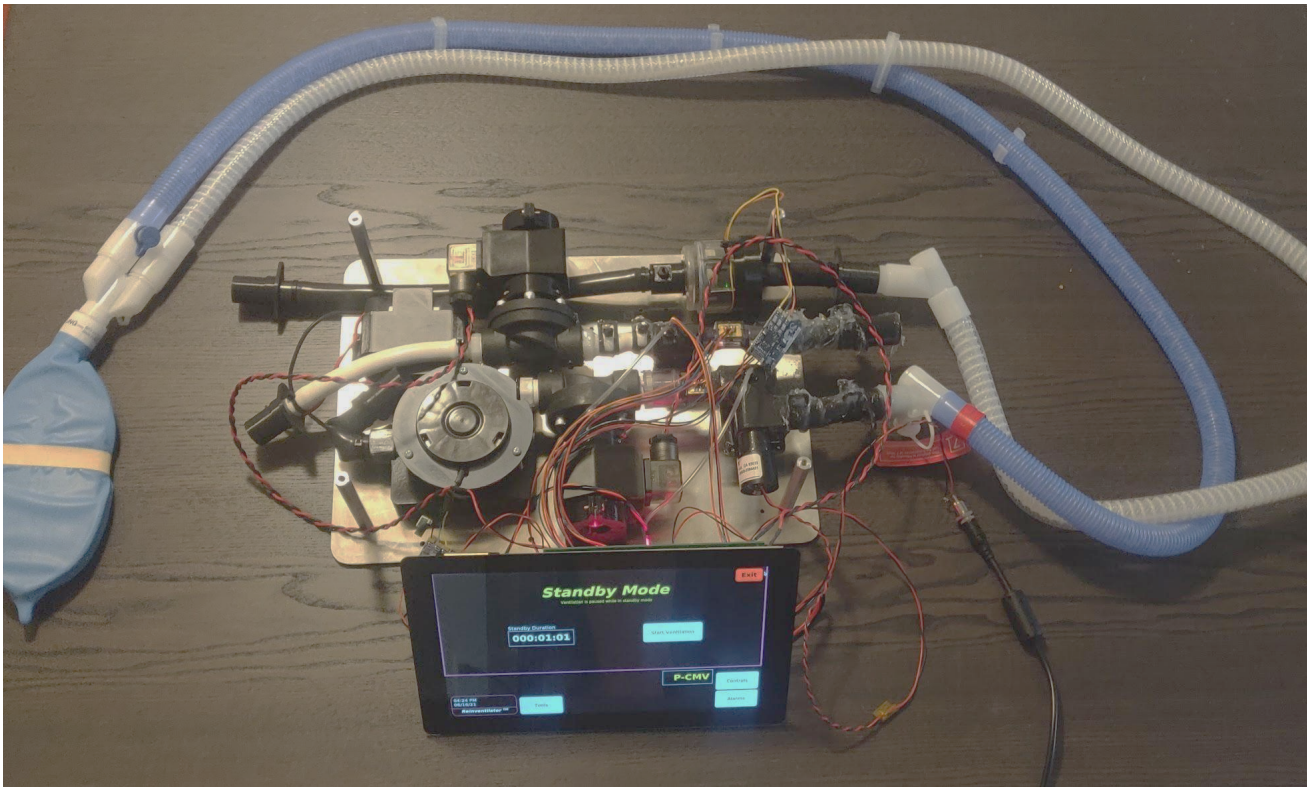
The Re-Inventilator Final Report

Team:
Kendall Brown, Jack Brown, Widney Gay, Cameron Labbe

Client:
Optical Engines

June, 2021



Introduction

These past five weeks, our team had the opportunity to work with Optical Engines, Inc. They are an optical fiber and laser company based in Colorado Springs, CO, whose CEO, Don Sipes, innovated a new remote way to design ventilators as a response to the COVID19 pandemic. The Re-Inventilator is an alternative ICU system that monitors a patient's breathing. The goal of this project was to add several functionalities to the device to improve the display, user experience, and cyclic execution systems. We accomplished this goal through the multiple new features implemented, such as the alarm system, the patient profiles, and a settings window.

The alarm system notifies a patient of a concerning value measured. This was tested heavily through UI testing on both the actual device and the Windows demo mode. A patient profile system was created to allow users the ability to edit ventilation settings and save them in memory for future use. Systems to adjust settings related to the data graphed were also implemented by our team to elegantly match with the previous existing codes.

There was some limited usability testing as our client acted as a filter of all UI elements from the simulated perspective of a user. The Raspberry Pi is reading the sensors and controlling the motor for mechanical ventilation. The GUI for the system allows users to change settings and view statistics. The vision of the product is to allow parts from machines like CPAPs to be repurposed in a cheap ICU solution to solve issues of spare or expensive health care machines.

Requirements

At the beginning of the project, we were optimistically given ten specific requirements to fulfill by the end of the session. First, the client requested that we remove the Qt designer from the Re-inventilator workflow. The Qt designer is a GUI software that generates .ui codes. These codes are then converted into python as stand-alone python files that due to the client preference and ease of coding needed to be integrated into the main programming. The workflow given by the client for modifying the GUI was creating a .ui file with Qt Designer, then using pyside2-uic to convert the .ui to a .py file which is imported as a module into the Re-Inventilator engine - the main python module. We changed this workflow by creating a .py file/module that describes all the components of a GUI that resizes itself to fit on the current screen if it is less than 1280x800 in either axis. This module was imported just like the previous workflow. No .ui files will be necessary in the future.

Additionally during the same sprint, we were tasked to develop a robust alarm system. The Re-Inventilator has settings for 10 different alarms - high and low for 5 values - and tracks the connection of at least 7 sensors. If any value goes out of bounds or a sensor disconnects, for example due to a reading failure, then the system sets off an alarm and changes the alarm text to match the current alarm. Each alarm has a different priority and the system always displays the highest priority alarm if more than one is active. The GUI also has a way to mute and clear the

alarm. The mute button mutes all active alarms, and clears out only the highest priority error, if it is allowed to be cleared.

Next, the group went on to establish a patient profile system. The device allows three patient profiles to be selected in the standby mode. Each profile should include the patient number, the gender, and the height and the height of the patient being viewed. Also, the system calculates sensible default settings for each patient profile, and once edited each profile is saved to the filesystem, as required.

Then, the group was asked to create a settings window. This window permits the user the ability to click the gear button to open a settings window. It takes up the whole screen and allows the user to modify anything customizable. It was developed in tandem with a number of other epics.

Furthermore, we customized the historic respiratory waveform displays. The initial respiratory waveform would display in white, then pink/purple, then green showing the respiratory history on a graph. This development allowed the user to modify this including the order of colors and the number of histories graphed from only the current data to up to three historical data lines. This development also allowed the user to also be able to select out of pressure, flow and volume, which data is displayed in each of the two waveform graphs.

Lastly, the client requisitioned five other requirements: the ability for users to select Volume Controlled continuous Mandatory Ventilation (V-CMV) as a mode of ventilation, to select to display the waveforms every 1-5 breath cycles - emphasizing the long term trend, have the system collect and store respiratory data, customize displayed respiratory statistics, and finally update previous tests. Unfortunately, due to our limited schedule and the fast-pace of the session the team was able to satisfy the first 5 requirements. Although, we did meet all the non-functional requirements to match the theme and display of the already existing programs and to follow all medical device standards on the market.

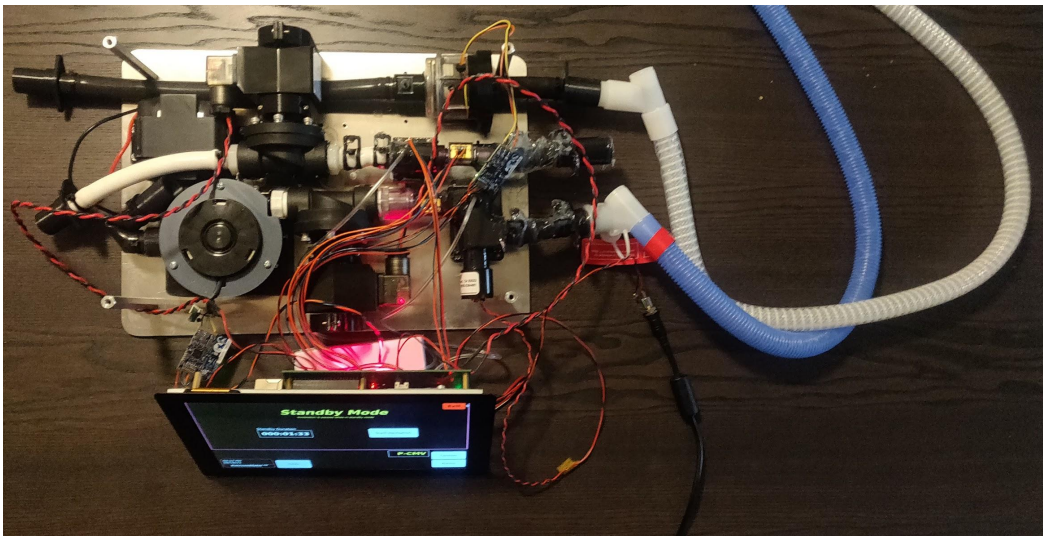


Figure 1

System Architecture

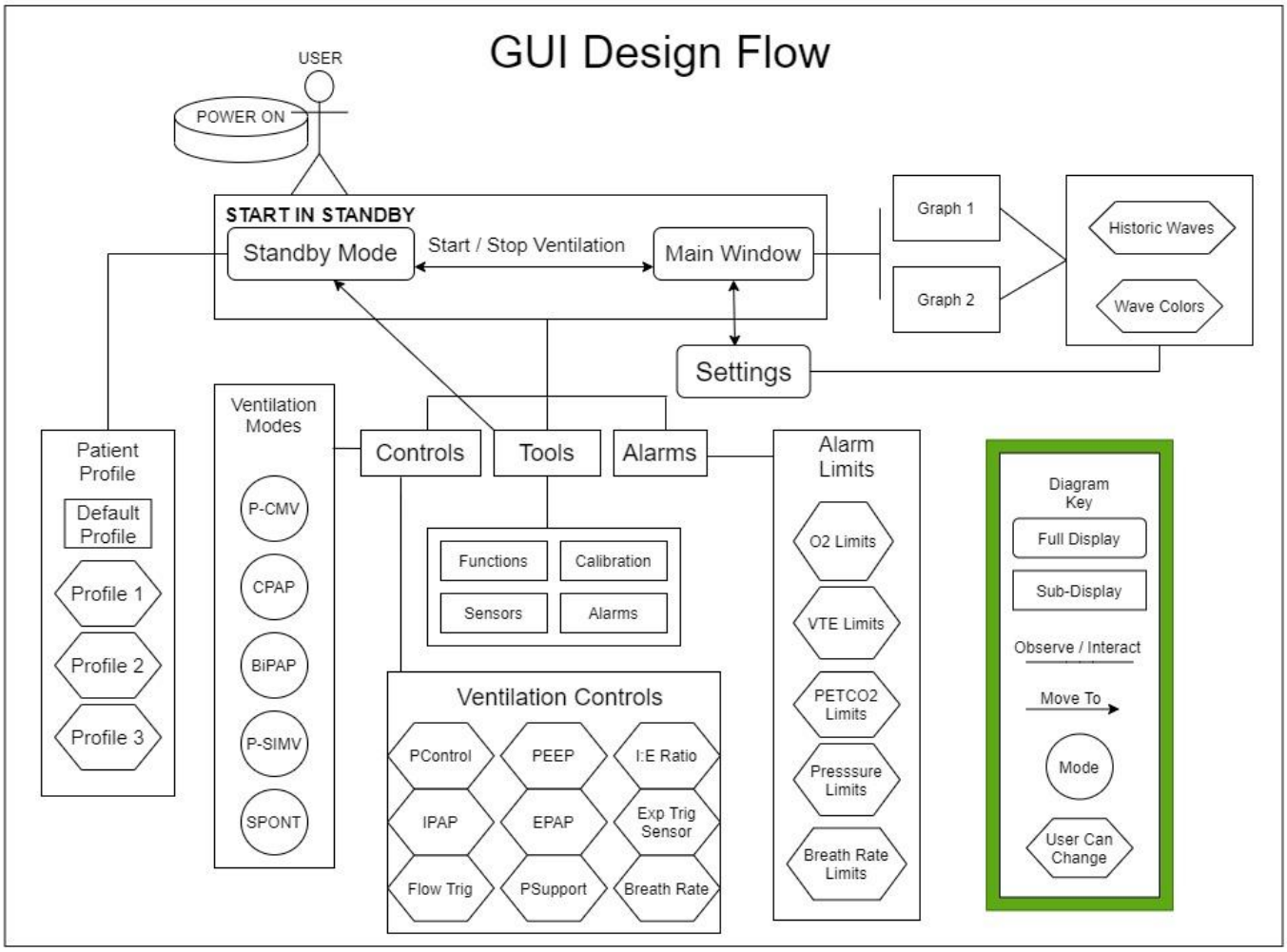


Figure 2

Figure 2 displays our updated GUI flow architecture. The greatest changes from our initial design includes adding the patient profile display and functionality, the alarms status list located in the tools menu, and widgets in the settings window to customize the waveform display. The patient profiles are displayed on the standby window on start-up. When selected, the profile editing tool displays and the user can edit profile data like height. There is also a default profile that has the predefined values for ventilation on an average human. When a profile is selected and the user selects the 'Start Ventilation' button, those profile values are used to define the mode and some parameters of ventilation. The profiles will be saved to the file, allowing users to keep their information stored and ready even after the device is shut down.

The alarms status toolbox can be reached from the 'Tools' button while in the main window or standby mode. The purpose of this toolbox is to display to the user what alarms are

going off currently, and it's especially useful if there are multiple alarms going off. Since the main window only shows the current alarm with the highest priority, it was necessary to add a tool to show the status of all alarms. The alarm names show up as green or red to denote their status.

Inside of the settings window nothing had been implemented prior to the start of the field session. We added the first few settings window functionalities that help the user change the waveform display to their liking. The user is now able to customize the colors of the waveform for each historical waveform and the current one. Also, the user can determine how many historical plots they want on top of the current plot.

Technical Design

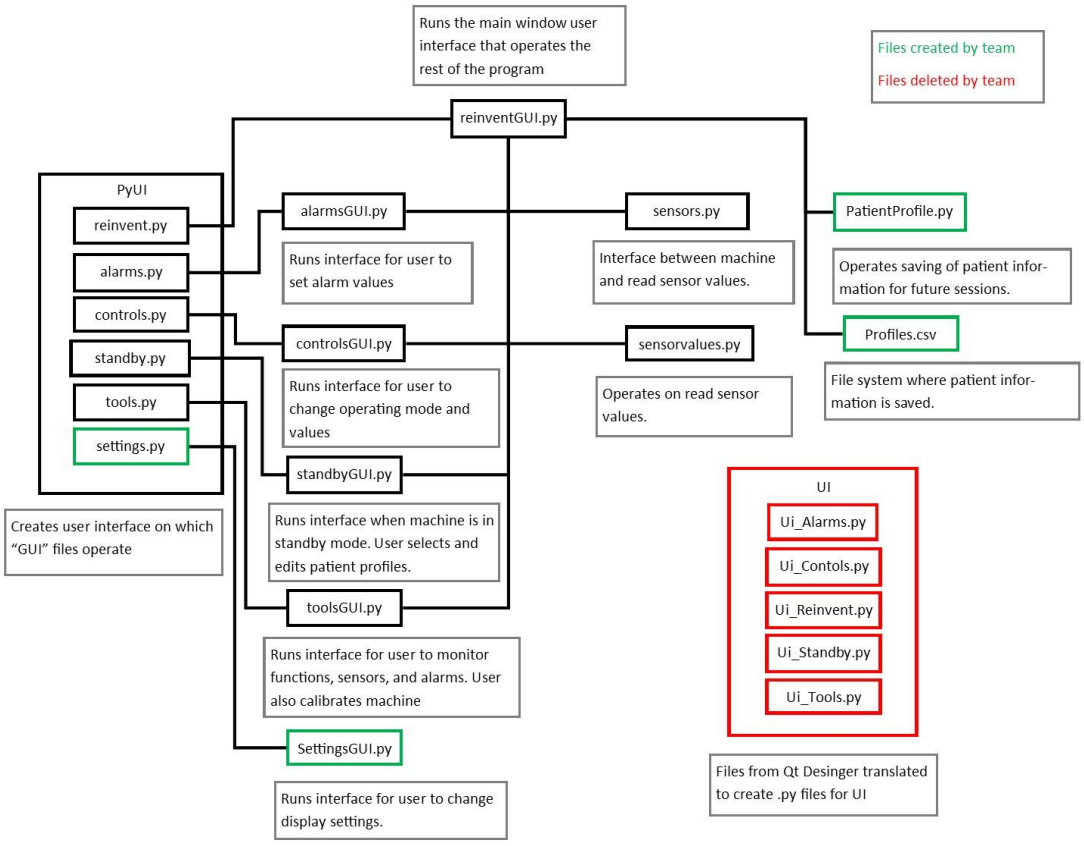


Figure 3

Figure 3 is the software architecture of the program that our team worked with. The program runs everything from the "reinventGUI.py" file, which constructs each of the GUIs

from the files stored in the “PyUI” folder. Shown in red are the .ui files that were deleted in favor of coding these files manually.

In order to program the alarm feature we worked primarily in the “reinventGUI.py” file to implement an “alarmCheck” function within the program’s regular cycle, which works alongside the “sensors.py” and “sensorvalues.py” files. “alarmCheck” not only triggers the alarm to display, but also operates the continuous alarm beep made by the machine. Because the prototype currently does not have a physical alarm installed and the program should be able to be demonstrated regardless, we implemented an alarm light alongside the beeping function that accurately shows when the beeper would function and is appropriately affected by the muting function.

To implement the use of patient profiles we worked in the “standby.py” file to create the GUI, as well as the “standbyGUI.py” and “reinventGUI.py” files to run the logic behind it. We also needed to create the “PatientProfiles.py” and “Profiles.csv” files to process and store the appropriate information from the new GUI feature.

Finally, to implement the settings window we needed to create the “settings.py” file for the new GUI component and the “SettingsGUI.py” file to operate it. The new GUI is accessed by a button in the main window and allows the user to customize the visual display of the ventilator. Currently it is used to change graph display and how many cycles of history are displayed, but many more features will be added in the future.

Quality Assurance

Our quality assurance consisted of a few critical components that worked to ensure we delivered quality software. The first tool used was manual user interface testing as the user interfaces of the various branches that we developed were compared to the master branch as a reference in order to ensure nothing was unintentionally altered. This also worked to simulate a real-world use case on the program from end-to-end to ensure proper functionality as branches were reviewed by members of the development team that were not closely associated with coding that section so if anything did not make sense to an outside viewer, it could be corrected. Finally during this process the reviewer would attempt to perform most if not all use cases in order to check for bugs.

The second component of our quality assurance was code reviews. These reviews would occur very frequently if not daily during which each team member would present the work they had been working on, and would guide other team members through the code, highlighting any issues or technical limitations. This worked to not only keep all team members up to date with the functionality of the system, but it was also a great way to get help and suggestions from other

team members and to reduce the risk that a solution to an issue will not cause conflicts down the road.

Another quality assurance tool that we implemented to our advantage during this project was pair programming. Pair programming was mostly implemented at the start of a large epic that would later branch out into smaller individual work, however pair programming was also used for some of the difficult technical sections. Doing this ensured that when work broke up into individual parts, all working members were familiar with the base they were building upon, and it saved us time where we would have been stuck working on technical aspects.

The final technique our team used to maintain quality was team reviews and approvals. When a member or pair believed a task was finished and ready to be added to the master branch of the software, they would submit a pull request. This request would then be reviewed by a team member who did not work on it and the client, who would both test the functionality within the UI and examine the code. If both parties approved the branch would be merged and features were officially implemented into the Re-Inventilator. This kept the client up to date on our project, and gave lots of opportunity to request additional features, which occurred frequently.

Results

Features Implemented**:**

- Alarm system
- Settings window
- GUI workflow simplification
- Historic waveform customization
- Alarms status toolbar
- Patient profiles

We would be remiss not to mention the various difficulties that the team encountered at the beginning of the session while programming for the device. Our team's first task was the removal of a visual editor so working with the GUI after that was rendered more difficult. Also, none of us on the team has trade medical knowledge or familiarity with a ventilator. Thus on several occasions, we had to read extensive literature to fill-in the gap. Thankfully, the client provided some well-needed and educational material to help with that gap.

For the majority of our members, this was an introduction to programming for a system that runs on a RaspberryPi. Although the system was quite easy to get a hang on, it still took some time to understand how the hardware and sensors are set up, how they are linked and read by the Pi. This also leads us to another area that posed a challenge, which was our level of expertise in Python. However, we were able to catch up pretty quickly to the industry standard of coding and create a palimpsest that implemented both the old code and our new code seamlessly.

Lastly, we did stumble when it came to understanding another programmer's code and to try to extract knowledge from what he had already established. The program was for the most part devoid of comments. This costs the team to spend more time than necessary to understand the code not only in it's aggregate but also a clearer understanding of the granular codes that made each part run. The silver lining in this is that it forced the team to be better at reviewing someone else's code and develop an expertise in python.

For example, during this process we learned that code clarity and good commenting practices can have an enormous impact on the ability of other people to assist with your code or continue working on it after your contribution is finished. We also gained a lot of familiarity with Qt as a UI tool especially when working with python. We learned a lot about agile software development, team communication, git commands, branch policies, new software development management tools such as zenHub. More importantly, we learned that one person's idea and willingness to change for the world can make a huge impact in an industry.

Future Work

The core concept behind the Re-Inventilator is software focused product development. This means that the core hardware is intentionally versatile with as many sensors and motors as is feasible. This puts a lot of power in the hands of the developers as over time even after the product is deployed, as they think of new features the versatile hardware makes it likely that they can be implemented purely via software updates. In light of this design decision, the future work to be done is very open ended as code can always be further developed. Some key features that our team would have liked the time and resources to implement are:

- Remote operation - Having the hardware in one location with the medical professional entirely remote operating the UI would be a powerful tool
- Additional sensors for safety - An example of this would be a biosensor to enable UI alerts if the device is not clean
- Usability testing with a real patient and medical professional
- Automated shutdown for extreme alarm conditions

All this being said, although it is a software defined mechanical ventilator, it would not be too difficult to add sensors to the RaspberryPi setup over time as well, as the product is far from commercial deployment. Additionally the features added by our team have been well documented by Github version control along with an Agile process addon for Github called Zenhub that contains descriptions and comments for all commits and features, so a future development team could easily continue where our team left off.