CS CAPSTONE FINAL REPORT

MAY 30, 2020

VOLUME-BASED ENTERAL FEEDING CALCULATOR

PREPARED FOR

DR. JUDY DAVIDSON

PREPARED BY

GROUP 60

ENTERAL FEEDING CALCULATOR TEAM

ALISON JONES

PARKER OKONEK

Abstract

The enteral feeding calculator is an open-source, windows desktop application written in C# and based off of the .NET framework. The purpose of the calculator is to provide an easy to use, dependable, and accurate replacement for enteral feeding paper calculation tables. The application presents a minimal interface with solid internal logic constructed from months of nurse feedback and continued development and validates user data on file read to ensure valid and recent feed times for its timeline.

CONTENTS

1 Forward: Release Notes 1.0

We plan to continue development on this project into the summer. There are a few bugs that remain (see bug updates in the GitHub repo). The project was developed in the .NET Forms framework. This means that the current release can be run on Windows devices with .NET installed. The installer will ask to auto install .NET if you do not have it. The framework we chose also means that there will need to be additional work when converting to another platform or for web hosting. This is where our continued development is headed. For those who want to host the app themselves they can use the desktop app already provided, however we also plan to push our tool to the web as it will be easier for our clients to integrate into their systems.

2 Introduction

The enteral feeding calculator was requested by Dr. Judy Davidson at University of San Diego Health in order to provide nurses with a more accurate and easy to use alternative to hand calculations of tube feeding rates and the current method of calculations using a paper table of hours missed and their corresponding feed rates. The calculator allows for higher accuracy in the resulting rate by providing a simple interface for nurses to input varying length time gaps and specify previous feeding rates. Both sets of values are used in combination with patient and institution specific information to return a new feeding rate for the patient which is accurate within 1 millimeter per hour.

2.0.1 Roles

The application was developed by Alison Jones and Parker Okonek. Alison Jones developed the timeline interface, designed the main interface layout, and led team discussions with the client and other groups. Parker Okonek developed the file input/output system, internal logic for managing patient info, and validation on file i/o and internal logic.

Dr. Davidson acted in a supervisory role along with Mike Mercer, guiding the implementation of the project and creating opportunities for showcasing progress with nurses for user studies.They helped provide the connections needed to discuss future project implementation and expectations as well as the IT perspective on their end. Judy was a great resource for the user perspective while Mike outlined the original solution and why it is problematic.

2.0.2 Covid-19 / Spring Schedule Changes

The new schedule enabled us to look at the project more thoroughly with extra feedback and time for making improvements. Specifically, the code review allowed us a second chance to see the feedback from our peers along with an additional week to make revisions afterwards. The initial timeline that we had created was much faster paced and looking back, may have been too ambitious. Despite the extra time, there still wasn't as much testing as we had planned for. Despite not ending as far as proposed, we believe that the added time definitely resulted in an improved quality of the work on the project. The major drawback to the extension was that as a 2 credit course, it had more work than expected.

2.0.3 Future Use

Many quality of life features could be added to the program, although they are not required for the full and functional use of the calculator. Epic integration is an always existent help for hospitals and nurses but cannot be accomplished without close and internal development with a hospital's IT team.

The most requested feature from nurses during the development for capstone was giving titration values when increasing the rate of feeding. In the context of enteral feeding, titration describes the process of gradually increasing the feeding rate in even increments to allow a patients gastrointestinal tract to adjust to the higher rate of feeding. The implementation of this feature in the software would allow nurses to step up each increase in feed rate with the click of a button.

3 Requirements

3.1 Introduction

3.1.1 Purpose

The volume-based enteral feeding calculator replaces the paper tables currently used when calculating feed rates for both gastric and small bowel enteral feeding. The calculator reduces error by requiring the source numbers to be entered and automatically produced, unlike the current method which requires the result to be produced by hand. The goal of this conversion is to reduce both error and time taken in getting the results.

3.1.2 Scope

Paper tables have been designed for nurses to use to calculate the catch up rate. The table is a commonly used rate-based system with a max rate cap based on the method of feeding. Many hospitals don't even use this method and just had calculate the catchup rate. The current method leaves some room for error and could be improved/automated.

The application and its source code will be released on GitHub. Both the compiled binary and the source code will be appropriately licensed for its own open source distribution and to be license compatible with any libraries, code, or other intellectual property used to create the application.

The platform for the project will be determined from surveying nurses to determine whether mobile or desktop is more convenient and widespread for projects of similar scale and niche.

3.2 Specific Requirements

3.2.1 External Interfaces

The enteral feeding calculator will interface with local storage on the machine to store any generated error logs and to save values which are not likely to change after being first set. The calculator will also display to the user's screen and input devices via a UI library to interface with the operating system's screen drawing routines and to accept pointer and text input from the user.

3.2.2 User Requirements

The user interface will be minimal and easy to read. The application will have no configuration window or views other than its main interface, and on its main interface will have the text inputs for the different input values: time of feeding stop, time of feeding restart, reset time of feeding, and daily feeding volume.

Time inputs will be only selectable options in drop down menus. Any time with a PM will have 12 added to its value automatically before being used for calculations. A time with PM at the end cannot have an hour value greater than 12. Volume inputs will accept only integer values. A dropdown menu will be used to display all available feeding types: Nasogastic, Nasojejunal, Percutaneous Endoscopic Gastronomy, Jejunostomy. The selected feeding type will be displayed on the closed dropdown and on the informational banner in the top left. At the bottom of the UI will be the output value and a calculation button that updates the output value, if valid.

3.2.3 Functions

The program will take the user's input for the various fields of the finished algorithm. The application will use these values to compute the enteral feeding rate for the remaining time of the day but not exceed safe maximum rates. The program will have modes for both gastric and small bowel feeding, which will alter the maximum rates and the calculations for rate. The app should be able to get date and time information for these calculations. As the user enters and updates information, it should be updated where applicable. The variable representing the time of feeding reset should remain constant between the application closing and being opened, but remain configurable when opened.

3.2.4 Software Quality

The enteral feeding calculator will need to have support for multiple versions of its target platform. Medical settings often have a variety of versions for device operating systems. The application will support legacy and current versions of Windows, from Windows 7 to Windows 10.

The software should not crash, freeze, or return erroneous results over normal use. The results returned by the program should never exceed the maximum safe values for their kind of enteral feeding, even with erroneous input. Results should also be consistent over up time of the application and regardless of the uptime of the operating system.

The user interface should indicate when a value is accepted or rejected, and offer further clarity through its interface with minimal clutter.

3.3 Verification

The project should decrease the time to calculate an enteral feeding rate by at least 25% with a 50% decrease being unlikely. The application should increase accuracy of nurse calculations. Current calculations are inaccurate since they do not address multiple feeding rates during the day. Our calculator will have an accuracy of 99.9%+ compared to the previous method which often results in around 80% accuracy. It can be more accurate in less extreme cases, but as the schedule becomes more complicated the accuracy drops! After being proven to work, the final project will hopefully make it into clinical testing. This is the true test to verify how effective the project is. The application should ultimately be well received and have a lowered chance for error. We can test this by comparing the percent error before and after implementation.

3.4 Gantt Chart

Items

1. Go over the rate table/ process that nurses go through when calculating rate based feeding or catch up rate for feeding.
2. Research which interface to use. Contact nurses and learn whether a phone or computer application would be better. Also ask about inputs/ first design check at this time.
3. Research App development based on chosen interface.
4. Develop Algorithm/ math and logic side to app.
5. Integrate algorithm into app framework.
6. Add time based input for application.
7. Ensure user input is saved locally on device to preserve data.
8. Work on app display and usability (pretty stuff)

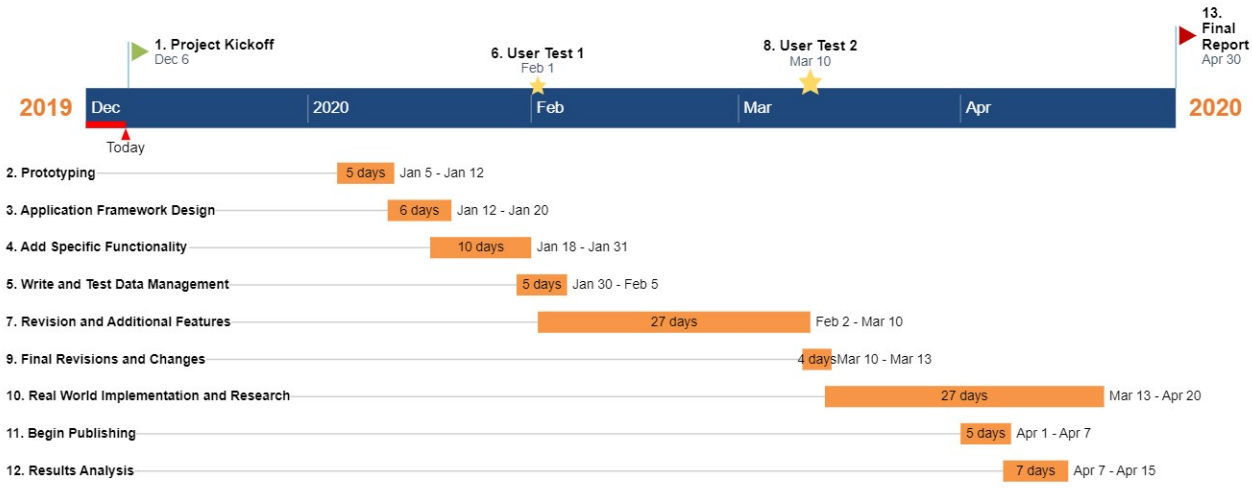

Fig. 1. Gantt Chart

9. Check app design again with nurses/ projected users.
10. Update app design based on feedback. (repeat as necessary).
11. Finalize Application and publish? Research what is needed to do with this.
12. Check whether we can start implementation and research.
13. Start if we can.
14. Final Report and Wrap-up

3.5 Changes

3.5.1 User Requirements

Previous Text	Current Text
Time inputs will allow only numerical values, a colon, and optionally AM/PM inputs. Any time with a PM will have 12 added to its value automatically before being used for calculations. A time with PM at the end cannot have an hour value greater than 12. Volume inputs will accept only numerical values.	Time inputs will be only selectable options in drop down menus. Any time with a PM will have 12 added to its value automatically before being used for calculations. A time with PM at the end cannot have an hour value greater than 12. Volume inputs will accept only integer values.
A checkbox will be used to indicate whether the feeding type is gastric or small bowel, with a large banner indicating the current state.	A dropdown menu will be used to display all available feeding types: Nasogastic, Nasojejunal, Percutaneous Endoscopic Gastronomy, Jejunostomy. The selected feeding type will be displayed on the closed dropdown and on the informational banner in the top left.

3.5.2 Software Quality

Previous Text	Current Text
Medical settings often have a variety of versions for device operating systems. If the mobile platform is chosen, the software will need to support multiple versions of both the Android and iOS mobile operating systems. If the desktop platform is chosen, the software will need to support legacy and current versions of Windows, from Windows 7 to Windows 10.	The application will support legacy and current versions of Windows, from Windows 7 to Windows 10.

4 DESIGN

4.1 Overview

4.1.1 Purpose

The main goal of the project is replace the manual method of calculating volumetric enteral feeding rates with an automated, digital method that results in less errors and takes less time to complete. This document outlines the development path of the project and details the implementation and design on the calculator.

4.1.2 Scope

The calculator will return adjusted feeding rates to accommodate for missed feeding time for a patient from any variety of factors. The user experience will focus on presenting a minimal and easy to read design where the user, a nurse or dietitian, enters in the minimum needed and easiest to derive data to determine the feeding rate and is returned the adjusted safe rate to catch up on daily volume.

4.1.3 Intended Audience

The intended audience of this design document are the sponsors, Dr. Judy Davidson and her team, and the student development team. The document serves as a reference for the student development team to track the product creation timeline and as a representation of the sponsor's needs and product expectations that is verifiable and referable.

4.2 Definitions

Enteral Feeding: A form of tube feeding used to deliver the appropriate nutrients to a patient.This is done through either the mouth, esophagus, or directly into the digestive system through an artificial opening. (Not to be confused with intravenous administration of medication!)
Volume-based tube feeding: The method of calculating hourly rates of tube feeding in an adjustable manner to assure a specified caloric intake daily.
IRB: An Institutional Review Board is a group that has received authority to review, deny, approve, request modification to, and monitor biomedical research in its institution.

4.3 Project Context

4.3.1 Hardware

The software will support first and foremost a desktop platform for both its development and production. The project is developed for desktop and can be completely developed using normal desktop hardware and be expected to run on common consumer desktops. If the project is developed for mobile, it will be able to run on most off the shelf smart phones or tablets.

1) System Architecture: i386 or amd64
2) Screen Resolution: 1024 x 768 minimum
3) Input: Keyboard and Mouse, or on-screen keyboard and pointer

4.3.2 Software

Software requirements for development will involve github and the chosen user interface framework, Windows Form App, whose libraries are required to build and compile the project. The user interface toolkit will not be required for common use by end users.

4.3.2.1 Desktop Requirements:

1) Version Control: Github
2) Operating System: Windows 7 or greater
3) DotNet 4.2 or higher
4) UI Toolkit: Windows Form App

4.4 Design Description

4.4.1 Design Stakeholders

Nurses: Nurses are our primary target user since they will be the ones directly using the application. Nurses conduct all the direct work including setting up feeding bags, doing the current calculations for catch-up feeding rates, and

administering food. Nurses also track how much people eat in the electronic health record.

Dietitians: These are the people who calculate caloric needs and recommended diet. When the diet requires tube administration, they determine the caloric need, which formula to use, and the feeding rate. Dietitians also monitor response to feeding in the form of nutritional biomarkers/laboratory tests and alter the recommendations for feeding based on the results. Dietitians are a secondary end-user of our application.

Physicians: Physicians order the nutrition, usually based upon a recommendation by the Dietitian. They are the people who check that the prescription and administration are performed accurately. They effectively oversee the process.

Some key stakeholders in our project include, Michael Mercer, a dietitian in charge of dietitian standards, Patricia Graham, a nurse lead providing us lots of insight on how nurses do their jobs and what they might want or expect from our application, Dr. Heyland, the person who invented the original feeding rate catch-up chart; We are effectively automating the usage of this chart and turning it into an application, and Judy Davidson, the nurse and research scientist overseeing our project, providing resources for our success.

4.4.2 Design View

4.4.2.1 Users: The enteral feeding calculator is a nurse's tool to easily retrieve an adjusted feeding rate. For nurses, the end users, using the calculator will be a simple and time-efficient method when compared to a paper table or manual calculation. The user-facing interface for the calculator will have a minimal number of inputs needed to reliably calculate the adjusted feeding rate to decrease possible input errors and promote common usability principles.

A simple interface lends itself well to both learnability, how easy it is for new users to learn the interface, and efficiency, the speed of task completion when a user has learned the interface. The interface will also feature either a prominent calculation button or automatic result return field to aid in efficiency and readability.

4.4.2.2 Stakeholders: Our key stakeholders (Judy Davidson, Mike Mercer, Dr. Heyland, and Patricia Graham are mostly concerned about the technical side and implementation of the project. The main goal of this project is to create an application that will increase the accuracy of the administering nurses; avoiding mistakes made during feeding that can be dangerous for patients. Despite directly influencing the nurses by providing them an application, this also makes the job of dietitians and physicians easier; avoiding additional catch-up/adjustments to the original proscription.

4.4.3 Design Timeline

4.4.3.1 Gantt Chart:

4.4.3.2 Items: 1. Go over the rate table/ process that nurses go through when calculating rate based feeding or catch up rate for feeding.

2. Research which interface to use. Contact nurses and learn whether a phone or computer application would be better. Also ask about inputs/ first design check at this time.

3. Research App development based on chosen interface.

4. Develop Algorithm/ math and logic side to app.

5. Integrate algorithm into app framework.

6. Add time based input for application.

7. Ensure user input is saved locally on device to preserve data.

8. Work on app display and usability (pretty stuff)

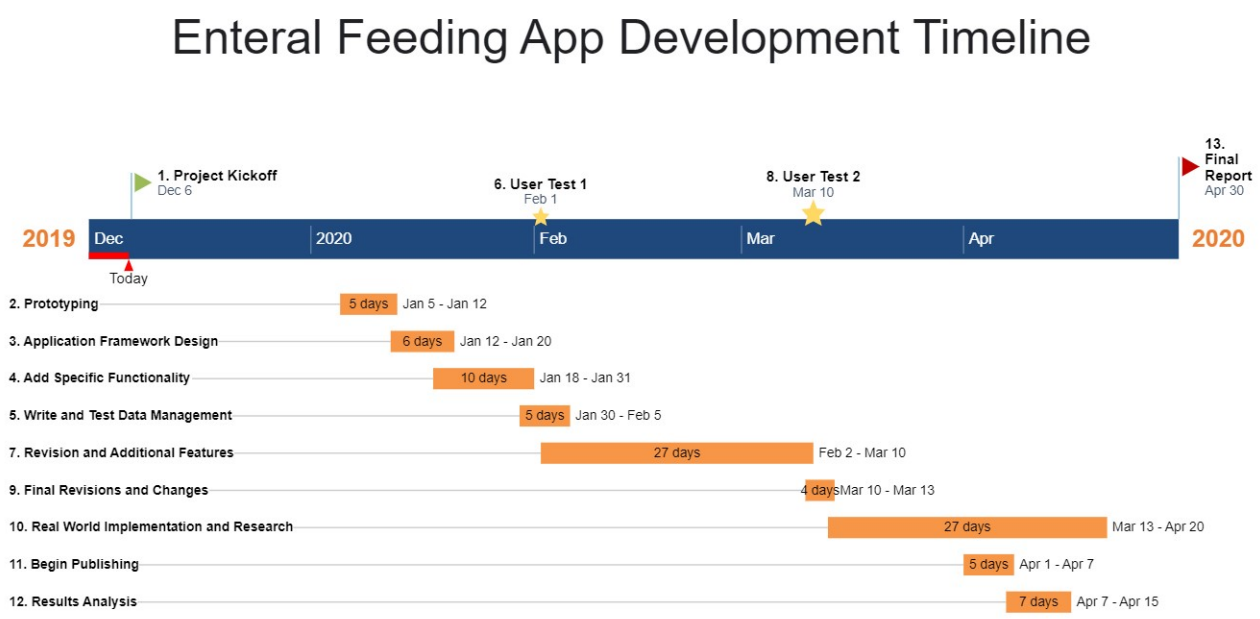

Fig. 2. Gantt Chart

9. Check app design again with nurses/ projected users.
10. Update app design based on feedback. (repeat as necessary).
11. Finalize Application and publish? Research what is needed to do with this.
12. Check whether we can start implementation and research.
13. Final Report and Wrap-up

4.4.4 Design Viewpoints

4.4.4.1 Context Viewpoint: The application should present a more accessible and easier to use option than the current methods of calculating the adjusted volumetric feeding rate. The user experience should be hassle-free, without any unexpected slow-downs, crashes, or graphical artifacts within the program's control.

A select test group of nurses will be surveyed to determine the optimal platform to improve usage rates and satisfaction. A similar group, or the same group, will also be asked to review paper prototypes of the applications basic UI layout. Once the interface for the program is developed, it will be tested by the developers, sponsors, and target users to verify the app functions as expected and returns accurate results.

4.4.4.2 Legal Viewpoint: In September of 2019, the FDA released a set of changes to medical software policies due to the 21st Century Cures Act. Due to these changes the enteral feeding calculator on a mobile platform would fall under the regulatory category of "mobile medical app" due to its use as an accessory to a regulated medical device [9].

The enteral feeding tube calculator may not be considered a non-significant risk device under FDA laws though could be approved under the app's IRB. The app does supplement the functionality of a pump that needs to have a rate of flow entered to properly feed a patient; however, if the calculations done by the app are considered to be a "simple

task" as described in the FDA's *Policy for Device Software Functions and Mobile Medical Applications* then this would not be the case. The application generates a flow rate that is entered into a medical device without outside verification and for patient safety should have its accuracy and error-rate examined by an IRB.

4.4.4.3 Composition Viewpoint: The project will be composed of three distinct pieces: a user-facing UI to collect and report data, a system interface to manage local storage and compatibility, and the internal logic which calculates results from user input and keeps the program state.

The user-facing interface presents an easy to read form input for the minimum required information to calculate a result and passes those values to the internal logic, when a result is created the interface will display this to the user as well.

The system interface manages the constant values between application runs that is not subject to regular change, for example daily target volume or time of feed reset, and allocates required resources from the operating system. The internal program logic manages the resulting rate and integrates the other two components of the program with itself and each other. This component will also manage the notifications to the user for the UI, such as missing inputs, invalid inputs, and indicating the new rate has been calculated.

This composition ensures that the UI framework can be replaced in the event of the current choice losing active maintenance or not adapting to the changing software landscape. It additionally separates the program logic from the interface display and system interfaces, reducing latency from calculation and allowing the program logic to remain consistent and constant regardless of other changes.

4.4.5 Design Rationale

- Uniform among all hospital rooms.
- Secure/ isolated from outside influence.
- Easier transition to third party.
- Accessible to many users.

4.4.6 Deliverables

4.4.6.1 Goal: The final product should be an application that automates and optimizes catch-up rate calculations for nurses and dietitians. We have two possible platforms on which to develop our app: mobile and desktop. The main goal is to complete an app for one of these (depending on most desirable platform) and conduct both simulated and field tests with the final product.

4.4.6.2 Stretch Goal: An additional goal, if we complete the first application ahead of schedule, is to develop, test, and implement another version for the secondary platform (the lesser desirable option for nurses). This means that we would have both phone and computer applications available,

4.4.6.3 Out of Scope: Integration with user health records or third party health management system. This would take a long time for approval due to sensitive data and is not necessary for the main function of the application.

4.5 Methods

4.5.1 Application Logic

4.5.1.1 Approach: The math driving the calculations is only required to be called on user input events and does not change dynamically throughout the course of its use. The language used for the application varies on the platform

used for the project. The calculation function will only be called when valid inputs are received, and the form of the functions will take a data-driven approach. Many languages implement features which allow for an optimized and easy-to-read data-driven implementation of functionality.

4.5.1.2 Concerns: Software developers will focus open source development effort in projects that are in languages they prefer, and both Python and Javascript remain popular languages with many enthusiastic followers. The language chosen will affect the platform versions that are available, while a language such as C++ can run on many platforms without the need to install an additional runtime per computer, a language such as Python is less often used by common users and would require more stringent packaging and installation management to ensure minimal effort between searching for the program to opening it for the first time.

4.5.2 Interface Framework Integration

4.5.2.1 Approach: Bound variables from the user interface will be read by the underlying code. These inputs will be checked for validity and converted into their proper data types. Once converted into the types required for the application logic's calculations, the volumetric rate output is calculated and returned to the user. The interface between the framework and the application logic will be performed using the Model-view-viewmodel design pattern.

4.5.2.2 Concerns: Software developers on open source projects focus their efforts highly on projects where they have the requisite skill. Model-view-viewmodel is a common design pattern for UI integration in desktop and mobile applications. User Interface designers that wish to improve the design will find familiarity with web design frameworks, such as Vue.js, which uses a model-view-viewmodel method to interface between client and server [5]. Users will experience a lower performance impact compared to other options, as the application will remain static when not in use.

4.5.3 Time-based Input Calculations

4.5.3.1 Approach: This section is important when reducing the math nurses have to do. Instead of asking for total amount administered or number of hours a patient was fed, we could have nurses simply track as they go. Nurses can tell the app at what time they started and stopped and the rate that the feeding was set to. The app should be able to figure out the current time and use past entries to tell nurses what the future rate should be. In this case, nurses wouldn't even need to calculate the number of hours missed, the app should do it. Depending on the platform being used the method can slightly change, however it will always remain fairly standard. We will grab a timestamp of Day:Hour:Minute:Second for calculations.

4.5.3.2 Concerns: We want to ensure that the application is as accurate as possible. Will the device have accurate time? Will the application know if times don't line up properly? Additionally, if times overlap or aren't in the correct order, it must tell the user an incorrect input was entered and reject the input.

4.5.4 Data Management

4.5.4.1 Approach: The app should be able to keep track of data being entered and preserve it until the user clears it for another patient. We may even be able to add the ability to manage multiple patients. The app should also preserve time information so that the timezone and start/stop time don't have to be entered more than once. They should remain static unless the user decides to reset them.

4.5.4.2 Concerns: The most important rules for our data to adhere to include persistence, accuracy, and security. Persistence ensures that the data being defined (such as starting time, total volume, and intervals) continues to exist even when the application is closed and reopened. This can be important when closing the application or even in an emergency such as a power outage. Accuracy and security ensure that the data remains constant unless dictated by the user and that it cannot be changed outside of the application (by user or third party). This ensures the data is not tampered with and does not change between uses. Below are some data storage examples and their respective benefits and concerns.

Shared Preferences: Great for standard settings, this method of information storage saves strings in key-value pairs. This may actually be sufficient for our application unless we need to start saving objects (which may be possible if we allow for multi-patient management).

Internal Storage: This method of data storage allows information to persist without being viewed or changed by other applications or the user. This is kept hidden from the user and due to this, may be ideal for our application. We only want the user to be able to interact with user, time, and rate data from within the app.

External Storage: This method of data storage is used when the user wants to export information to view or send someplace else. This method is not idea since the application won't be exporting anything. All data being stored is used by the app and shouldn't be viewable or editable by the user unless within the app. (even then some data is still obfuscated)

4.5.5 Interface Design

4.5.5.1 Approach: The chosen framework is one that makes the application easy to interact with and enjoyable while also reducing the amount of developer time to achieve that goal. A desktop platform will be the main focus of development and framework selection.

4.6 Conclusion

The enteral feeding calculator will first and foremost be an application to quicken and reduce the time of administration time for nurses and quicken responses for dieticians, but the calculator solves a variety of other issues as well. The calculator will offer nurses, its users, a simple and reliable interface to produce their needed values. The design and open source nature of the process will provide administrators, providers, and dieticians an easy to justify and available product. The architecture of the project provides future developers and contributors an easy start on upkeeping the project after its initial creation to ensure the application continues to improve. Utilizing a UI framework for the interface of the project makes an app that is visually consistent with many other applications that also use it in addition to reducing development time and improving maintainability. All of these aspects lend themselves to a enteral feeding rate calculator that is simple, fast, accurate, and a proper replacement for paper tables.

4.7 Change Table

Original	Updated
1) System Architecture: ARMv8 2) Screen Resolution: 720 x 1280 minimum 3) Input: Pointer and on-screen keyboard or pointer and physical keyboard	Removed
Software requirements for development will involve github and the chosen user interface framework, Qt, whose libraries are required to build and compile the project.	Software requirements for development will involve github and the chosen user interface framework, Windows Form App, whose libraries are required to build and compile the project.
1) Version Control: Github 2) Operating System: Windows 7 or greater 3) UI Toolkit: Qt (other options, see below)	1) Version Control: Github 2) Operating System: Windows 7 or greater 3) DotNet 4.2 or higher 4) UI Toolkit: Windows Form App
Assuming a multi-platform or desktop implementation, Qt, supports a variety of platforms with similar or identical interfaces with the underlying programming language. The layout and bindings to underlying data will be written using Qt's provided API, and the required libraries will be packaged with the software as allowed by the license for distribution.	Removed

Mobile Interface Framework React Native: A very popular cross-platform option for Android and iOS development. This framework is inclusive and uses JavaScript. This makes it very familiar, especially for our team who hasn't had experience with mobile app development. This may be the easiest option to learn of the three listed here. Apache Cordova / PhoneGap: This framework seems to emphasize the ease of use with peripherals such as the phone's camera and other apps. While this is great, we likely won't need to make use of this. The benefit of this framework is that many versions of an application can be created using the same code base which makes developing for multiple platforms easier.	Removed

An few examples of what this may look like include: Assuming we use the Ionic framework for a mobile interface we would use the ate() function to grab a timestamp [3]. let date = new Date() console.log(”Current Date ”,date) Assuming we use JavaScript for a web interface we would use the getTime() function to grab a timestamp. Date.getTime() Assuming that we use Cordova framework for a mobile interface we would use the currentdate class to grab the current time data [4]. var currentdate = new Date(); var datetime = currentdate.getHours() + ”:” + currentdate.getMinutes() + ”:” + currentdate.getSeconds();	Removed

5 Tech Reviews

5.1 Alison Jones

5.1.1 Overview

Current clinical technology is often outdated. While medicine and doctors have improved their effectiveness over the past few decades and technology has reached amazing capabilities, many tools used in a clinical environment lack any automation or computer software to accompany them. Enteral feeding, a frequently used medical procedure, is an example of this disjoint relationship. The current method for determining feeding rates is lengthy and repetitive while also leaving large room for human error.

The calculations being made can easily be automated and presented to nurses in the form of an app. It is our team’s job to develop this application and test it in both simulated and clinical environments. The application should take minimal input from nurses and provide them with the correct feeding rate and be released with open source to aid in deployment.

After development, the app will be taken into clinical trials to test it’s efficacy over the original hand-calculated method. To conduct trials our team must verify that we follow the rules outlined by the FDA and IRB.

The actual platform that we are using for this project is still unknown until later this week. Additionally, if we choose to use a mobile framework, this calculation will be done differently depending on the the one we choose. This makes it

difficult to create a list of options. I could easily list 5 or more methods and we still may not use any of those precise commands. Ideally, it will just be a sort of gettime() or date() function. Some frameworks will require us to convert it to the correct format, but as for how it retrieves this data, it doesn't matter. We may be able to look at the different options within a framework we ultimately settle on later.

An few examples of what this may look like include:

Assuming we use the Ionic framework for a mobile interface we would use the date() function to grab a timestamp.

```
let date = new Date()
console.log("Current Date ",date)
```

Assuming we use JavaScript for a web interface we would use the getTime() function to grab a timestamp.

```
Date.getTime()
```

Assuming that we use Cordova framework for a mobile interface we would use the currentdate class to grab the current time data.

```
var currentdate = new Date();
var datetime = currentdate.getHours() + ":"
        + currentdate.getMinutes() + ":"
        + currentdate.getSeconds();
```

5.1.2 Data Management

The app should be able to keep track of data being entered and preserve it until the user clears it for another patient. We may even be able to add the ability to manage multiple patients. The app should also preserve time information so that the timezone and start/stop time don't have to be entered more than once. They should remain static unless the user decides to reset them.

5.1.2.1 Shared Preferences: Great for standard settings, this method of information storage saves strings in key-value pairs. This may actually be sufficient for our application unless we need to start saving objects (which may be possible if we allow for multi-patient management).

5.1.2.2 Internal Storage: This method of data storage allows information to persist without being viewed or changed by other applications or the user. This is kept hidden from the user and due to this, may be ideal for our application. We only want the user to be able to interact with user, time, and rate data from within the app.

5.1.2.3 External Storage: This method of data storage is used when the user wants to export information to view or send someplace else. This method is not idea since the application won't be exporting anything. All data being stored is used by the app and shouldn't be viewable or editable by the user unless within the app. (even then some data is still obfuscated)

5.1.3 Interface Design

This is the section to make the application look pretty. We need to choose a framework that makes it easy to interact with and enjoyable.

The obvious solution for any web-based computer application would be HTML and CSS and building off of that with something like JS. However, we may not make a computer application. If this is the case we will need to research some options for mobile apps. Here are some options:

5.1.3.1 React Native: A very popular cross-platform option for Android and iOS development. This framework is inclusive and uses JavaScript. This makes it very familiar, especially for our team who hasn't had experience with mobile app development. This may be the easiest option to learn of the three listed here.

5.1.3.2 Apache Cordova / PhoneGap: This framework seems to emphasize the ease of use with peripherals such as the phone's camera and other apps. While this is great, we likely won't need to make use of this. The benefit of this framework is that many versions of an application can be created using the same code base which makes developing for multiple platforms easier.

5.1.3.3 Ionic: This framework comes with a lot of default UI features and integrates Cordova for easier access to features like the camera and contacts. It allows for cross-platform development, making it easy to build an app for both Android and iOS. This could be a really good candidate for our project due to it's inclusive nature.

Edit: After more research I found that publishing with Ionic is not free while React Native is.

5.1.4 Conclusion

Our interface (and therefore framework) that we will be using fro this project is still unknown. Of all the frameworks, Ionic seemed to be the best candidate, however I discovered that it is also not free to publish with this framework. This means that we will likely use React Native if we do pursue a mobile app route. With regards to data preservation, we want to make sure that it is saved in a way that can't be edited and won't be deleted. The best method for this need would be saving data in internal storage since it is flexible enough to hold more than keys and remains obfuscated from the user and other applications. Lastly we will need to figure out how to get time data for whatever interface we settle on. This is generally pretty easy and code examples are included in this section above (Part A.).

5.2 Parker Okonek

5.2.1 Overview

Our team will be selecting technologies for the volume-based enteral feeding calculator. This application will replace the paper tables currently used by critical care nurses when determining the amount of food or medicine to provide a patient per hour when being fed via a feeding tube. The calculator will need to be easy to use for nurses with a variety of technology backgrounds and present a clear, easy to understand interface regardless of its platform. The calculator will also provide results with a minimum accuracy and error rate equivalent to the paper table while also allowing a user to efficiently use the interface to manipulate these calculations.

I will be focusing on what user interface framework our team will choose for the project, the methods used to implement the logic, and the integration between the underlying logic and the displayed inputs and outputs. The user interface framework will need to be stable and easy to use, so that my teammate can quickly make changes and develop the

interface's form without extra development cost. The application logic will also need to be flexible to changes in the form of the interface follow a method of programming that lends itself well to the UI framework's APIs.

5.2.2 User Interface Framework and Platform

5.2.2.1 Platform: A desktop focused app could be built on a user interface (UI) framework for Windows, Android, iOS, or a generic cross-platform library. Our team has yet to determine whether the desktop or mobile platform will be preferred by nurses; because of this, a major focus for consideration is cross-platform compatibility. There are few user interfaces that contain compatibility for both mobile and desktop platforms with even fewer offering compatibility across both iOS and Android. The chosen framework should be compatible with the open source license of the project and not prevent the release of the product's own code. If the framework requires a license which restricts it from being released on a freely available open source project, it cannot be used for this project. These stringent requirements present two major options for a UI library: JavaFX or Qt. In our project we have selected the Qt for its increased breadth of programming languages. **JavaFX**

JavaFX is the standard GUI library for the Java ecosystem and, as a result, is available on all of the non-embedded platforms that are supported by Java's virtual machine. Java and JavaFX have platform support for Windows, MacOS, Android, iOS, among others giving it excellent cross-platform support [**jplatforms**]. However, JavaFX as a standard Java library is only available for the Java language, or languages that are compatible with Java such as Scala. This restriction would restrict our team form working in any other language than Java, preventing us from choosing another language for the project based on our other requirements. A recent change in Oracle's licensing has also required enterprise costumer to pay fees for usage of the language's runtime environment, which could hamper adoption in many hospitals that may not have an active license. **Qt**

Qt offers wide cross-platform support with the listed platforms of Windows, macOS, Android and iOS [**qplatforms**]. This wide platform support would ease development by reducing considerations of which platform to choose and which framework to select for each supported platform. Qt is widely used allowing for a wealth of documentation and a broad base of information to reference when building an application. This wide user base also increases language support, with third party APIs available for Python, Go, C#, Haskell, Java, and others in addition to its native C++ bindings [**qbindings**]. A wide selection of language bindings allows the user interface framework to not be a deciding factor in the selected language, instead the language can be be chosen by which features let it fit best to the task of making an enteral feeding calculator with source that is easy to read and modify.

5.2.3 Application Logic

The basic application logic will focus on performing multiple sequential algebraic equations with inputs in the form of time since last feeding, mode of feeding, and others. The form that this functionality takes can be in the form of a functional or data-driven approach. **Functional Programming**

Our application is narrow in its scope, which allows for it to be easily defined in functional programming. This method would result in a simpler implementation but would also be less easily integrated into the UI framework. Additionally many of the languages supported by our chosen framework, Qt, such as C++, do not support this paradigm efficiently or with language features. Functional programming itself could lead to performance issues as many processors and libraries are constructed to operate imperatively rather than functionally [**haskell**]. **Data-Driven Programming**

Data-driven design easily matches the scope of this application. The application will receive a simple set of pre-defined

data and then return a matching set of calculated values. Our program has no internal state that needs to be managed, making this a non-consideration. This method of design is well implemented in multiple languages and UI frameworks. Data-driven programming is shown to create clearer code in regards to the data it represents and the changes the data undertakes, while also allowing for performance improvements with certain idiomatic design patterns [**datadriven**]. As data-driven design does not have the same limitations as functional design, it is the choice for this project going forward.

5.2.4 Interface Framework Integration

Once the user interface is designed and the basic algorithm for the enteral feeding rate are created, they will need to be linked using a method that does not significantly increase overhead. The integration can take the form of multiple design patterns, including model-view-controller, model-view-presenter, and model-view-viewmodel. **Model-View-Controller** This design pattern can be regarded as a classic for logic and UI integration. This method does, however, have many caveats. JavaFX can be used very effectively with this design pattern as it was created with it in mind [**jplatforms**]. This design pattern allows for a simple overview of the tasks each subsubsection of the code should perform and makes the responsibilities of the UI defined as a display, but requires a certain level of object-oriented concepts to be easily and idiomatically implemented [**mvc**]. A model, a view, and a controller are all objects with specific interactions and data flows that allow a more or less direct concept to code implementation, but our code is not necessarily object oriented for this application. This restriction to object-oriented design, combined with the presence of alternatives which rely on it less, removes this option from consideration. **Model-View-Presenter**

This design pattern has a cyclical data flow and is implemented in three parts: a passive view, a presenter, and a model. The model in this case is the same as the model in MVC and drives all the program functionality while the presenter cleans user input and presents information from the model to the passive view, and the passive view returns values from its input but does not perform its own functionality, waiting for the presenter to update its state [**mvp**]. This design pattern allows for a complete separation of the underlying functional code and the information displayed to the user. It would also allow for user input to only be equivalent to the value needed for the model's calculations instead of equal. This design pattern would also help facilitate persistence of data needed for the application in an easy to implement manner. Because of these considerations, this will be the design pattern used for integrating the UI to the business logic.

Model-View-Viewmodel

This design pattern is more complex than both model-view-controller and model-view-presenter. This method has a model with a state, a view, and a view model that automatically updates the view and the model based off of state changes in each [**mvvm**]. This pattern would be unsuitable for this project as it assumes a high level of state mutability and constant updates from the user and the business logic. Our program is almost entirely stateless, requiring inputs for a function that are then output to the user. The previous or next states are both irrelevant to the current output or input.

5.2.5 Conclusion

The project has several requirements deriving from its goal of a multi-platform, open source application. These requirements create other expectations including code maintainability, an easy to read and transferable UI, and keeping dependence on other libraries to other open source compatible code bases. Qt meets these requirements for UI framework, by allowing use in open source and existing on a plethora of separate platforms, namely Windows, Android, and iOS. Data-driven programming will provide a consistent, well known format for laying out the architecture of the code that is equally implementable on many of the programming languages with supported Qt bindings. To merge the UI

with the business logic in a data-driven format the project will use model-view-presenter in concert with Qt to keep the code maintainable and revisable.

6 Weekly Blog Posts

6.1 Alison Jones

6.1.1 Fall

Oct 18, 2019:

Progress: Timeline and basic application logic are outlined. Requirements for implementation in the clinical environment is being researched.

Problems: The interface to write our app is not known yet. Both team members do not have experience with app development.

Plans: We must do research with app development after surveying nurses about which interface might work best.

Oct 24, 2019:

Progress: The team is currently scheduled to start the interview process and get feedback for our designs. We have a meeting set for a week and a half from now.

Problems: we will need feedback from multiple sources so we should make sure we have access to this.

Plans: Create a few design plans for the app before the next meeting. **Nov 1, 2019:**

Progress: I have read through most of the information provided to me. I will need to finish reading and creating my versions of the app design (mock-ups).

Problems: we need to create different designs before our next meeting. We will need to understand the problem enough to explain to nurses during interviews.

Plans: We have a meeting next week to discuss the desired platform to use and to talk with nurses about an optimal design.

Nov 8, 2019

Progress: We had a meeting with Mike who designed the original feeding rate table that we are turning into an application. The call went over the specifics of the table.

Problems: I was expecting the call to include more discussion about interface or planning. It mainly reviewed the logic behind the math being used. Meetings aren't very frequent. We need to take a step up and start planning our own as well for issues we run into. It's difficult sometimes because everyone seems to be in different schedules.

Plans: We still need to get together with some nurses for interview. This can be in a call with Judy or on our own at local hospitals (assuming permission is granted).

Nov 16, 2019:

Progress: This week was mostly spent gathering project information for our final design document. We also started a few paper prototypes.

Problems: We need to set up a meeting with Judy soon to verify some information for the design document asap. Many areas are still grey with regard to how we plan on approaching. We need to truly understand our audience and their need.

Plans: Two major things still remain before the end of the term. 1. We need to talk to Judy and hash out the details that are not clear for our design document. 2. We need to talk to nurses to verify the desired project platform and then

finalize our plans moving forward.

Nov 22, 2019:

Progress: Call with Judy this week resulted in a lot of very beneficial notes. We have a few plans that need to happen before the end of the term. For now, we have sent Judy the hospital information and heard back from one of the nurses.

Problems: The most recent reply from the hospital said that the nurses would be busy until the last week of school meaning we may not have a full idea of our platform for our design document. :(

Plans: We need to interview nurses, finish our design document, send the document to Judy for review and approval, and get the technical side of our project sorted out before the end of this term.

Dec 1, 2019

Progress: Made contact with a nurse and set up a date to discuss the platform!

Sent our paperwork to Judy for review and approval.

Problems: The projected date to discuss with nurses is after the final design document is due.

Waiting on Judy about documentation, likely delayed due to the holidays.

Plans: Review the documentation and make a decision for what will be in our final before discussing with nurses. We can make a few adjustments after that meeting and get it re-approved? Client says this is okay.

6.1.2 Winter

Jan 10, 2020

Progress: Our group touched base with our clients this week to go over the plan for this term. We have paper prototypes being prepared and are getting the last bit of research done before beginning the code.

Problems: none yet! Just need to plan some meetings.

Plans: Future meetings for paper prototype discussion, first draft of code, and alpha stage application are being set soon!

Jan 17, 2020

Progress: Paper prototypes are done and code has been started.

Problems: none :)

Plans: finish programming basic version of app by next Wednesday for meeting with Judy. Follow up meeting with nurses is week after (week 4).

Jan 24, 2020

Progress: A really rough alpha stage application has been made using .NET (windows only). About 90% of the mandatory functionality is finished meaning that a lot of changes will be due to additional features and usability.

Problems: we need to figure out a plan after app development to avoid any delays. How will we integrate into other systems?

Plans: meeting with nurses next Wednesday with our alpha stage app to get some user testing and feedback!

Feb 8, 2020

Progress: Alpha stage is finished and beta is being started. We need to convert to web based from desktop and some timeline stuff is being started.

Problems: Conversion from desktop to web is uncertain and we need to figure out how to host.

Plans: Add the rest of timeline stuff and have meeting with nurses after new features are added.

Feb 14, 2020

Progress: Poster rough draft is finished. The desktop app platform had been reconfirmed.

Problems: There was a delay while looking into epic integration. We can't build into epic anymore/ it seems there is no desire from them to include our app.

Plans: Add in timeline to visual interface so it is easier to see previous times added before shift changes as well as the time to reset time.

Possibly integrate a method for titration.

Feb 21, 2020

Progress: Working on timeline implementation and demo.

Problems: We are unable to integrate into Epic, which is disappointing, however it makes our app implementation easier.

Plans: meeting on Sunday to start demo prep/ future feature plans.

Feb 28, 2020

Progress: Currently working on improving data storage with new data types for saved patient time data.

Problems: I am unsure how to visually display the timeline.

Plans: After having time data figured out we will work on adding/ removing/ deleting time blocks.

Also we need to work on our design review plan for Tuesday.

Mar 6, 2020

Progress: File I/O is completely finished and now final timeline stuff needs to be addressed.

Problems: Communication has been low, we will continue to work on project and reach out at our next milestone for an update.

Plans: We need to figure out the best possible way to display the interactive timeline.

Also, will we address the titration issue?

Mar 13, 2020

Progress: Beta is almost finished, display for timeline still needs to be added. (this weekend)

Problems: Have not contacted client in awhile. We will reach out again when the beta is ready to show.

Plans: Beta video/demo and final code will be finished this weekend (before Tuesday).

6.2 Parker Okonek

6.2.1 Fall

3

Progress We have done two meetings with our client. We have a pretty good understanding of what the project will require for us to implement the math it requires and what kinds of inputs the users will be expecting the most.

Problems

I personally have been having issues figuring out the requirements document, even when working with my partner I haven't quite figured out what it is asking. It seems like all the templates that exist, including the 3 separate IEEE ones and the requirements for the document from the assignment description don't align in any meaningful way and the requirements can be about anything. I have also been having problems managing the homework of my other classes and this class and my work, none of which I can reduce my time on.

Plans

We will be getting into contact with a focus group of nurses to determine what platform they would like the program to be on, we will also determine if there are any additional features the nurses would find useful in the program and what kind of UI will best suit their needs.

4

Progress

We have arranged another meeting with our client and the doctor who originated the work our project is based on. We will meet with them on November 4th to continue discussing the details of the product and arrange for meeting with nurses.

Problems

We had some issues on the requirements document first draft. We were missing an abstract and Gantt chart. In the second draft we should have these portions of the requirements document completed for more accurate reviews.

Plans

We will have the November 4th meeting, and at that time arrange more work to do.

5

Progress

We have determined all the points that we can break our project into among the two of us for the upcoming tech review assignment, we just need to split the work and get it done.

Problems

I have had a lot of work this week in other classes that has been keeping me from working on the content for this course, this week I've had multiple large writing assignments due as well as other homework, studying, and midterms. Luckily the current assignment got postponed, but I will be in midterms until week 7, so I'll have to balance that between ensuring that I don't hurt the group project for this class.

Plans

Decide what requirements each of us will be covering for our individual assignments, then go our separate ways to get a good amount of coverage for the project overall.

6

Progress

We have communicated with our client once again with the doctor, Doctor Heyland who created the process. There seems to be confusion between him and our client on changes our client has made to the process we are replicating, however.

Problems

The working platform has not been solidified, and we haven't a chance to show paper prototypes to choose one to stick to makes the requirements somewhat vague, but this can be fixed with more communication in the upcoming weeks.

Plans

We will start initiating more conversations with our client. We still have several design considerations which need to be solidified.

7

Progress

No updates from our client yet. We will still need to reach out to her about our design decisions we would like to make on platforms and UI frameworks. Once these are laid down we will have a much easier time establishing requirements as needed for in-class projects and for the project going forward.

Problems

We spent most of the time working on the assignment trying to create a working structure for the document and trying to understand what the document was supposed to be. As such we were unable to create a complete draft for the assignment. We still need confirmation from our client about our platform and some design decisions we would like to make going forward.

Plans

Create a variety of paper prototypes to present to our client via email and have her review them and make a tenative decision over the platform or at least narrow the available platforms.

8

Progress

We have established all of the stakeholders for the project and have updated our design document to match. We have gotten all of the information we need to fill out the requirements document fully except for the platform.

Problems

The local chapter of nurses in the Samaritan hospital in Corvallis use a niche system that doesn't use the method our technology is replacing, so we will not be able to reliable survey them for their preferred platforms for the method.

Plans

We will send the requirements document to our client after finishing a draft of it for her to review, and then we will update it accordingly and work on contacting the nurses or finding another way to empirically choose our platform.

9

Progress

Judy Davidson, our client, helped us get in touch with a nearby hospital in order to further the selection of our platform. While the nurses there do not use the same method, we should still be able to determine a representative set of demographic information for nurses nationwide on the issue of computing platforms for this application.

Problems

The Good Samaritan Hospital in Corvallis uses a different method for feeding than our tool is built with, so we cannot do direct testing there regardless of if the nurses and the administration would OK the process. We may have to rely on other methods of gathering the information or check other neighboring hospitals.

Plans

We will be arranging a meeting with Sara Thomas on December 5th, over the phone if it can be done. This will help us solidify our platform and be well poised at the beginning of winter term to begin development.

6.2.2 Winter

1

Progress

We are in the process of restarting development from fall term. Currently we are creating a set of paper prototypes with the finalized requirements from the end of the last development period to aid in resolving differences in opinion we encountered between members of our client's team and the various nurses we had access to.

Problems

We still will need to clarify the UI layout and usability for our clients, based off of their inputs, our paper prototypes, and initial alpha versions. The designing of the UI to fit the specific needs of the nurses in an environment where the tool will be accessed quickly and sporadically will remain a top priority.

Plans

Our team will be meeting with our clients between the 15th and the 20th in order to discuss the first steps of development and further clarify and update our timeline of UI layout and code development.

2

Progress

We have had a meeting with our clients where we discussed and presented several different paper prototypes created by Alison and I. Alison presented prototypes with more features and abilities, while I had created prototypes that were minimized. Overall, we settled on adding the more complex features for the desktop application with some work to be left over for making them easy to access and understand, and the more simplistic version being delegated to a possible other platform.

Problems

During our research and document creation, we forgot to research tools that would be very helpful for actually creating the code. Since we are focusing on C++ for our language, the main concern is build management. There are several languages with wonderful supplied build managing and verbose errors, C++ is not one.

Plans

By the end of this week, it would be good to have our coding environment set up, our main libraries for UI installed and ready, and the first lines of code coming into git. We will need to meet up as a group to resolve these for the next step.

3

Progress

The base logic of the application is set up along with UI elements that are interact-able and create updates. The Application currently does not maintain an internal state between items and makes heavy use of C# string parsing, which works fine for the current demo version but will need to be changed in subsequent versions for security and accuracy.

Problems

Not many problems this week. Main issues concern getting a live reading of time from the OS and presenting it in the UI and setting up file IO for saving and loading settings, which the former is a greater concern.

Plans

UI improvements based off of our talks with our clients will happen this weekend. Following this we will be presented the updated and more feature-full application to a group of nurses for them to critique it and raise any concerns they see.

5

Progress

We have met with the technical team for our clients' central software access product for patient data, Epic. We learned a lot of important technical information about how the hospitals actually operate, what their current UI looks like, and many other pieces of information and are seeking more information on how to properly integrate this new software.

Problems

It seems that in order to be available to the nurses and have the product be useful we will need to pivot platform again, though we will try to keep from doing so if it looks like it is unfeasible in the development time frame.

Plans

Following the advice of our TA, we will keep the web development in its own branch and leave our minimum viable product level desktop app buildable and available for alpha release in week 6. We will still need to work with our clients, Judy and Mike, in order to coordinate to make the web app happen, but based off of the way we built the original desktop app, most of the work can be easily replicated into a web platform.

6

Progress

Most progress has been on solidifying the path for the application, it is essentially fully featured and now needs more ease of use and usability features added to continue its development.

Problems

The planned web development for the app has been cancelled, due to security and maintenance concerns for integration with EPIC. After our clients talked to their EPIC integration team it was determined that this would be unwieldy.

Plans

We will continue to develop the desktop side and add usability features for nurses to have an easier time doing other enteral feeding functions, such as titrating up to a dose of feeding rather than an immediate increase.

7

Progress

Progress has stalled as we try to re-orient on continued desktop development.

Problems

Without the planned work on a web conversion we have to implement new and more advanced features into the desktop app, but we lose all of the planned infrastructure for the web interface.

Plans

We have planned a meeting to work on the presentation materials and to add new features.

8

Progress

We have implemented new operations for file IO of saving anonymous patient data, presenting a visual representation of missed times, and have begun integrating these into the application for a cleaner internal function and for a more visible and usable UI and data storage.

Problems

We have been having problems working on the code and on the presentation due to an inordinate amount of work in other classes.

Plans

We will get the file IO, file layout, and timeline pushed into the master branch. Then we will try to restart communication with the clients, again. And the work on the presentation will resume.

9

Progress

The file IO portion of the project is now in its first usable state. This also includes the new data structure for storing anonymized patient data in memory and saving or loading it from disk. With the patient data structure complete and integrated with the UI, real work can now begin on the patient missed timeline.

Problems

Development of the file IO and timeline were both slowed by issues outside of the course or the project. The file IO was pushed onto github more delayed than I had hoped, which in turn slowed the development of the timeline, however it has not been slowed too much to stop the project development timeline.

Plans

I will work personally on code clean-up and improving the error handling of file IO. I will also ensure that it stays up to date with changes made in the patient timelines.

10

Progress

Not much progress has happened this week. It was planned to squash some bugs and integrate the timeline more closely, but I have not had enough time between other classes and work to accomplish this.

Problems

Due to the recent COVID-19 outbreak and measures being taken to combat it in Oregon State University. I lost quite a bit of my private time to work on converting my entire department at RecSports to remote workstations as part of my duties as a IT support student. As such I have not had as much time, personally, to work on aspects of the project.

Plans

In the upcoming weekend I will work more squashing bugs, making the code overall more sound, and producing the end of the year documents and video.

7 Usage Examples

Upon first launching the calculator the set-up screen will appear. This screen, shown in figure 3, has modification options for every required value of a patient for the enteral feeding process. The daily start time will be set by default to 6am, however the user can change this value at any time, and the daily start time will stick to the new value across application sessions.

The feeding type, total daily volume, and max feed rate must be set per patient, but not per application launch. The max feed rate can be manually set, but will set its value from the feeding type by default.

These options can be bypassed by loading patient information from a file.

The main window will open after the user inputs all required user data manually or via a file. This window both displays all the information needed for a nurse to administer the adjusted rate and provides interfaces for the nurse to add missed times, displayed in red on figure 4, and to adjust the previously entered values for patient information.

The patient timeline of the patient is able to be adjusted manually and can have all hours entered as missed unmarked. When missed hours are entered, the rates before the missed time are recorded, and the new adjusted rate is presented and saved.

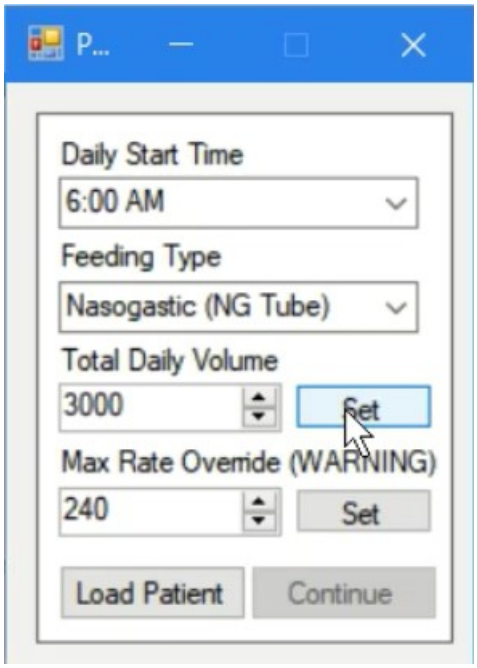

Fig. 3. Enteral Calculator Settings

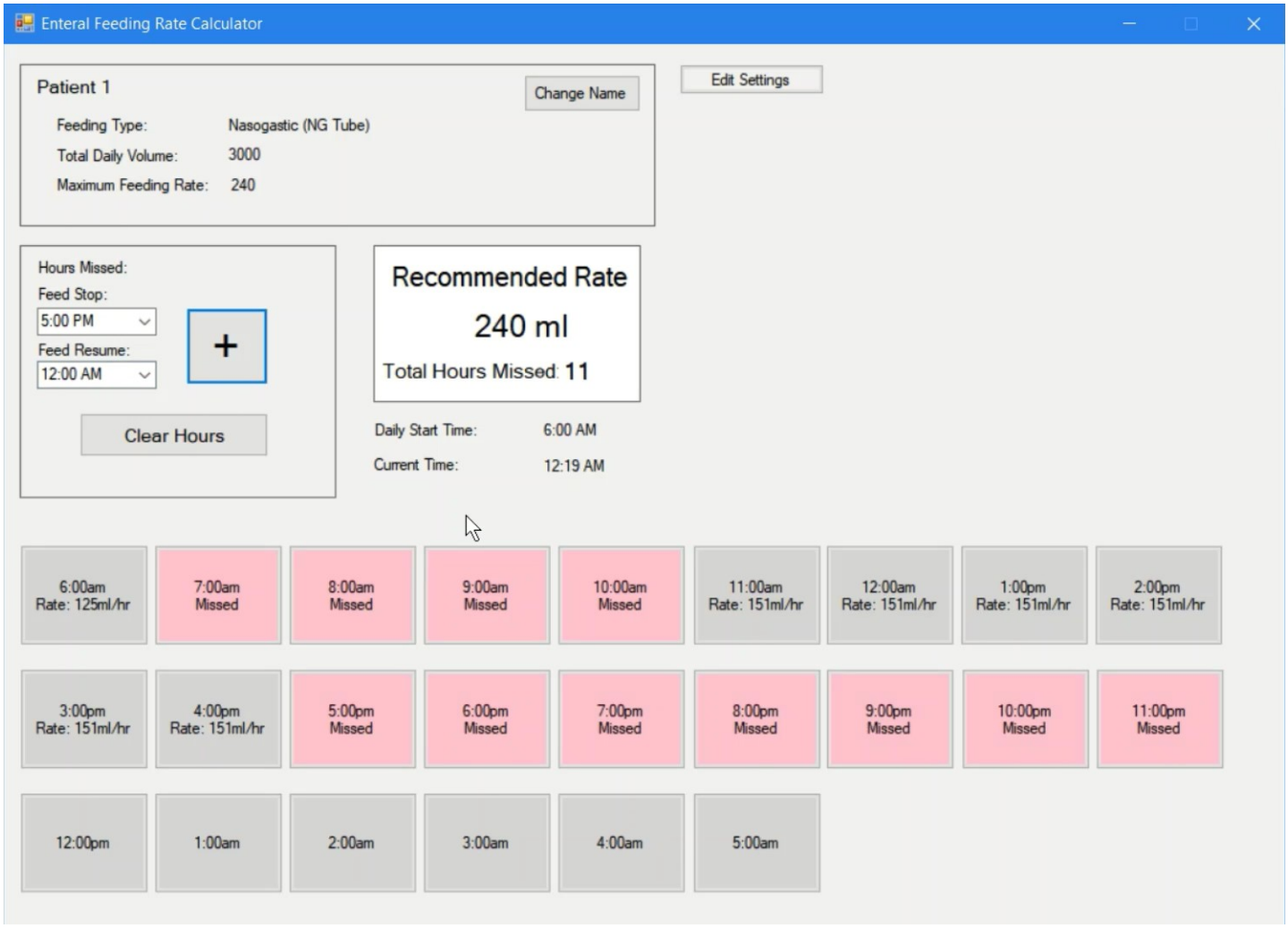

Fig. 4. Enteral Feeding Calculator in use.

8 Project Documentation

8.1 Structure

The project is broadly divided into the user-facing interface, the system-facing interface, and the internal logic. The user-facing interface displays values present in or calculated using the internal logic while allowing simple inputs and displays for a user to understand and manipulate the data. The system-facing interface reads and writes values to local storage and the system registry to allow for persistence of settings and patient data across user sessions. The internal logic calculates feed rates, manages the patient timeline, updates feed information as needed, and maintains a program state at all times, including managing memory and handling errors.

8.2 Theory of Operation

The application starts by creating a blank patient information class which can be filled with user input values or values read from a file. If the user chooses to hand enter information, they can set any information required to calculate the initial feed rate, including feed type, start time, daily volume, and max rate (though, a default for max rate is supplied by the feed type). If the user chooses to load from a file, the entire file is loaded into memory and then parsed line by line as individual fields in the patient information data structure are set, see figure 5.

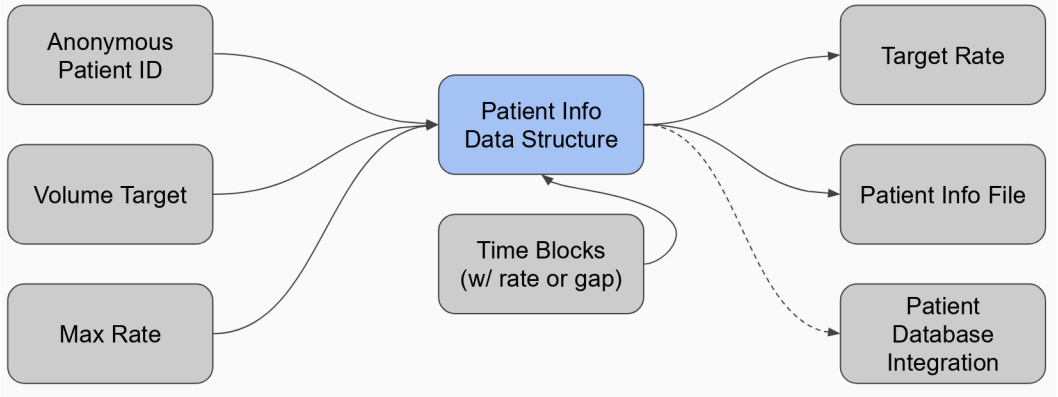

Fig. 5. Data flow using the patient information data structure.

After all patient data is available, the main window of the program will appear with the recommended feed rate, an input box for missed hours, the full 24 hour timeline of the patient, and controls for different fields of the patient's data. At this point the user is free to alter any of the fields on the patient's data and add or remove missed times as needed. The main window also has an available input to give the patient an anonymous name or ID. This ID provides an instant verification that the correct file was loaded for a patient regardless of the file's name in the case that the user needs to save multiple patients' information.

The fields for the patients data and the loading and saving options for a patient will be available in the screen toggled by the Edit Settings button.

8.3 Installation

The program is installed using the installer available on the project's Github release page. The installer comes in a zip file containing all of the files needed for installation unless the user does not have a sufficient version of .NET installed. In

the case of a user lacking .NET, the minimum required version will be downloaded automatically to the user's computer in the installation process.

The application requires Windows 7 or higher to install and run due to its use of .Net frameworks and the Windows Form App library. The program may be able to run on MacOS or Linux based devices, however this has not been tested and may fail to launch or run correctly.

After the installation process completes, the program can be run by launching the enteral.exe file. Our team was unable to get any key to sign the installer or program, so the user may need to bypass warnings regarding this on the first launch.

9 Recommended Reading

The main reading resources for this project were the Microsoft C# Reference and the Microsoft .NET Reference. The former reference is a source of examples, descriptions, and explanations of different C# concepts and components. The latter reference is a source of in depth descriptions and presentations of the various .NET constructs needed for any windows-focused C# project, especially when using the Windows Form App libraries.

For further information on the Microsoft references, see the following urls:
Microsoft C# Reference: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/
Microsoft .NET Reference: https://docs.microsoft.com/en-us/dotnet/

10 Conclusions and Reflections

10.1 Alison Jones

This was my first applied experience in App development. While we did not end up creating a phone app, the desktop app was still completely foreign to me. This was not only my first experience with the .NET Forms framework but also with C. I actually really enjoyed the language as it is very similar to C and C++ (my preferred languages) but has additional libraries for displaying stuff! There definitely a lot I can still learn about the language, however this felt like a good starting point.

Moving away from the coding side of the project, I learned about the medical application of our project, the process of enteral feeding, and the math that goes into managing patients. I learned about specific rules for medical application of new technology and what clearances are required. The security requirements and various types of architecture helped shape the decisions the team made throughout the design. I think it was just as important to understand it from the medical perspective as it was the coding perspective.

This project has taught me that working in a team can be both rewarding and difficult. It is important to establish proper communication, meet regularly, and ensure everyone is on the same page. The project had many changes throughout its course due to so many varying perspectives on what it should be. This could have been decided much earlier if communication was better int the beginning. On the positive end, group work has allowed us to split the project load and do sections that each team member feels confident about. We can work on the areas that we most understand.

As someone who procrastinates a lot, I have been trying to avoid doing so. Managing a project requires a lot of forethought. When are certain milestones to be met and are we on track? When and how often should we meet with

teammates or clients? Understanding how much time something will take can be very difficult in the programming world and it is important to know your project well in order to provide proper feedback/ guarentee to your client.

If I had to do this project again, the one thing I would push to improve is my communication and foresight. While the frequency and content of our group's meetings were sufficient, it always felt like we were bouncing between ideas and that it took awhile to settle on a solution. It would have been better to fully agree on a platform before doing research on the types of technologies we would use. It would have been better to get information and interview all sources earlier in the planning stage. This could have been avoided if we had gathered more information on the available technologies and the user need earlier on in development.

10.2 Parker Okonek

This project was a great introduction to C# and graphical applications for windows based computers. Using Windows Form App to develop the application gave me a new understanding of UI design and the development issues that are incurred when trying to present information effectively while keeping things simple for the uses and for the developers.

The target user group of nurses required careful consideration in every step of the project. The project is made for nurses with input from nurses, and every design element internal to its functionality or external to the users needed to be made with minimal possible confusion, simple ease of use, and reliability. Making the software reliable was a struggle even with the dot net framework's validation for all GUI inputs. We had new bugs surface during our Code Review which we hadn't encountered in months of development, and the bugs appeared almost instantly in their basic testing. Many of the bugs encountered were esoteric, for example the "-1AM" bug for an uninitialized time to start, even though the different times were stored in an array, and fetching the -1st value should have resulted in a failure to find a time, not the negative version of the 1st value.

Managing the times for capstone and coordinating the work even in our group of two was sometimes difficult, conflicting schedules, unplanned happenings, and unexpected git issues surrounding Visual Studio all made team development more unpredictable than need be.

If this project were to start over from the beginning, the first change would be solidifying the platform with the proper groups at the client's institution as soon as possible. The uncertainty brought from an unknown platform exacerbated the problems from lack of understanding about the hospital environment on our end and lack of understanding about software development on our client's end. The most important aspect in the early stages of the project is bridging the gap between the the two knowledge spaces and if we had done this more quickly in the beginning of the project, we would have had a smoother overall experience.

APPENDIX A
ESSENTIAL CODE LISTINGS

A.1 UI Syncing

```
public void sync_ui() {
    // Set values for every visual indicator to match the values stored in patient data
    this.DailyVolumeNumeric.Value = (decimal)this.patientData.get_volume();
    this.feedTypeCombo.SelectedIndex = (int)this.patientData.get_feed_type();
    this.MaxRateNumeric.Value = (decimal)this.patientData.get_maxrate();
```

```
    this.label1.Text = this.patientData.get_id();
    this.dailyVolumeDisplay.Text = "" + this.patientData.get_volume();
    this.FeedRateDisplay.Text = "" + this.patientData.get_maxrate();
    this.feedTypeDisplay.Text = this.feedTypeCombo.Text;
    setTimeline();
    // Update feeding rate
    display_feed_rate();
}
```

This code syncs the user-facing interface elements to the internal patient information data structure. This function is called almost every time the patient information is altered to keep the UI updated while separating the visually presented information from the data used for calculations.

A.2 Timeline Updates

```
private void setTimeline() {
    int start = 0;
    int stop = 0;
    int blockNumber = 0;
    double rate = 0;
    int time = 0;
    string timestring = "";

    for (int i = 0; i <= 23; i++)
    {
        time = RegistryEdit.check_daily_start() + i;
        if (time > 12)
        {
            time -= 12;
            if (time > 12)
            {
                time -= 12;
                timestring = time + ":00am";
            }
            else
            {
                timestring = time + ":00pm";
            }
        }
        else
        {
            timestring = time + ":00am";
```

```
        }
        rate = this.patientData.getTimeBlocks(start, stop, blockNumber, i);

        if (rate == 0)
        {
            Timeline[i].Text = timestring + "\nMissed";
        }
        else if (rate == -1)
        {
            Timeline[i].Text = timestring;
        }
        else
        {
            Timeline[i].Text = timestring + "\nRate: " + rate + "ml/hr";
        }

    }

    blockNumber++;
    stop++;
}
```

This function is used to display the patient's timeline from internal data. The values for each clickable element of the timeline must display their time (starting at the institutional start time), the rate at the time slot (if any), and the status of the feeding at that time (missing, present, future, or past).

A.3 Serializing Patient Data

```
public PatientInfo(String fileContents, DateTime timeDateReset) {
    // Windows line endings are \r\n, but the \r doesn't matter
    string[] lines = fileContents.Replace("\r", "").Split('\n');
    // 4 lines is the absolute minimum data for a patient if no miss times specified
    if (lines.Length < 5) { return; }

    // Instantiate all the variables we will need to instantiate the patient class
    // All but the patient ID could fail in parsing, so they aren't assigned yet
    string patientID = lines[0];
    FeedType feed;
    double totalVol;
    double maxRate;

    // This will read the line and check if it has any of our valid feed types
```

```
// The cases can be changed as needed to match any text
// we only have a finite amount of feed types, this switch statement *just works*
switch (lines[1]) {
    case "NG":
        feed = FeedType.NG;
        break;
    case "NJ":
        feed = FeedType.NJ;
        break;
    case "PEG":
        feed = FeedType.PEG;
        break;
    case "J":
        feed = FeedType.J;
        break;
    default:
        return;
}

// Total Volume is either parsed, or we give up
if (!double.TryParse(lines[2], out totalVol)) {
    return;
}
// Same with our maxrate
if (!double.TryParse(lines[3], out maxRate)) {
    return;
}
// We have now *hopefully* parsed all of the information we need to create thpatient info
// So let's instantiate it
this.PatientID = patientID;
this.feedingType = feed;
this.totalVolume = totalVol;
this.maxFeedRate = maxRate;
this.times = new TimeBlock[24];
this.trimmed = false;


// Now we need to read the reset time from the file in order to properly do time comparisons
DateTime startingTime;
```

```
    string dateFormat = "yyyy MM dd HH";
    if (!DateTime.TryParseExact(lines[4], dateFormat, CultureInfo.CurrentCulture, DateTimeStyles.N
        return;
    }

    // Now we can try reading the list of stuff and toss if we have to
    // Currently we also kill the file read if any time slot is invalid
    // This is for patient safety purposes
    for (int i = 5; i < lines.Length; i++) {
        //First make sure the time is within our bounds
        int hour_offset = i - 5;
        if (hour_offset > 23) {
            this.trimmed = true;
            return;
        }

        if (DateTime.Compare(startingTime.AddHours(hour_offset), timeDateReset) < 0) { this.trimme
continue; }


        string[] timeSlots = lines[i].Split(' ');
        if (timeSlots.Length != 2) {
            return;
        }

        // Let's try to parse some stuff
        double feedRate;
        Boolean missed;

        if (!double.TryParse(timeSlots[0], out feedRate) || !Boolean.TryParse(timeSlots[1], out mi
            // TODO: Report an error
            throw new System.InvalidOperationException("Could not parse time block data at line "
        }

        int index = (int)(startingTime.AddHours(hour_offset) - timeDateReset).TotalHours;

        System.Diagnostics.Debug.WriteLine("File start time: " + startingTime.AddHours(hour_offset
        System.Diagnostics.Debug.WriteLine("Writing some data to " + index);
        this.times[index] = new TimeBlock(feedRate, missed);
```

```
    }
}
```

This function is pivotal to the file input and output done by the program. A patient information class is instantiated and the various fields are set in line with readable values from the file, which is read in its entirety as a string. If any information can not be correctly found, the values are left undefined and a separate validation function can be used to check if a read was successful.

APPENDIX B

ADDITIONAL CONTENTS

No additional contents listed.

APPENDIX C

CODE REVIEWS

Category	Description	Reviewers Comment	Action taken by reviewed group
Build	Could you clone from Git and build using the README file?	It was unexpected that the README led us to a google drive, to download the project. The setup did work very quick and simple.	Moved compiled program builds to the releases section of github.
Legibility	Was the flow sane and were variable names and methods easy to follow? Does the code adhere to general guidelines and code style?	The code seems to be following the guidelines, and it is easy to follow overall. Maybe some reordering here and there.	No specific action taken.
Implementation	Is it shorter/easier/faster/-cleaner/ safer to write functionally equivalent code? Do you see useful abstractions?	The code seems repetitive, which should be fixable with loops. Fixing this will probably reduce the total amount of code.	Repetitive code is an artifact of buttons in a WPF project being references to an object rather than sequential spaces in memory. No action.
Maintainability	Are there unit tests? Should there be? Are the test covering interesting cases? Are they readable?	No unit test, I'm not sure if there should be any.	No specific action taken. Verification functions for inputs exist in framework and file io.
Requirements	Does the code fulfill the requirements?	I think the code meets the requirements	No action needed.

Other	Are there other things that stand out that can be improved?	I would change a bit of the UI, like bigger buttons, so it's easier to press on a screen. Have the same amount of time buttons per row.	Made changes to the timeline in order to make more readable.
Category	Description	Reviewers Comment	Action taken by reviewed group
Build	Could you clone from Git and build using the README file?	Yes, setup was easy using the files from the Google Docs as directed through the README.	No action needed.
Legibility	Was the flow sane and were variable names and methods easy to follow? Does the code adhere to general guidelines and code style?	Yes, it seemed to be consistent in code style and general guidelines. It could be improved by reordering some things like the timelines.	No action needed.
Implementation	Is it shorter/easier/faster/-cleaner/ safer to write functionally equivalent code? Do you see useful abstractions?	It would be faster to have some of the code in a loop to reduce repetitiveness. It is generally not a good idea to have the same code repeated in case changes are needed.	Repetitive code is an artifact of buttons in a WPF project being references to an object rather than sequential spaces in memory. No action.
Maintainability	Are there unit tests? Should there be? Are the test covering interesting cases? Are they readable?	There are no unit tests. The app seems simple enough that it could be tested simply through user tests.	No specific action taken. Verification functions for inputs exist in framework and file io.
Requirements	Does the code fulfill the requirements?	Yes.	No action needed.
Other	Are there other things that stand out that can be improved?	More guidelines on required input would be good. That, or more checking that the input is correct.	No specific action taken. Verification functions for inputs exist in framework and file io.
Category	Description	Reviewers Comment	Action taken by reviewed group

Build	Could you clone from Git and build using the README file?	It was a little confusing being directed to download all the files from Google Docs, but after I did, setup was quick and simple.	Moved compiled program builds to the releases section of github.
Legibility	Was the flow sane and were variable names and methods easy to follow? Does the code adhere to general guidelines and code style?	I'd say the code followed general guidelines and code style. It was a little hard to follow through and to figure what methods were available or being used. Some things were also kind of out of order, like having Timeline 2 followed by Timeline 4, 3, then 8 in order.	Comments have been added to many areas of the code, certain pieces of code such as registry value changes have been extracted to their own files.
Implementation	Is it shorter/easier/faster/-cleaner/ safer to write functionally equivalent code? Do you see useful abstractions?	A lot of the code seemed repetitive and could be contained in a loop. For example, with the Timelines objects in Form1.Designer.cs, only the drawing coordinates seem to be the unique element between each one that can't be done via increments in a loop. Adding each coordinate to an array and increment through that would be a solution to that. Changes like that would cut down the size significantly.	Repetitive code is an artifact of buttons in a WPF project being references to an object rather than sequential spaces in memory. No action. Additionally, the fix described here is the current code.
Maintainability	Are there unit tests? Should there be? Are the test covering interesting cases? Are they readable?	No unit tests.	No specific action taken. Verification functions for inputs exist in framework and file io.
Requirements	Does the code fulfill the requirements?	This code definitely meets the requirements.	No action needed.

Other	Are there other things that stand out that can be improved?	Making it more obvious how to use the app would make it a lot easier to use. Things like the recommended feeding rate not being on the time boxes and those only getting feeding rates when a break is added, and only before, just would be confusing as a new user with no other information.	Readme covered by nurse training programs. Bugs fixed by addition of a new window to ensure valid variable values.
Category	Description	Reviewers Comment	Action taken by reviewed group
Build	Could you clone from Git and build using the README file?	I was able to clone and launch the program.	No action needed.
Legibility	Was the flow sane and were variable names and methods easy to follow? Does the code adhere to general guidelines and code style?	There was lots of repeated code, but things were structured in a sane way with related code being grouped together and indented nicely.	Repeated code based on api restrictions, no action taken.
Implementation	Is it shorter/easier/faster/cleaner/ safer to write functionally equivalent code? Do you see useful abstractions?	There was lots of code that seemed to be able to be put into for loops / abstracted. Such as Timeline assignments, for example.	Repetitive code is an artifact of buttons in a WPF project being references to an object rather than sequential spaces in memory. No action.
Maintainability	Are there unit tests? Should there be? Are the test covering interesting cases? Are they readable?	No unit tests. There should be unit tests for different possible inputs to the program to make sure things don't break on certain inputs.	No specific action taken. Verification functions for inputs exist in framework and file io.

Requirements	Does the code fulfill the requirements?	Meets most requirements except the not crashing requirement.	Made changes to the startup routine that prevent crashes from uninitialized variables.
Other	Are there other things that stand out that can be improved?	Still some bugs they mentioned. A readme on how to use the program would be nice.	Readme covered by nurse training programs. Bugs fixed by addition of a new window to ensure valid variable values.
Category	Description	Reviewers Comment	Action taken by reviewed group
Build	Could you clone from Git and build using the README file?	Yes	No action needed.
Legibility	Was the flow sane and were variable names and methods easy to follow? Does the code adhere to general guidelines and code style?	Yes and yes	No action needed.
Implementation	Is it shorter/easier/faster/-cleaner/ safer to write functionally equivalent code? Do you see useful abstractions?	There might have been a faster way to initialize all of the time buttons, but not a high priority item.	Repetitive code is an artifact of buttons in a WPF project being references to an object rather than sequential spaces in memory. No action.
Maintainability	Are there unit tests? Should there be? Are the test covering interesting cases? Are they readable?	Yes, it's easy to test with loading/saving .txt files.	No specific action taken. Verification functions for inputs exist in framework and file io.
Requirements	Does the code fulfill the requirements?	Yes	No action needed.
Other	Are there other things that stand out that can be improved?	The -1:00AM bug when the application is first opened.	Changed the first window from the main window to a smaller pop-up which forces initialization of all variables.

D REFERENCES

[1] “How to Store Data Locally in an Android App.” Android Authority, 20 Nov. 2017, www.androidauthority.com/how-to-store-data-locally-in-android-app-717190/.

[2] “Top 10 Best Mobile App Development Frameworks in 2019–20.” By Alex Hales, hackernoon.com/top-10-best-mobile-app-development-frameworks-in-2019-612b95cf930f.

[3] Krishna12. “How to GetCurrent Time without Using Picker.” Ionic Forum, 15 Nov. 2017, forum.ionicframework.com/t/how-to-getcurrent-time-without-using-picker/112208.

[4] Zak-DevZak-Dev. “Cordova : Android Device Current Date/Time.” Stack Overflow, 1 Jan. 1968, stackoverflow.com/questions/467
android-device-current-date-time.

[5] Martin Fowler. Presentation Model. 2004. url: https://martinfowler.com/eaaDev/PresentationModel.html (visited on 11/03/2019).

[6] Naveed Khan. Introduction to Data Driven Programming. 2018. url: https://www.effective-programmer.com/2018/05/27/introdu
to-data-driven-programming/ (visited on 11/08/2019.

[7] Mike Potel. The Taligent Programming Model for C++ and Java. 1996. url: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf (visited on 11/03/2019).

[8] The Qt Company. Supported Platforms. 2019. url: https://doc.qt.io/qt-5/supported-platforms.html (visited on 11/03/2019).

[9] FDA. U. S. Food & Drug Administration. 2019. url: https://www.fda.gov/media/80958/download (visited on 11/19/2019).

[10] “How to Store Data Locally in an Android App.” Android Authority, 20 Nov. 2017, www.androidauthority.com/how-to-store-data-locally-in-android-app-717190/.

[11] “Top 10 Best Mobile App Development Frameworks in 2019–20.” By Alex Hales, hackernoon.com/top-10-best-mobile-app-development-frameworks-in-2019-612b95cf930f.

[12] Krishna12. “How to GetCurrent Time without Using Picker.” Ionic Forum, 15 Nov. 2017, forum.ionicframework.com/t/how-to-getcurrent-time-without-using-picker/112208.