability to requirement changes. The team has come up with a total of 10 milestones that we would have to accomplish in a timely manner in order to be able to develop a well-rounded product. These milestones have been compiled into a Gantt chart labeled as table 6.1 to allow for an easy way of determining what is to come.

As we begin the process of development in the upcoming months, the team would first like to define what our coding conventions, such as issue tracking and resolving, would look like since all members of our team code differently. This will be the first thing to be established as early as January 2024 and being set in stone as the development process takes place.

After this, the team would start working on the front end which includes the approval of the UI design, discussions with the client and team members to perceive which functionalities are appropriate and/or feasible to add, how the back end can be connected to the front end, and how the testing will function along with a prototype simulator. This development will cover a time frame of two months (January-February) to allow polishing of features and debugging.

Concurrently, the acquisition of the Arduino device and the pneumatic/robotic nipple prototype should also be taking place in late January or early February of 2024. The start of development for the Arduino code and its components will include the back end code and the communication with the latter, how the back end is controlled via the prototype simulator, and how the prototype properties will be controlled and managed with the Arduino. The development process of this milestone will last four months (January-April) since the process of acquiring components and actually incorporating code that can establish a connection between the Arduino and the prototype will require rigorous testing. A stretch goal that is worth mentioning is also including a connection with Dr. Mayerl's DAC.

Finally, the team will also create a simulator that can work as a means of testing the Arduino and front end functionality. The simulator development will include unit tests that will be devised to ensure that the Arduino code is scalable and debugged. On the other hand, the team will also make sure that the simulator and Arudino use the same input and output streams to ensure that the simulation can accurately replicate the change of properties in an actual, physical prototype. This last milestone will also take four months (early January-late April) to complete.

Task	Start	Efforts (hrs/week)	January	February	March	April	May
Issue Control and Conventions	TBD	1	🟦				
Front End Development	TBD	4	🟦	🟦			
Back End Development	TBD	4	🟪	🟪	🟪	🟪	
Prototype Simulator	TBD	4	🟪	🟪	🟪	🟪	

Table 6.1: Gantt chart describing the team's upcoming milestones

7.0 Conclusion

To wrap things up, our project aims to help combat the real-world problem of feeding issues. These can lead to problems that can snowball throughout the life cycles of many infants of the upcoming generations. To help with this cause, we want to contribute to Dr. Mayerl's research with a functional robotic/pneumatic nipple prototype whose stiffness, fluid viscosity, and ducts can be modified to adjust to the specific needs of the piglet, and potentially infants, to help combat or identify root causes of feeding problems.

Because our team wants to provide a way to control the properties of the prototype visually, the team will create a front end interface that allows Dr. Mayerl and further down the line, other researchers, to adapt the prototype's properties without having a need to understand what is changing under the hood. To be able to connect the physical prototype with the front end, a back end software will be developed to ensure that the instructions sent from the front end are communicated with the back end to then apply those changes with the prototype. To be able to test the changes in properties and the communication between the front end and back end, a simulator will be created to allow for further testing and diagnostics before moving on to testing on the actual, physical prototype.

To make sure that all features for our project are given enough time for polishing and fixing, the team has developed milestones to ensure that the final product is as complete as possible in every aspect. The team decided to establish the conventions of how code will be written as well as deciding how issues will be handled. After the latter, the start of development for the front end, back end, and simulation will begin simultaneously, with the front end being the first feature to finish. Meanwhile, the back end and simulation will take more time since these require more complex testing as these are the components that are interacting with the physical prototype and the DAC.

Our team wants to acknowledge as many risks as possible before starting our development, especially since physical components are involved. The most crucial risk to take into consideration is that of delivering a product whose functionality does not meet the clients expectations. To combat this, the team is committed to meet with Dr. Mayerl when possible to clear as many uncertainties as possible of what is and is not asked of us. Conversely, another risk that could happen is for software to be incompatible with any tools or equipment Dr. Mayerl needs to use for his research. To help prevent this, we want to steadily release software updates to ensure that the equipment we need to interact with is responsive. Finally, not receiving the actual prototype is a low-impact risk since the development does not fully depend on the physical prototype, but rather on the front end, back end, and simulation.

Finally, the team feels on track with the development process. On one hand, we are awaiting the arrival of an Arduino device to be able to demonstrate that a connection between the latter and the computer is possible. On the other hand, a barebones front end interface in Qt has been created to show the client what the user interface structure will look like, the team will gladly accommodate any changes requested by the client. Overall, the team is optimistic that it will have something of high quality to show for its efforts as the semester comes to an end.

References

[1] “Preterm Birth | Maternal and Infant Health | Reproductive Health | CDC,” *www.cdc.gov*, Oct. 24, 2023.
https://www.cdc.gov/reproductivehealth/maternalinfanthealth/pretermbirth.htm#:~:text=In%202022%2C%20preterm%20birth%20affected

[2] R. Kamity, P. K. Kapavarapu, and A. Chandel, “Feeding Problems and Long-Term Outcomes in Preterm Infants—A Systematic Approach to Evaluation and Management,” *Children*, vol. 8, no. 12, p. 1158, Dec. 2021, doi: https://doi.org/10.3390/children8121158.

[3] G. Vizzari *et al.*, “Feeding Difficulties in Late Preterm Infants and Their Impact on Maternal Mental Health and the Mother–Infant Relationship: A Literature Review,” *Nutrients*, vol. 15, no. 9, p. 2180, Jan. 2023, doi: https://doi.org/10.3390/nu15092180.

[4] P. Fellow, “Christopher J Mayerl EDUCATION.” Available:
https://www.mayerllab.com/uploads/1/4/3/5/143527008/cv_2022-08.pdf

[5] Selected Publications | Mayer Lab - UC Santa Barbara. 2023. Accessed: Nov. 29, 2023. [Online]. Available: https://www.mayerllab.com/publications.html

[6] *The Model-View-ViewModel Pattern*. 2021. Accessed: Nov. 13, 2023. [Online]. Available:
https://learn.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[7] Office for Human Research Protections, “45 CFR 46,” *HHS.gov*, Feb. 16, 2016.
https://www.hhs.gov/ohrp/regulations-and-policy/regulations/45-cfr-46/index.html

[8] Association for Computing Machinery, “ACM Code of Ethics and Professional Conduct,” *Association for Computing Machinery*, Jun. 22, 2018. https://www.acm.org/code-of-ethics