Tachometer

FINAL REPORT

SDMAY19-30
Client: Matt Post
Adviser: Matt Post
Team Members/Roles:
Jessica Bader - Chief Software Engineer / Communication Manager / Scribe
Meghna Chandrasekaran - Chief Software Engineer / Meeting Facilitator
Katrina Choong - Chief Hardware Engineer / Timeline Manager
Seth Noel - Chief Hardware Engineer
Kyle Zelnio - Project Manager
sdmay19-30@iastate.edu
Sdmay19-30.sd.ece.iastate.edu

Revised: 4/29/19 - Version 1

Table of Contents

List of Figures

List of Tables

List of Definitions

AC: Alternating Current

COM: Communication

CPRE: Computer Engineering

DC: Direct Current

EM: Electromagnetism

ETG: Electronics Technology Group

GUI: Graphical User Interface

PCB: Printed Circuit Board

SE: Software Engineering

RPM: Rotations Per Minute

0. Executive Summary

The stroboscope used to measure the rotational speed of a motor shaft for the Iowa State University Electrical Engineering 448 course currently has a high cost of repairment and equipment replacement. Our job was to design and develop a more cost effective and robust solution while continuing to meet all current lab requirements. This piece of equipment would affect students, TAs, the course professor, and the Engineering Technology Group's staff. The new piece of equipment is planned to replace the stroboscope starting Fall 2019.

Our solution uses a tachometer to automate the process of calculating the speed. We divided our product into three modules: the hardware circuit, the microcontroller, and the GUI. The interfaces between these components are well defined. We divided ourselves into two teams. The hardware team was in charge of the hardware circuit (Kyle and Katrina). The software team was in charge of the microcontroller and the GUI (Jessica and Meghna). We left one team member to work between the two groups in a systems role and help with implementation and design as needed (Seth).

We verified through system-level testing that the new product can accurately measure rotational speed from 100 to 2000 rotations per minute on both Direct Current and Alternating Current motors. We also tested our product at unit and interface levels. After accuracy testing, we did field-testing in the EE 448 lab with the students taking the course in Spring 2019. We were able to apply their feedback to our final product.

Upon evaluation of our tachometer, we found it costs 12% of the original stroboscope and takes 20% of the original average time to measure. Furthermore, our product gained approval from our client, the student users, and the EE 448 professor and TAs. The students rated our tachometer at an average of 9.5/10 in terms of usability.

1. Requirements Specification

1.1 Functional Requirements

- It will be able to perform all the functions required by the lab
- It will have 1% accuracy
- It will range from 100 to 2000 RPM
- It will have a GUI for user interaction
- It will allow the user to change the communication port to match the port chosen by the computer

1.2 Non-Functional Requirements

- It will be more cost-effective to replace than the current version
- It will be user friendly in a manner consistent with the backgrounds of EE 448 students
- It will be documented sufficiently
- It will be flexible enough to allow (small) potential future changes to the lab
- It will be resistant to breaking due to physical abuse by students
- It will have easily accessible parts for the ETG

1.3 Use-Cases

1.3.1 Student

- Change the COM port
- Check the motor RPM on the GUI
- Open the program
- Close the program
- View the 'help' manual

1.3.2 ETG Worker

- Mount the tachometer
- Load the code onto the Arduino
- Connect the tachometer, Arduino, filter, and desktop
- Test the mount stability
- Load the GUI code onto the computers

2. System Design & Development

2.1 Design Plan

We planned to create a tachometer. As a group, we decided this would compose of a hardware circuit which would output some representation of the rotational speed, a microcontroller which would convert this output into an RPM calculation, and a GUI through which the user could read the RPM. We divided into hardware (circuit) and software (microcontroller and GUI) teams to implement the design of these components.

For the development and test of our tachometer, we developed the process which can be found in *Appendix A*. For each prototype, we started by identifying the problems we were trying to solve with this prototype. Then, we brainstormed ideas to solve these problems. After this, we developed the new hardware and software prototype components. Finally, we tested the prototypes. We started with hardware and software testing, and once that was complete, we moved on to system level testing. At this point, if our final evaluation test criteria had been satisfied, we were done. If not, we created a new prototype.

2.2 Design Parameters

2.2.1 Design Objectives

For our design, we had several objectives we were focusing on meeting. The first was to meet all requirements. It was important to both our client/adviser and ourselves that we completed our project with full success.

Our next objective was to gain user approval from every user group. We determined that the users of our project would include our client, the other workers in the ETG, the EE 448 students, TAs, and professor.

Our third objective was to implement field testing to fully evaluate the success of our product. Luckily, the lab in which the tachometer is used is at the end of the semester. This made the timing of the lab ideal for field testing. Field testing would allow us to evaluate our requirements, evaluate the completeness of our documentation, evaluate user approval, and receive feedback.

2.2.2 System Constraints

Assumptions:

- The tachometer will be used by at least two students at a time
- The tachometer will be used in the EE 448 Motors Lab
- The lab will not be drastically changed in the near future
- The students will refrain from touching the tool (as they have no need)
- The lab computers run on Windows, and will not change in the near future

- The final design will be mounted above the motor in the lab and be stationary
- The motor for the lab will not change in the near future
- The Arduino can be powered by the computer

Limitations:

- The limit of RPMs should not be below 100 RPM or exceed 2000 RPM (the values used in the lab)
- The size of the tool should be no larger than the motor it is evaluating (as determined by the motors in Coover 1102)
- The cost to produce the end product should not exceed $500 for each tachometer (the cost of the previous stroboscope)

2.2.3 Design Trade-offs

How to handle excess EM

We noticed that the AC motor emitted EM. Because our Hall effect sensor reads the rotational speed of the motor using the change in magnetic field induced by key in the shaft as it moves momentarily away each rotation, this excess EM was destroying the validity of our hardware output. Furthermore, it was peaking at about 10V, while the Arduino is rated for no more than 5 V and would therefore have the potential to destroy the microcontroller. The pro/con list can be found in *Table 2.2.1: Optical Encoder vs. Low-Pass Filter*. Ultimately, we decided the low-pass filter would best suit our needs. This decision was largely driven by the cost; creating a filter circuit cost very little for the components, while we could not find any optical encoders which were within the acceptable price range outlined by our client.

Optical Encoder		**Low-Pass Filter**	
Pros	Cons	Pros	Cons
- Easy/fast to integrate into current design - Less components in the overall design	- Expensive - Would possibly require rewriting software	- Inexpensive - Can use current software	- Takes time to design - Have to order PCB -Solder on PCB

Table 2.2.3.1: Optical Encoder vs. Low-Pass Filter

Microcontroller

At the beginning of the second semester, we decided to move from a stroboscope design to a tachometer design. When we switched over, we needed to decide if we wanted to keep the Tiva board as our microcontroller or change to an Arduino. The full pro/con list can be found in *Table 2.2.2: Tiva board vs. Arduino*. Ultimately, we decided to switch to the Arduino. This decision was made because we could not recycle any of our previous code, so we would need to write the software from scratch regardless. Even though we had no experience with Arduino, it is much simpler and therefore we believed we would be able to write the Arduino software faster, given the limited time frame.

Tiva Board		**Arduino**	
Pros	Cons	Pros	Cons
- Already have experience and code - Client has easy access to Tiva boards	- Complicated to write code for - Not much time to write the code - None of the code is reusable, so it needs to be rewritten anyway	- Easier IDE to use - More online resources - Can write faster - Still inexpensive	- Would have to order components - No previous experience

Table 2.2.3.2: Tiva Board vs. Arduino

GUI Software

Finally, we needed to choose the software language for our GUI. Our advisor suggested using Python, so we decided to evaluate whether Python would be suitable for our needs or if we wanted to use a different language, such as Java. Based on the factors discussed in *Table 2.2.3: Python vs. Java*, we decided Python would be sufficient because it is easy to learn, and we had a resource for Python from our advisor. If we chose Java, we would still need to learn how to complete all the functions in the new language.

Python		Java	
Pros	Cons	Pros	Cons
- Easy to use language - Good GUI class - Provided with example code from a different project	- No previous experience - Bad documentation	- Experience with Java - Experience using Java GUI classes	- IDE with which we have experience is for mobile applications - Will need to learn some functions still - No human resources

Table 2.2.3.3: Python vs. Java

2.3 Solution Overview

2.3.1 Architectural Diagram

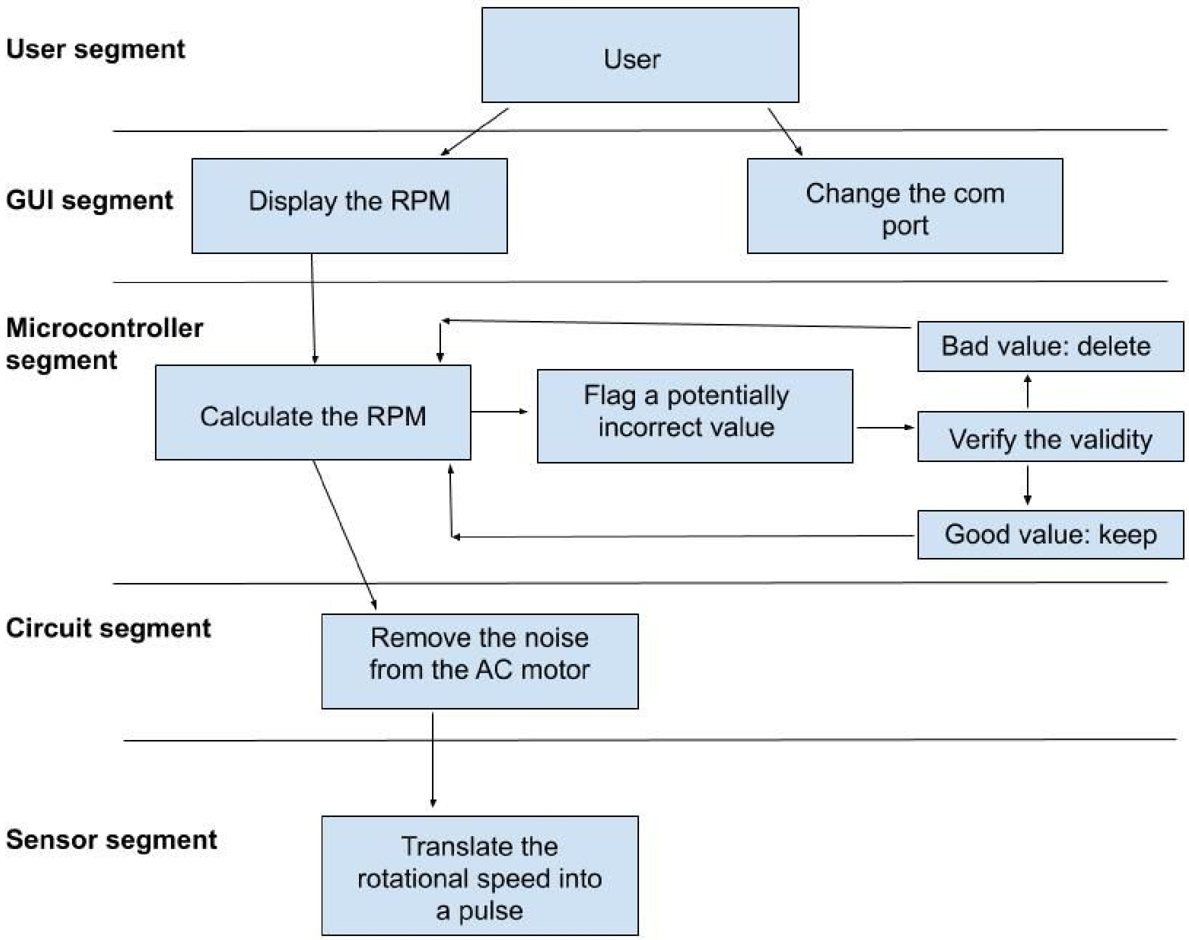


Figure 2.3.1: Architectural Diagram

2.3.2 Design Block Diagram

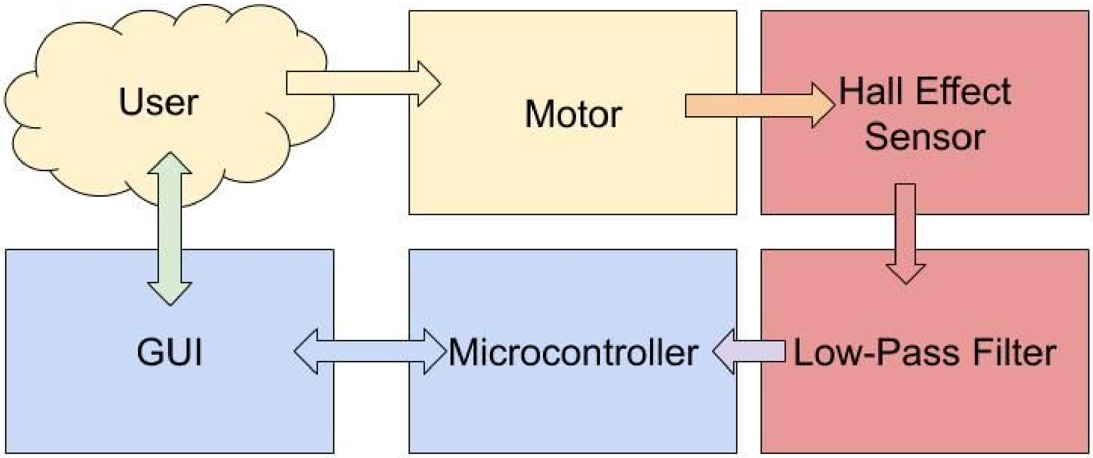


Figure 2.3.2: Design Block Diagram

2.4 Constraints, Modules, & Interfaces

2.4.1 Constraints

GUI

For our solution, we were required to use a GUI and restrict user access to the GUI.

Price

We needed to reduce the price of the product compared to the original stroboscope, so we were constrained by money. The price of our components was always one of the most important factors when choosing parts.

Motor

The motor for the lab was already designed; while we could make small adjustments to the motor (such as adding a key, adding mounting to the motor mount, etc) the final product needed to work with this motor.

Lab

The final solution needed to be designed to fulfill the specific labs it was being used in. No modifications could be made to these lab manuals.

2.4.2 Modules

Module 1: The GUI

The first portion of the software component is the GUI. The user will be able to change the COM port the computer is using to communicate with the microcontroller. From the GUI, they will be able to view the most recent RPM calculation from the microcontroller. The GUI will be programmed in Python using Tkinter, which is a Python interface to the Tk GUI toolkit. However, the Arduino board will be programmed through the Arduino IDE. This is where we will open a line of communication between the GUI and the microcontroller through the COM port. The GUI will retrieve the output from the Arduino.

Module 2: The Microcontroller

The microcontroller also belongs to the software component. It will be an Arduino programmed with the Arduino IDE. The microcontroller will send the most up to date calculation of RPM to the GUI. It will also take input from the hardware circuit in the form of a pulse. The microcontroller will utilize an interrupt on the input pin to count the number of rotations each second. It will use this to calculate the RPM. To help it keep steady, it will always average the last ten RPM calculations. Furthermore, the system may experience some faulty input values which do not accurately represent the rotational speed of the motor. To provide some protection against these affecting the calculation, the microcontroller will be able to identify and remove some input values which are deemed ‘faulty’. It will determine that a value is faulty if it is more than 20% different than both the measurement before and the measurement after. In the case, where the speed of the motor is changing, the microcontroller will be removing values once the motor speed remains constant.

Module 3: The Hardware Circuit

The hardware circuit takes RPM readings from a Hall effect sensor attached to the motor over the key. Each time the motor shaft rotates, the key will pass the sensor and induce a pulse. Specifically for the AC motor, EM waves are emitted which creates noise in the output waveform. A low-pass filter is implicated to mitigate unwanted noise created by the EM waves to the output of the Hall effect sensor. The final output from the hardware circuit, the PCB, to the microcontroller is the output of this filter. This means only the correct voltage changes are passed on.

2.4.3 Interfaces

Interface 1: Hardware Circuit and Microcontroller

The hardware circuit will be connected to the microcontroller in three places. First, it will use input pin D2 as the output of the circuit. This will be connected to an interrupt on the microcontroller. Next, it will use the 5V output from the microcontroller to power the Hall effect sensor. Finally, it will use the ground pin from the microcontroller to ground the whole circuit. The hardware circuit will output a pulse with an amplitude of more than 4V and less than 5V and an offset of 2-2.5V. This pulse will be clean and consistent. See *Figure 4.3.1: Hardware Output Waveform* for a visual example.

Interface 2: GUI and Microcontroller

The GUI will communicate with the microcontroller through a serial port. The GUI will open this port. The user will need to set the COM port in the GUI to match the COM port used by the computer to communicate with the Arduino (this can be found in Device Manager). The GUI will receive the RPM calculation from the microcontroller once the COM port has been sent. It will display this RPM for the user. The GUI will be in charge of closing the COM port when it is finished.

The microcontroller will start calculations upon startup. When it is on, it will output the most up to date RPM calculation every second. The microcontroller will be powered by the computer via the USB type-A to USB type-B mini cord.

Interface 3: User to GUI

The user will only use the GUI to set the COM port. They will be able to modify the COM port used by the program to match the COM port already used by the computer (which can be found in Device Manager). When the microcontroller is running, they will be able to see the most up to date RPM calculation every second.

3. Implementation

3.1 Implementation

3.1.1 Diagram

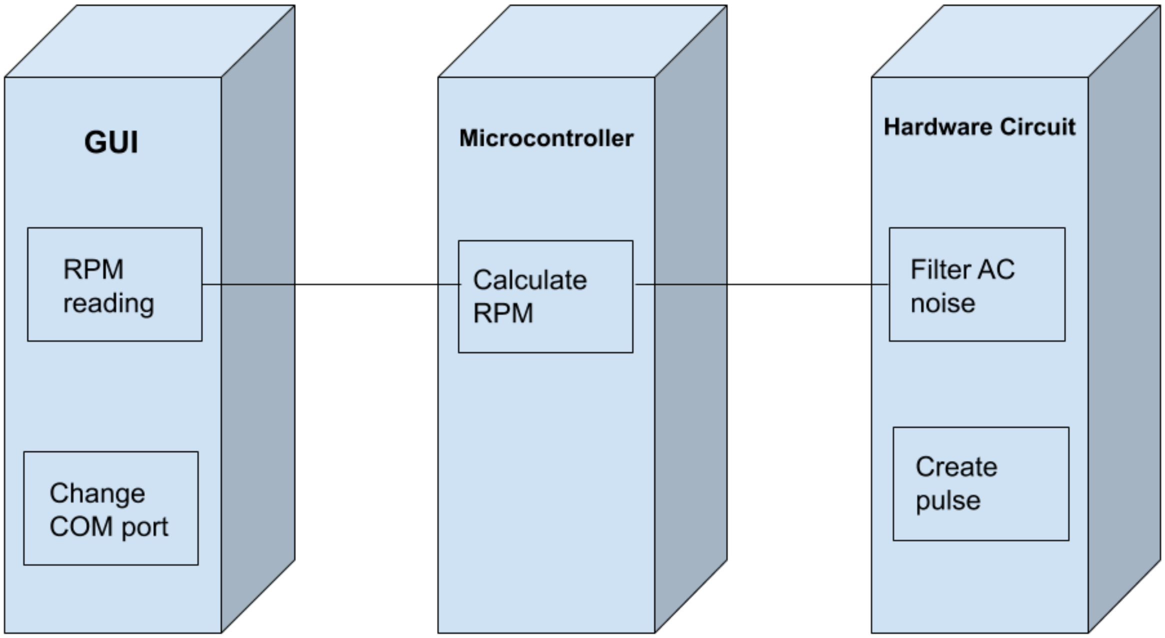


Figure 3.1.1: Implementation Diagram

3.1.2 Microcontroller - Technologies, Software, and Rationale

<u>Microcontroller: Arduino nano</u>

We picked the Arduino for our microcontroller due to its low price. Affordability is one of the main drivers of our project, as the project revolves around creating a more cost-efficient tool than the stroboscope currently being used. Another factor was the well-developed IDE for programming the Arduino. This allowed us to write the code reasonably quickly, which was important because changes to our project required us to redo all our code in the second semester. After verifying the nano (the smallest and least expensive Arduino) had sufficient processing power to run at our top speeds, we decided it was a good fit.

<u>Language: Arduino</u>

Once we had chosen the microcontroller, this was the required language. The Arduino IDE was also a reasonable choice because we were able to download it for free

IDE: Arduino

Finally, the IDE was easy to choose after picking our language. The Arduino IDE is open source, easy to use, and the most recommended IDE online.

Repository: Git

Due to our need to have multiple team members develop the software as well as use the software, we knew we needed a repository. We chose Git because it is provided through ISU. Furthermore, we already knew how to use it due to previous courses. Finally, Git was the recommended software repository for Senior Design I.

3.1.3 GUI - Technologies, Software, and Rationale

Language: Python

The first factor which drove us to choose Python was it was the GUI language recommended by our faculty adviser. His previous Senior Design group had created a GUI with Python. Therefore, we were able to use their work as a template to start our own. Furthermore, we found through research that Python was simple, which meant we could pick it up quickly on our own. It also has many online resources.

IDE: PyCharm

Based on our research, PyCharm is commonly considered the best IDE for Python. It is also open source, so we did not need to pay to use it. Furthermore, one of our group members had experience with PyCharm from an internship.

Libraries: tkinter, serial, PyInstaller

The decisions to use the tkinter and serial libraries came out of our Python research. Tkinter is a well-documented library commonly used to create GUIs in Python. It also has many online tutorials. Serial is used to open the serial port to communicate with the Arduino. We were able to find this suggestion in a tutorial about communicating between a Python GUI and an Arduino. Upon research about how to create an executable from a Python program, we found that a recent version of Python changed much about the language. Hence, many programs to create executables do not work with the current versions. We chose PyInstaller because it was simple to use, well recommended online, and it worked with the version of Python we are using.

Repository: Git

(See section *3.1.2 Repository: Git*) In addition, we decided it was ideal to use the same repository to save all GUI code as our microcontroller code.

3.1.4 Circuit - Technologies, Software, and Rationale

Filter: low-pass

We chose the low-pass filter to decrease the extra EM emitted from the AC motor mainly due to the cost. We were not able to find an optical encoder to replace the Hall effect sensor that was within the price range of our client. The low pass filter worked because the noise was isolated to a lower frequency than our pulsing output from the sensor. It was also ideal because it could be implemented to the system without requiring any major changes. Finally, we could build the circuits from components easily accessible by ETG, and it also lowered the costs and removed shipping/arrival time. This was fitting because we were close to our in-class testing demonstration when we found this issue.

Sensor: Hall effect

We researched various instruments that could output values based on rotational speed. Originally, we thought of using an encoder, but there was no encoder that was non-contact. Our reason for wanting a non-contact device was to prevent wear on our product as much as possible in order to reduce costs. Further into our research, we discovered active and passive speed sensors. Active sensors require digital output direct to a controller whereas passive sensors do not need a digital output but rely on permanent magnets, which can wear out over time.

At first, we ordered a Hall effect sensor as a test device to receive a signal from the motor’s shaft and output the data to the arduino because it was non contact and easier to work with. In the end, we deemed the Hall effect sensor, compatible with our circuit and code.

CAD: SolidWorks, MultiSim

SolidWorks is a 3D modeling software provided by Iowa State and allowed us to create virtual designs of the mount that we can then 3D print. Once printed, we were able to test the fit and stability then quickly make changes and print again. The main reason we chose this over other modeling software was it was already provided by ISU.

MultiSim is a circuit board designer and simulator also provided by Iowa State. The software let us place components on a PCB and get a board printed by a third-party for a professional implementation of the filter we designed for the Hall effect sensor. As the time, delay between creation and shipping from the third-party provider was around two weeks, we used proto-board to get our testing done because it was also provided by ISU.

Mount: 3D printer

First, we chose to 3D print the mount for repeatability. At some point the parts to the tachometer may need to be replaced or repaired, in which case an ETG worker will need to make a new mount. 3D printing is an easy, fast, and readily available option.

3.2 Standards and Best Practices

In this project, there are several IEEE standards which we will be following to ensure our project is high quality, ethical, and comparable to other products:

1. IEEE Standard for Software and System Test Documentation [1]

This standard creates a process for acquiring data, creating documentation, and maintaining documentation. Our product is to be used in a lab for years after we have left ISU. Therefore, the documentation of our product is essential for the maintainability of the product. By following this standard, we will ensure that our documentation will allow future users at all levels to successfully use the product. We will also ensure that upgrades and maintenance can be made on the system so it will be usable over a long period, even if the lab needs are slightly modified.

2. IEEE Standard for System, Software, and Hardware Verification and Validation [2]

   This standard defines the verification and validation process used to review a product to determine whether it satisfies the requirements and the user's needs. We will follow this standard to verify that our system meets the requirements that we have defined. By using this method to prove we have satisfied our requirements, we will be able to evaluate whether we have met our requirements and prove to our client this is the case.

3. IEEE Guide for Selecting and Using Reliability Predictions Based on IEEE 1413 [3]

   This standard outlines factors which can be used to determine the reliability of a product, defines what information should be included in the reliability report, and identifies acceptable ways to collect data regarding product reliability. One of our most important functional requirements is that the tachometer can operate with a 1% reliability from 100 RPM to 2000 RPM, compared to the original stroboscope that the lab used. Therefore, we will need to evaluate how reliable our tachometer is across this range. We will use this standard to determine reliability factors, compile information for our reliability report, and develop a method to collect data. Following this standard will best evaluate whether our product meets this criterion and will give substance to our final report for our client.

4. Peer Code Reviews (Best Practice)

   To hold our code to the highest standard, we used a peer code review. We had three SE/CPRE students (Francisco Arreola, Kristina Robinson, and Kyle Zelnio) who had not helped write the code, review it to fix any deficiencies they could find. They also ensured the code was well commented, easy to understand, efficient/well written, and error-free. Peer-reviews are often used in the workplace because they help reduce errors and the amount of time spent debugging. The full set of feedback can be seen in *Appendix B*. From these peer reviews, we received feedback which we were able to use to improve our final product.

4. Testing, Validation, & Evaluation

4.1 Test Plan

Our plan for testing includes four parts. The first part is the unit testing, at which point we verify that each of the three modules (GUI, microcontroller, and hardware circuit) perform their functions as required. Next, we will specifically test the interfaces between these modules to ensure they are interacting as we have specified. After this, we will test the system as a whole to ensure it follows all functional and nonfunctional requirements. Finally, we begin field testing by using our tachometer in the EE 448 lab with real students so we can receive feedback from the users.

All of our tests are manual. We decided to do this because we did not have applicable use cases with automatic testing for the full tachometer. Most of our testing included user feedback or hardware output, which further limited us in terms of automatic tests.

Our tests covered 100% of our functional and nonfunctional requirements, with some extra tests for units and interfaces. We knew that in industry, it is important to be able to compare the requirements to their tests. Not only did we write tests for each requirement, but the tests define which requirement they are verifying.

4.2 Unit Testing

Functional Requirement #4: It will have a GUI for user interaction.

Test Case:

For this requirement, we want to test and make sure the GUI works.

Test Steps:

1. Open the GUI
2. Ensure all buttons perform their intended function
3. Connect the GUI to the microcontroller
4. Ensure that “0 RPM” is being output

Expected Results:

We expect the GUI to be functional. It should be able to display the RPM and change the communication port.

Final Results:

As of 4/24/19, our GUI is fully functional.

Requirement: The Hall effect sensor will create a clear pulse from where it is mounted.

Test Case:

For this requirement, we want to ensure the Hall effect sensor is mounted correctly and capable of outputting a pulse as described in our interface specifications.

Test Steps:

1. Correctly mount the Hall effect sensor on the motor station
2. Connect the sensor to an oscilloscope to check the pulse
3. Start the motor and set it to a static RPM value
4. Check if the reading on the oscilloscope outputs a clean, clear, and consistent pulse

Expected Results:

We expect the Hall effect sensor to output a clear pulse reading when the motor is running.

Final Results:

As of 4/24/19, our Hall effect sensor outputs a clear pulse for the RPM being read.

4.3 Interface Testing

Requirement: The hardware circuit shall output a pulse which will be clean, consistent, and accurate within 0.5% to the rotational speed of the motor in hertz from 100 RPM to 2000 RPM.

Test Case:

For this requirement, we wanted to ensure the output of the hardware would be readable by the software components.

Test Steps:

1. Set up the hardware part of the circuit, including the Hall effect sensor and low-pass circuit
2. Attach an oscilloscope to see the output of the low-pass circuit
3. Start the motor at 100 RPM
4. Evaluate the output pulse to ensure it is clean and consistent
5. Measure the rotational speed of the motor with a stroboscope. Measure the speed of the output wave in hertz with the oscilloscope. Ensure these

match with the equation $f = RPM/60$ to within 0.5%

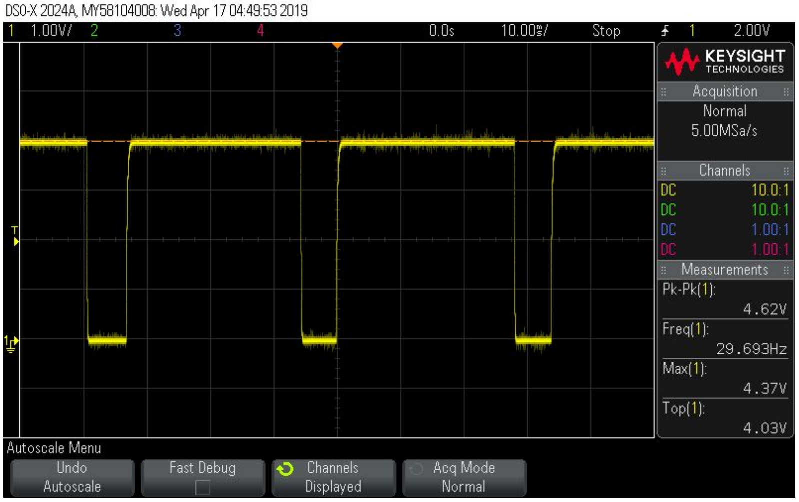
6. Repeat this process for every increment of 100 RPM until 2000 RPM
7. Repeat for both the AC and DC motors

Expected Results:

We expect the output wave to be clear and consistent. We also expect it to be accurate with the frequency. These should be true from 100 to 2000 RPM.

Final Results:

We found that the output of the hardware was clean, consistent, and accurate to the rotational speed of the motor as measured by the stroboscope across the range from 100 RPM to 2000 RPM. *Figure 4.3.1* shows an example of the output of the hardware as read by the oscilloscope. The stroboscope reading at this value was 1787 RPM, which we can see matches the frequency of 29.693 ($1787/60 = 29.783$) with 99.70% accuracy.


DSO-X 2024A, MY58104008: Wed Apr 17 04:49:53 2019

1 1.00V/ 2 3 4 0.0s 10.00ms/ Stop 1 2.00V

Acquisition: Normal, 5.00MSa/s

Channels: DC 10.0:1; DC 10.0:1; DC 1.00:1; DC 1.00:1

Measurements:
Pk-Pk(1): 4.62V
Freq(1): 29.693Hz
Max(1): 4.37V
Top(1): 4.03V

Figure 4.3.1: Hardware Output Waveform

Requirement: The software should be able to calculate RPM with 0.5% accuracy from a clean, consistent output pulse from 100 RPM to 2000 RPM.

Test Case:

For this requirement, we wanted to ensure the software was accurately calculating the RPM if it was given a near-perfect pulse.

Test Steps:

1. Set up the GUI and Arduino
2. Set up the signal generator as the input to the Arduino
3. Create a pulse with amplitude of 4.7V, an offset of 2.35V, and a duty of 80%

4. Use the equation $f = RPM/60$ to ensure the RPM calculation displayed on the GUI is within 0.5% of the frequency of the pulse
5. Repeat this from 1.67 Hz to 33.3 Hz at intervals of 1.67 Hz

Expected Results:

We expect to find that the software can accurately calculate the RPM if given a near-perfect pulse.

Final Results:

We found that every measurement was 99.5% accurate or higher, which was within our 0.5%. The table with complete results can be found in *Appendix C: Software Accuracy Testing*.

4.4 System Integration Testing

Functional Requirement #1: It will be able to perform all the functions required by the lab.

Test Case:

Based on [4, 5], we were able to determine that the lab will require students to measure the speed of a motor rotating at speeds ranging from 234 RPM to 1804 RPM. Based on these values, we were able to determine that a range from 100 RPM to 2000 RPM will fulfill all needs of the lab.

Test Steps:

1. Complete a prototype for the tachometer design
2. Test out this prototype in the EE 448 Lab by allowing the students and TAs to use it
3. Get feedback from the students and TAs
4. Use their feedback to make improvements to our tachometer prototype design

Expected Results:

We expect our tachometer design to fit all the needs of the lab after our final design. The feedback we will be getting will hopefully be based on simple technical issues.

Final Results:

As of 4/26/19, the tachometer was used to complete the lab. No use of the original stroboscope was required. The following feedback was taken to improve the functionality:

- The RPM can only read in multiples of six. It would be preferred if it could read in multiples of one.
- It takes about 10 seconds to get a correct reading once the motor has stopped changing speed. It would be preferred if this happened faster.
- When the GUI first connects to the microcontroller, several values are output at once. It would be preferred if this did not happen.
- Add a label to the RPM.

Functional Requirement #2 & #3: It will have an accuracy of 1%, and it will range from 100 to 2000 RPM.

Test Case:

For this requirement, we want to make sure the tachometer is accurate within the required range of 100 to 2,000 RPM with an accuracy within 1%

Test Steps:

1. Rotate the motor at 100 RPM
2. Measure the speed with our tachometer
3. Measure the speed with the original stroboscope
4. Determine if the measurement is within 1% of the original strobe because it has 0.005% accuracy
5. Repeat the first 4 steps at increments of 100 RPM up to 2,000 RPM

Expected Results:

The rotational speed we measure using our tachometer should be 1% accurate in comparison to the rotational speed measured from the original stroboscope.

Final Results:

As of 4/15/19, the system can range from 100 to 2000 RPM. We have found that our accuracy is within 1% for higher RPM values ranging from 300 RPM to 2000 RPM. For values below 300 RPM, the accuracy ranges from 1% to 4%. We have decided this is acceptable because the system is very difficult to measure with the stroboscope at this range. The complete results can be found in *Appendix D.*

Non-Functional Requirement #4: It will be flexible enough to allow potential future changes to the lab.

Test Case:

For this requirement, we want to determine that we have programmed our software to allow for future updates.

Test Steps:

1. All values will be stored in variables (nothing will be hard coded)
2. All code will be commented to explain its functionality
3. A review will take place at the end to ensure the first two steps have been completed

Expected Results:

We expect that upon review, our code will follow these standards 100% of the time.

Final Results:

A code review on 4/26/18 has determined we are following these standards 100% of the time.

Non-Functional Requirement #6: It will have easily accessible parts for the ETG.

Test Case:

For this requirement, we want to make certain that the ETG will be able to order all parts for the tachometer easily and through the processes they are already using.

Test Steps:

1. Our first choice for parts shall always be parts which the ETG is already ordering for another course, purpose, or has in stock
2. If there are no parts already available through the ETG, we will choose parts from the suppliers the ETG most often uses
3. We shall only order parts from new places when the first two criteria cannot be met
4. A review will take place at the end to ensure we followed these criteria

Expected Results:

We expect that upon review, we will find we have followed these steps 100% of the time when choosing components for our project.

Final Results:

A review of our current parts as of 4/24/19 has determined that we are following these standards 100% of the time.

4.5 User-Level Testing

Non-Functional Requirement #2: It will have a user-friendly GUI

Test Case:

For this requirement, we want to determine that the GUI is easily usable with the background level of our student users.

Test Steps:

1. Install our fully-functioning tachometers in Coover 1102 before the stroboscope lab for EE 448 in April of 2019
2. Have the students complete the lab
3. As the students leave, ask them to rate the tachometer on a scale from 1 to 10 in terms of usability. Ask them to give any additional feedback about the usability at this point.

Expected Results:

We expect that we will receive an average rating of over 80 usability. We also expect that we will receive feedback on difficulties which we can use to further improve user-friendliness of the tachometer.

Final Results:

Our tachometer received an average of 9.5/10 in terms of usability. We received the following feedback on the usability:

- Add a help button.
- Put the executable in an easier place to locate (such as on the desktop).
- Get rid of the start button as it can cause problems for some students, and has no true purpose.

Non-Functional Requirement #3: It will be documented sufficiently.

Test Case:

For this requirement, we want to determine that setup manual that for the ETG workers is well documented.

Test Steps:

1. Ask a member of the ETG to assemble our product from the individual pieces using our instructions
2. Have the ETG member complete a survey which discussed the ease of setup and asks them to give an explanation of any difficulties they had.

Expected Results:

We expect that the ETG member will have no medium or large difficulties. We also expect them to have minimal difficulties and give enough feedback for us to diagnose and resolve any issues with the clarity of the instructions.

Final Results:

As of 4/23/19, we tested out our manual with an ETG member and found that he had no issues with setting up using our manual. He had no additional feedback or suggestions.

Non-Functional Requirement #5: It will be resistant to breaking due to physical abuse by the students.

Test Case:

For this requirement, we want to determine that the product has a low level of maintenance required due to the product breaking.

Test Steps:

1. Install our fully-functioning products in Coover 1102 before the stroboscope lab for EE 448 in March of 2019
2. Have the students complete the lab
3. Take note of all requests for maintenance of the product to the ETG staff
4. Have the Teaching Assistant fill out a survey about their experience with the product, including if any of the products broke or required maintenance of any kind

Expected Results:

We expect that no products will require repair during the first semester. We also expect that if one breaks, we will be able to collect enough data through records from the ETG and TA in order to diagnose and determine a long-term solution to ensuring the tachometers do not regularly break in this way.

Final Results:

As of 4/26/19, we found that our tachometer is fully functional in the EE 448 lab. During in-lab testing, we found that none of the parts got damaged and did not need any repair. This is better than the stroboscope because there were three to four stroboscopes that were in need of repair by the end of every semester.

In addition, we found it was more resistant to breaking than our original scope required: we assumed that students would not mess with the hardware because they had no need. While one set of students did ‘break’ the tachometer by detaching the microcontroller from the computer, the system was fine once reattached. In the future, the motor will be entirely encased which will prevent this from happening.

4.6 Validation and Verification

Non-Functional Requirement #1: It will be more cost-effective to replace than the current stroboscope.

Test Case:

For this requirement, we want to determine that the final product is less expensive than the current stroboscope being used.

Test Steps:

1. Look up the price of the current stroboscope
2. Calculate possible price ranges for the different possible parts used for our design
3. Total the price to fully outfit a station with our new parts for the tachometer
4. Compare our total price for the tachometer to the original stroboscope price
5. The price of the new product should be cheaper and if not, change parts to something that would result in a cheaper model

Expected Results:

We expect that our final price to outfit a lab station with the new product will be less than 25% of the price of the current stroboscope.

Final Results:

As of 4/26/19, our tachometer costs $60.10 in total. This is 12% of the cost of the original stroboscope at $500, so we passed this requirement.

Piece	Price
Mount	$4.00
Sensor	$17.60[2] / $21.56[1]
Arduino	$22.00
PCB board	$10.00
USB cord	$3.50
Miscellaneous hardware	$3.00

[1] Price for individual sensors.

[2] Price per 10 sensors.

4.7 Final Evaluation

First and foremost, our final product meets all functional and nonfunctional requirements. Furthermore, our product received client, student, professor, and TA approval. The TAs were especially excited for the new tachometer as it made their job much easier. By removing the need for the user to interact with the motor shaft and measuring device, we were able to make the product more durable. It also made the lab safer, as the motor was a dangerous piece of equipment which required eye protection and had the potential to injure students as they measured if they were not careful.

We were able to reduce the cost by 88%, as the original stroboscope cost $500 compared to $60.10 for our tachometer. We were also able to reduce the average time to measurement by 80%, from an average of 50 seconds to 10 seconds every time. We verified usability of the product through student ratings, and scored an average of 9.5/10 on our model. We also added user feedback from this in-field testing to further improve our product in terms of usability.

5. Project & Risk Management

5.1 Task Decomposition, Roles, and Responsibilities

5.1.1 Roles and Responsibilities

See *Table 5.1.1.1: Roles and Responsibilities* for the list of roles, descriptions, and the members assigned to carry out these responsibilities.

Role	Member
Hardware Team - coordinate and lead the research, design, and implementation of the hardware	Kyle, Katrina, Seth*
Software Team - coordinate and lead the research, design, and implementation of the software	Meghna, Jessica, Seth*
Systems Engineer - attend both hardware and software meetings; assist in interface specifications	Seth
Testing Team - design and implement testing at the module and system level; in charge of ensuring the final product satisfies all requirements	Meghna, Jessica, Seth
Timeline Manager - create the group timeline; ensure the group is meeting deadlines	Katrina

Communication Manager - facilitate communication with the adviser/client, professor, and all other outside stakeholders	Jessica

Table 5.1.1.1: Roles and Responsibilities

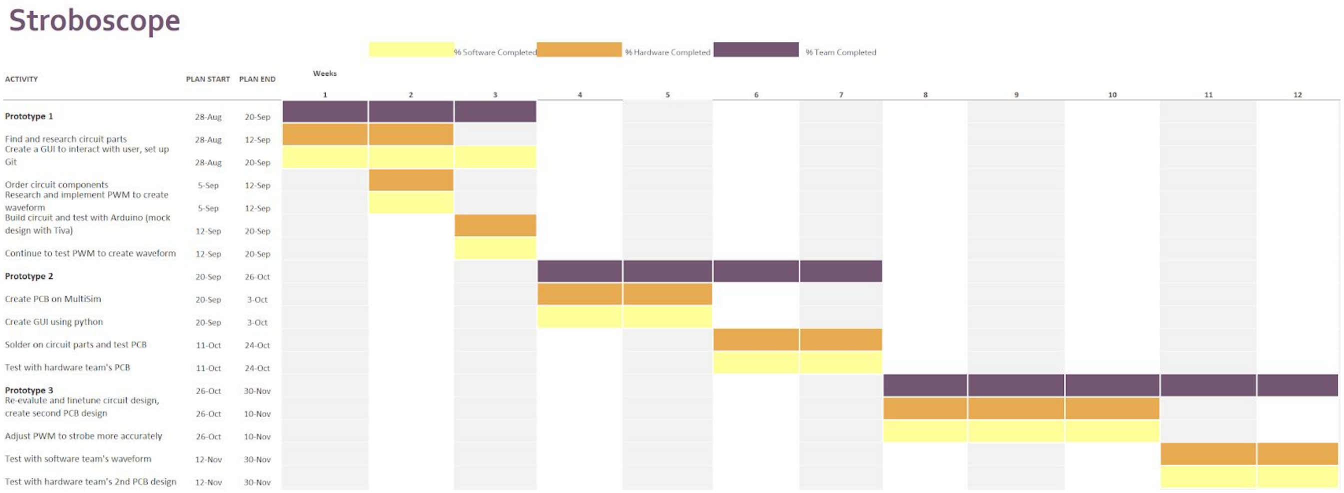
* refers to member holding this role for a partial semester

5.1.2 Task Decomposition

Task	Team member(s)
GUI creation	Jessica, Meghna
Arduino programming	Jessica, Meghna
GUI/Arduino interface	Jessica, Meghna
Software testing	Jessica, Meghna, Seth
Picking/ordering components	Seth, Katrina, Kyle
Mounting	Katrina, Kyle
Low-pass circuit design	Seth
PCB Design	Kyle
Hardware testing	Seth
System-level testing	Seth, Meghna, Jessica
Documentation	Full team
Language/IDE Research	Meghna, Jessica

5.2 Project Schedule

5.2.1 Proposed

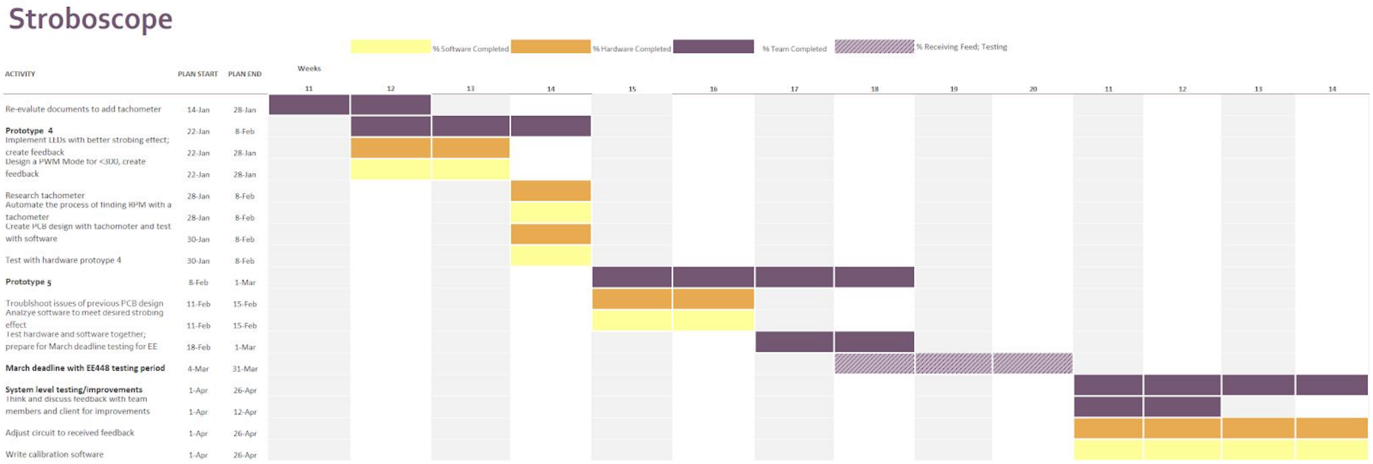


Stroboscope

🟨 % Software Completed 🟧 % Hardware Completed 🟪 % Team Completed

ACTIVITY	PLAN START	PLAN END	Weeks 1	2	3	4	5	6	7	8	9	10	11	12
Prototype 1	28-Aug	20-Sep	🟪	🟪	🟪									
Find and research circuit parts	28-Aug	12-Sep	🟧	🟧										
Create a GUI to interact with user, set up Git	28-Aug	20-Sep	🟨	🟨	🟨									
Order circuit components	5-Sep	12-Sep		🟧										
Research and implement PWM to create waveform	5-Sep	12-Sep		🟨										
Build circuit and test with Arduino (mock design with Tiva)	12-Sep	20-Sep			🟧									
Continue to test PWM to create waveform	12-Sep	20-Sep			🟨									
Prototype 2	20-Sep	26-Oct				🟪	🟪	🟪	🟪					
Create PCB on MultiSim	20-Sep	3-Oct				🟧	🟧							
Create GUI using python	20-Sep	3-Oct				🟨	🟨							
Solder on circuit parts and test PCB	11-Oct	24-Oct						🟧	🟧					
Test with hardware team's PCB	11-Oct	24-Oct						🟨	🟨					
Prototype 3	26-Oct	30-Nov								🟪	🟪	🟪	🟪	🟪
Re-evalute and finetune circuit design, create second PCB design	26-Oct	10-Nov								🟧	🟧	🟧		
Adjust PWM to strobe more accurately	26-Oct	10-Nov								🟨	🟨	🟨		
Test with software team's waveform	12-Nov	30-Nov											🟧	🟧
Test with hardware team's 2nd PCB design	12-Nov	30-Nov											🟨	🟨

Figure 5.2.1: Proposed Project Schedule, Semester 1



Stroboscope

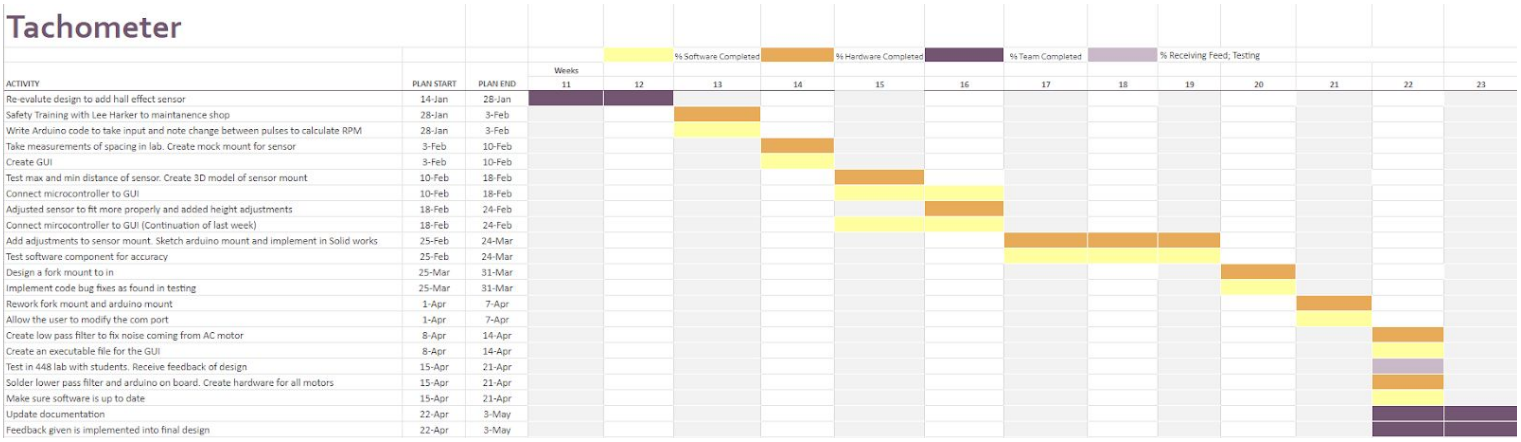
🟨 % Software Completed 🟧 % Hardware Completed 🟪 % Team Completed ▨ % Receiving Feed; Testing

ACTIVITY	PLAN START	PLAN END	Weeks 11	12	13	14	15	16	17	18	19	20	11	12	13	14
Re-evalute documents to add tachometer	14-Jan	28-Jan	🟪	🟪												
Prototype 4	22-Jan	8-Feb		🟪	🟪	🟪										
Implement LEDs with better strobing effect; create feedback	22-Jan	28-Jan		🟧	🟧											
Design a PWM Mode for <300, create feedback	22-Jan	28-Jan		🟨	🟨											
Research tachometer	28-Jan	8-Feb				🟧										
Automate the process of finding RPM with a tachometer	28-Jan	8-Feb				🟨										
Create PCB design with tachomoter and test with software	30-Jan	8-Feb				🟧										
Test with hardware protoype 4	30-Jan	8-Feb				🟨										
Prototype 5	8-Feb	1-Mar					🟪	🟪	🟪	🟪						
Troublshoot issues of previous PCB design	11-Feb	15-Feb					🟧	🟧								
Analzye software to meet desired strobing effect	11-Feb	15-Feb					🟨	🟨								
Test hardware and software together; prepare for March deadline testing for EE	18-Feb	1-Mar							🟪	🟪						
March deadline with EE448 testing period	4-Mar	31-Mar								▨	▨	▨				
System level testing/improvements	1-Apr	26-Apr											🟪	🟪	🟪	🟪
Think and discuss feedback with team members and client for improvements	1-Apr	12-Apr											🟪	🟪		
Adjust circuit to received feedback	1-Apr	26-Apr											🟧	🟧	🟧	🟧
Write calibration software	1-Apr	26-Apr											🟨	🟨	🟨	🟨

Figure 5.2.2: Proposed Project Schedule, Semester 2

5.2.2 Actual

(Semester 1 was equivalent to the *Figure 5.2.1: Proposed Project Schedule, Semester 1*).



Tachometer

🟨 % Software Completed 🟧 % Hardware Completed 🟪 % Team Completed ▨ % Receiving Feed; Testing

			Weeks												
ACTIVITY	PLAN START	PLAN END	11	12	13	14	15	16	17	18	19	20	21	22	23
Re-evalute design to add hall effect sensor	14-Jan	28-Jan	🟪	🟪											
Safety Training with Lee Harker to maintanence shop	28-Jan	3-Feb			🟧										
Write Arduino code to take input and note change between pulses to calculate RPM	28-Jan	3-Feb			🟨										
Take measurements of spacing in lab. Create mock mount for sensor	3-Feb	10-Feb				🟧									
Create GUI	3-Feb	10-Feb				🟨									
Test max and min distance of sensor. Create 3D model of sensor mount	10-Feb	18-Feb					🟧								
Connect microcontroller to GUI	10-Feb	18-Feb					🟨	🟨							
Adjusted sensor to fit more properly and added height adjustments	18-Feb	24-Feb						🟧							
Connect mircocontroller to GUI (Continuation of last week)	18-Feb	24-Feb					🟨	🟨							
Add adjustments to sensor mount. Sketch arduino mount and implement in Solid works	25-Feb	24-Mar							🟧	🟧	🟧				
Test software component for accuracy	25-Feb	24-Mar							🟨	🟨	🟨				
Design a fork mount to in	25-Mar	31-Mar										🟧			
Implement code bug fixes as found in testing	25-Mar	31-Mar										🟨			
Rework fork mount and arduino mount	1-Apr	7-Apr											🟧		
Allow the user to modify the com port	1-Apr	7-Apr											🟨		
Create low pass filter to fix noise coming from AC motor	8-Apr	14-Apr												🟧	
Create an executable file for the GUI	8-Apr	14-Apr												🟨	
Test in 448 lab with students. Receive feedback of design	15-Apr	21-Apr												▨	
Solder lower pass filter and arduino on board. Create hardware for all motors	15-Apr	21-Apr												🟧	
Make sure software is up to date	15-Apr	21-Apr												🟨	
Update documentation	22-Apr	3-May												🟪	🟪
Feedback given is implemented into final design	22-Apr	3-May												🟪	🟪

Figure 5.2.3: Actual Project Schedule, Semester 2

5.3 Risks & Mitigation

5.3.1 Anticipated

Going into the first semester we had anticipations about how our implementation of a stroboscope would go.

First, we anticipated it would be somewhat of a hurdle to get the specific SMD LEDs on time as the shipping time would slow things down. Also, we thought that when we got to designing and printing the PCBs, the wait time on them could potentially halt our progress.

Next, we were worried because no one in our group had any experience in designing printed circuit boards and we saw this as a huge learning curve to overcome. We planned on seeking help from Lee Harker to aid us before getting our first design completed and printed.

Finally, we were unsure of how accurate our clock speed on the Tiva board would be. We thought the latency of our circuit could add to the accuracy of the stroboscopes RPM measurement.

Most of the anticipated risks became irrelevant when we changed from a stroboscope to a tachometer at the beginning of the second semester.

When we found out we were switching to the tachometer going into the second semester, our first anticipated risk was that we would not have very much time to redo our designs and implementations. We attempted to mitigate this by changing our software code from the Tiva board to the Arduino.

We also worried about how much time we would need to redo our documentation, which was now irrelevant. Starting documentation early mitigated this issue.

Furthermore, working on hardware and software at the same time looked to be difficult. Because the hardware was not ready, we would not fully be able to test the software early. To smoothen this transition, our team defined unit tests which could be done without using other modules.

5.3.2 Actual

Our biggest risk was testing of our product. Because we were designing the hardware and software at the same time, we were somewhat limited in our ability to test the product as a whole. We solved this by isolating the components during unit testing, so we could test the hardware and the software without depending on the other. This also helped us when we got to system level testing because we already knew the components worked on their own. Another factor that helped us mitigate this risk was working on the components early; by having everything ready for testing at a system level several weeks before we needed the product, we gave ourselves time to test and fix errors.

Ordering parts became a large risk for us as well, due to the timeline of arrival. The first time we ordered parts with 2-day shipping, the shipping was delayed and it took about four days to arrive. The second time we did not realize that Saturday did not count as a shipping day. Both of these part delays required us to change our field testing schedule as we were unable to make our original deadlines. Luckily, in this situation we were able to rearrange our schedule. If we were to do it again, we would have ordered parts earlier.

Another risk was required components that were outside the scope of our education as computer and electrical engineers. A problem we ran into was that our system needed to be mounted to the motor well enough to avoid shaking, which would distort our sensor output. As this is generally a mechanical engineering task, we struggled to design this mount. This took a large amount of the semester for our hardware team. Eventually, we decided with our client/adviser that mount design was outside the scope of our project and decided that the client would have the mount redesigned later by someone in the mechanical engineering domain.

A final risk we faced was that our system would not be documented well enough to maintain in the future. We mitigated this by testing some of our documentation. After writing a manual to describe the setup of the full product on the motor, we had a third party who had no exposure to our project set the tachometer up on a fresh motor using the instructions, from the point of view of an ETG worker. We were able to take feedback from this experience to improve our documentation. Finally with our improved documentation, we had an ETG member set up the tachometer to verify the final version was clear and easy to follow.

5.4 Lessons Learned

Test out the product in the intended environment early on. We made the mistake of assuming the tachometer would work the same on the AC motor as on the DC motor, and did not try it on the AC motor until later in the process. This was when we found that the AC motor was unshielded and therefore gave off too much EM for the Hall effect sensor. If we would have verified this earlier, we would have had more time to solve this problem.

Check with all first to third party users early in the process. When we started, we worked with our client but did not verify the needs of the professor who taught the class, until the end of the first semester. This discussion ended up changing the project entirely because we did this so late, we were rushed in the second semester to redesign the entire project.

Order parts early, as others' time management is not always reliable. Twice, our expedited shipping took several days longer than what we paid for due to unforeseen shipping errors. While we were able to adjust our schedule to adapt to these delays, we would not have been able to make these adjustments in every situation. Therefore, we should have had our parts ready to order earlier.

Obtain measurements of the motor and dimensions of possible product before starting a 3D print. In the beginning, we would approximate the size of components because we were impatient and wanted to have a print to test; however, this proved to be time insensitive and inconvenient when we wanted to meet deadlines or mass produce a mount. Towards the end of the semester, we knew we had to meet the deadlines, especially for field testing, and were more conscious of taking accurate measurements before 3D printing.

Prepare all components. Because we did not prepare well the first time by gathering all the needed supplies required to set up our tachometer on each station in the lab, the setup ended up taking several hours longer than needed. By preparing ahead of time for the second setup, we reduced our time by quite a bit.

6. Conclusions

6.1 Closing Remarks

Our main goal was to meet all functional and non-functional requirements. We were able to verify 100% of our functional and non-functional requirements through accuracy testing (see *Section 4*).

Through field testing with our users, we were able to verify that our client (Matt Post), the EE 448 students, the professor teaching EE 448, and the lab TAs were all happy with the final product. The students rated our product an average of 9.5/10 in terms of usability. Furthermore, no tachometers broke during the lab sessions; while there is not yet enough testing to prove it is less or more reliable than the original stroboscope, still, this is a promising sign. We also removed the user's need to physically interact with the tool, which should make the tool more reliable and the motors safer. The TAs were able to verify that the new system is easier to teach to the students. In addition, we found that we were able to reduce the cost by 88%. We were able to reduce the time taken on average per measurement by 80% as well and overall, we found that our project was 100% successful in meeting our project objectives.

6.2 Future work

Our client has identified that it would be useful if the system could also change the rotational speed of the motor, rather than just measuring it. Adding this feature would remove 100% of the user's need to interact with the motor. Not only would this be easier for the user, but it would be safer because the motor could be encased completely.

Another addition to the system is the mount may be redone. As this was closer to mechanical engineering than computer/electrical engineering, it was decided this was outside the scope of our project. While the system as a whole fulfills all accuracy requirements with the current mount, a mount designed by a mechanical engineer could further improve the system.

Furthermore, the Arduino will be set to a specific COM port every time. This would allow the COM port to be hard-coded into the GUI code. Therefore, the user would not need to look up and change the COM port on the system.

6.3 References

[1] *IEEE Standard for Software and System Test*, IEEE Standard 829, 2008.

[2] *IEEE Standard for System, Software, and Hardware Verification and Validation*, IEEE Standard 1012, 2012.

[3] *IEEE Guide for Selecting and Using Reliability Predictions Based on IEEE 1413*, IEEE Standard 1413.1, 2002.

[4] T. Bigelow. "EE 448 Lab 5 Report.doc." Unpublished manuscript, EE 448: Introduction to AC Circuits and Motors, Iowa State University, Ames, Iowa, U.S.A.

[5] T. Bigelow. "EE 448 Lab 6 Report.doc." Unpublished manuscript, EE 448: Introduction to AC Circuits and Motors, Iowa State University, Ames, Iowa, United States.

6.4 Team Information

Jessica Bader - Computer Engineering and French LCP - jabader@iastate.edu - Spring 2020

Meghna Chandrasekaran - Computer Engineering - meghnac@iastate.edu - Spring 2019

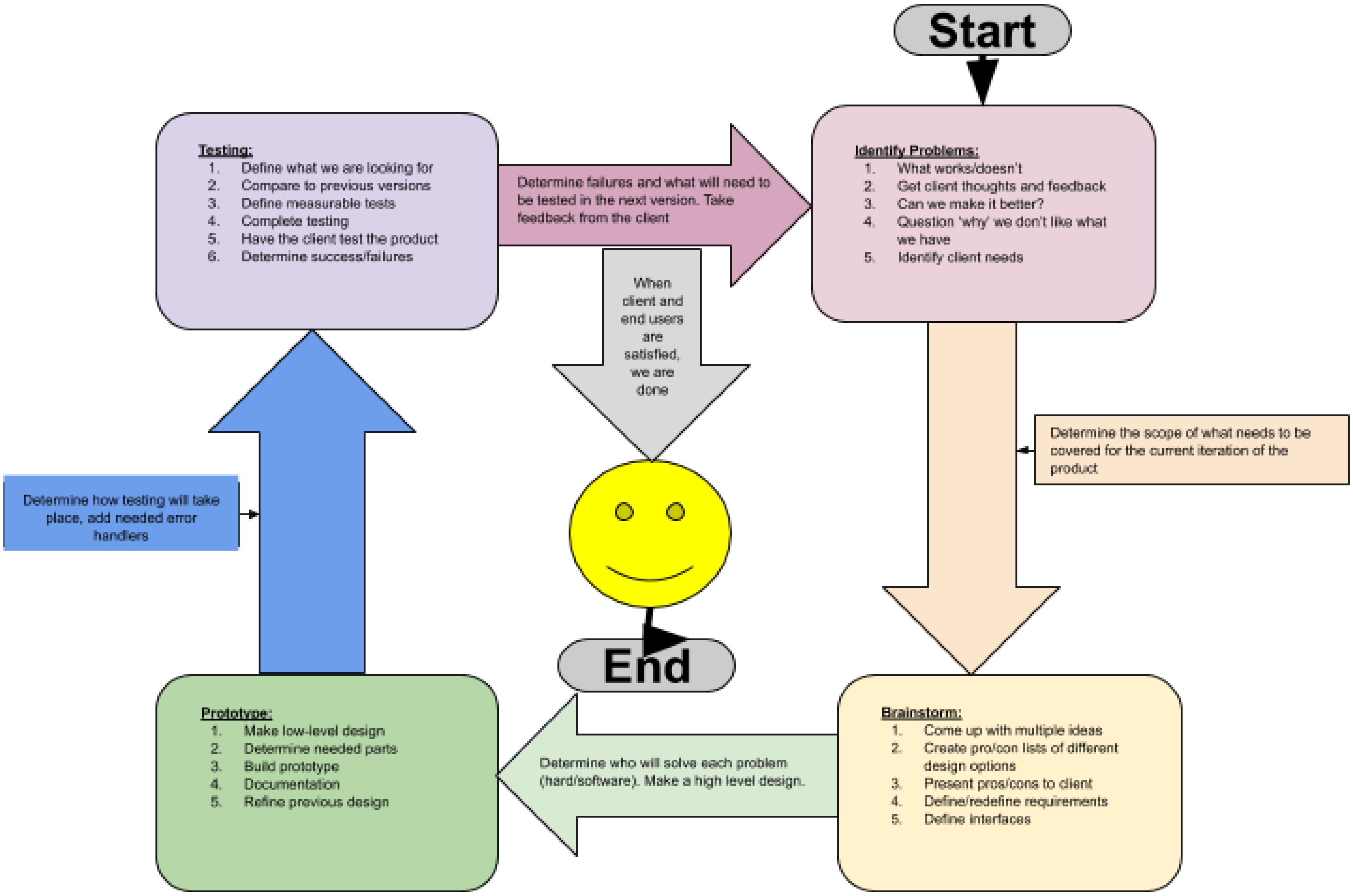
Katrina Choong - Electrical Engineering - kachoong@iastate.edu - Spring 2019

Seth Noel - Computer Engineering - sanoel@iastate.edu - Spring 2019

Kyle Zelnio - Computer Engineering - kjzelnio@iastate.edu - Spring 2019

Appendices

Appendix A: Design Process



Identify Problems:
1. What works/doesn't
2. Get client thoughts and feedback
3. Can we make it better?
4. Question 'why' we don't like what we have
5. Identify client needs

Determine the scope of what needs to be covered for the current iteration of the product

Brainstorm:
1. Come up with multiple ideas
2. Create pro/con lists of different design options
3. Present pros/cons to client
4. Define/redefine requirements
5. Define interfaces

Determine who will solve each problem (hard/software). Make a high level design.

Prototype:
1. Make low-level design
2. Determine needed parts
3. Build prototype
4. Documentation
5. Refine previous design

Determine how testing will take place, add needed error handlers

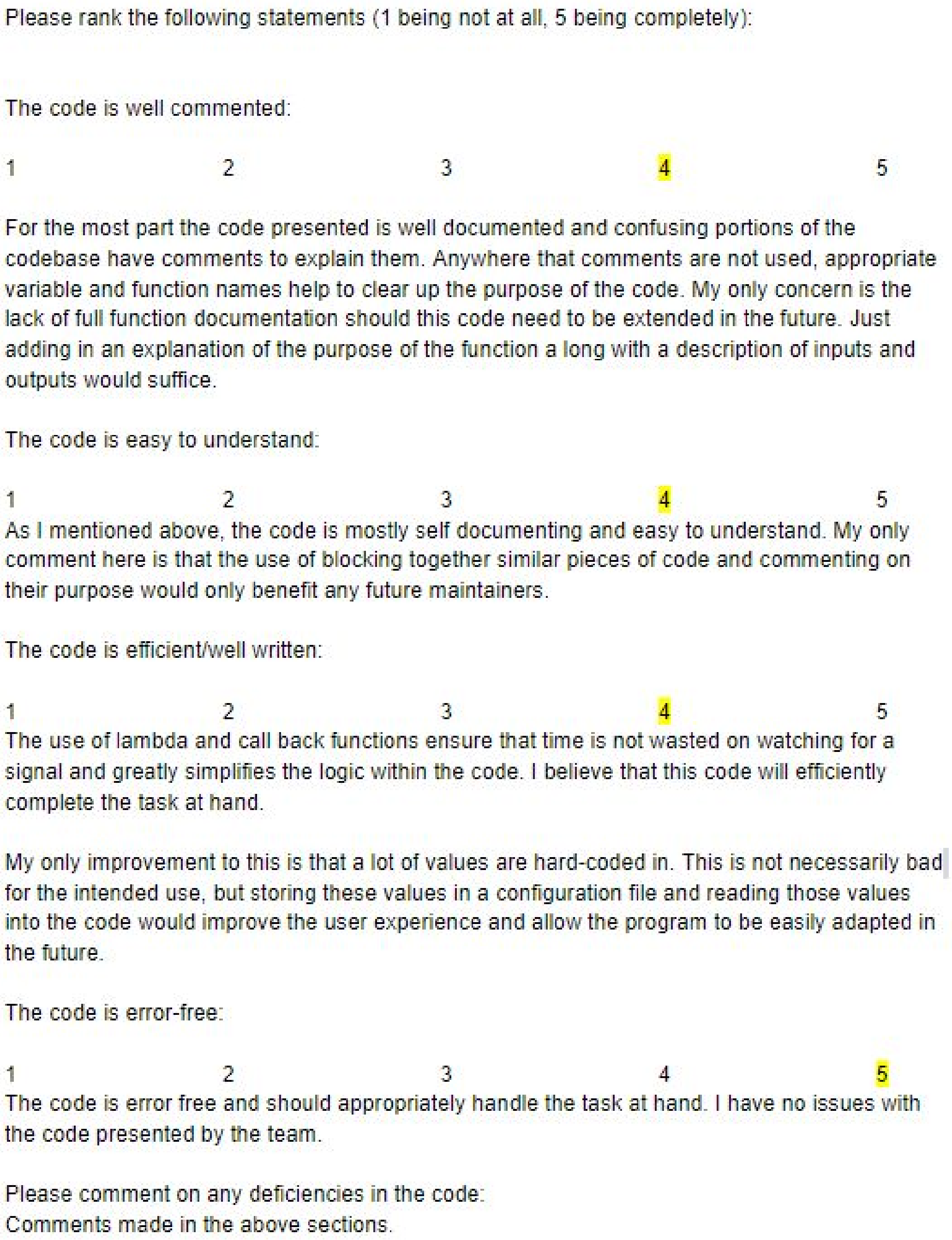
Testing:
1. Define what we are looking for
2. Compare to previous versions
3. Define measurable tests
4. Complete testing
5. Have the client test the product
6. Determine success/failures

Determine failures and what will need to be tested in the next version. Take feedback from the client

When client and end users are satisfied, we are done

Appendix B: Code Peer Reviews

Francisco Arreola



Please rank the following statements (1 being not at all, 5 being completely):

The code is well commented:

1 2 3 [✓ 4] 5

For the most part the code presented is well documented and confusing portions of the codebase have comments to explain them. Anywhere that comments are not used, appropriate variable and function names help to clear up the purpose of the code. My only concern is the lack of full function documentation should this code need to be extended in the future. Just adding in an explanation of the purpose of the function a long with a description of inputs and outputs would suffice.

The code is easy to understand:

1 2 3 [✓ 4] 5

As I mentioned above, the code is mostly self documenting and easy to understand. My only comment here is that the use of blocking together similar pieces of code and commenting on their purpose would only benefit any future maintainers.

The code is efficient/well written:

1 2 3 [✓ 4] 5

The use of lambda and call back functions ensure that time is not wasted on watching for a signal and greatly simplifies the logic within the code. I believe that this code will efficiently complete the task at hand.

My only improvement to this is that a lot of values are hard-coded in. This is not necessarily bad for the intended use, but storing these values in a configuration file and reading those values into the code would improve the user experience and allow the program to be easily adapted in the future.

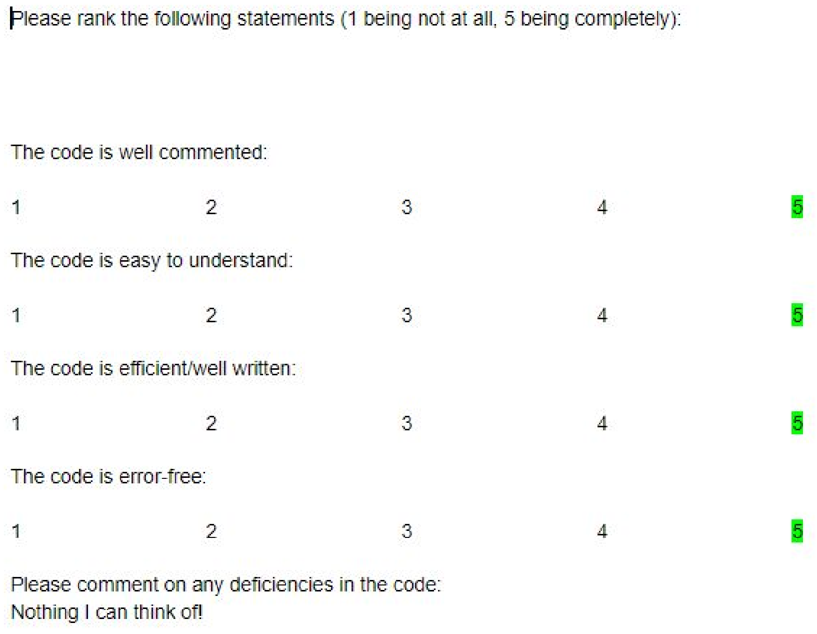
The code is error-free:

1 2 3 4 [✓ 5]

The code is error free and should appropriately handle the task at hand. I have no issues with the code presented by the team.

Please comment on any deficiencies in the code:
Comments made in the above sections.

Kristina Robinson


Please rank the following statements (1 being not at all, 5 being completely):

The code is well commented:

1 2 3 4 [✓ 5]

The code is easy to understand:

1 2 3 4 [✓ 5]

The code is efficient/well written:

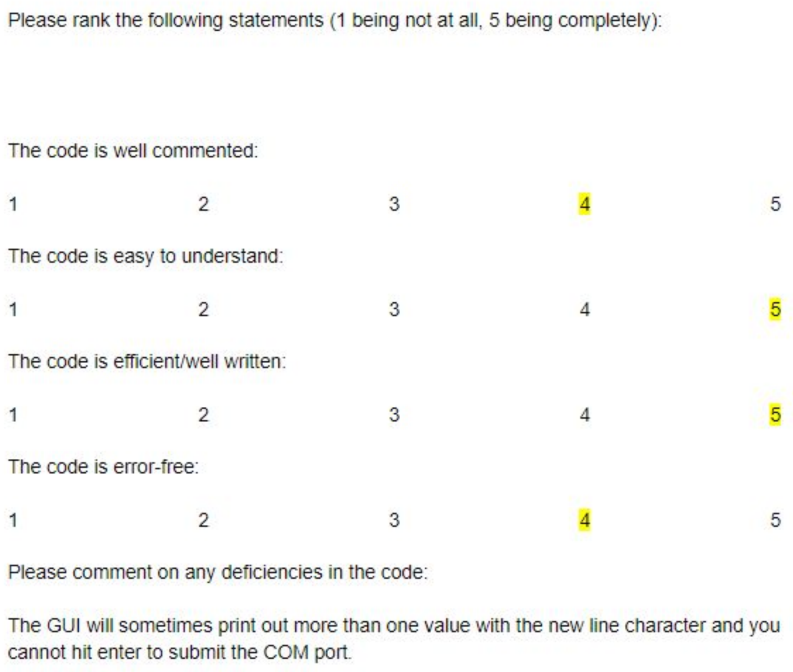
1 2 3 4 [✓ 5]

The code is error-free:

1 2 3 4 [✓ 5]

Please comment on any deficiencies in the code:
Nothing I can think of!

Kyle Zelnio


Please rank the following statements (1 being not at all, 5 being completely):

The code is well commented:

1 2 3 [✓ 4] 5

The code is easy to understand:

1 2 3 4 [✓ 5]

The code is efficient/well written:

1 2 3 4 [✓ 5]

The code is error-free:

1 2 3 [✓ 4] 5

Please comment on any deficiencies in the code:

The GUI will sometimes print out more than one value with the new line character and you cannot hit enter to submit the COM port.

Appendix C: Software Accuracy Testing

Goal RPM:	Hz	RPM (generator)	RPM (Arduino)	Accuracy
100	1.666666667	100	99	99%
200	3.333333333	200	201	99.50%
300	5	300	300	100.00%
400	6.666666667	400	399	99.75%
500	8.333333333	500	501	99.80%
600	10	600	600	100.00%
700	11.66666667	700	699	99.86%
800	13.33333333	800	801	99.88%
900	15	900	897	99.67%
1000	16.66666667	1000	999	99.90%
1100	18.33333333	1100	1101	99.91%
1200	20	1200	1200	100.00%
1300	21.66666667	1300	1299	99.92%
1400	23.33333333	1400	1401	99.93%
1500	25	1500	1497	99.80%
1600	26.66666667	1600	1599	99.94%
1700	28.33333333	1700	1701	99.94%
1800	30	1800	1797	99.83%
1900	31.66666667	1900	1899	99.95%
2000	33.33333333	2000	1998	99.90%

Appendix D: System Accuracy Testing

AC:

Goal RPM:	RPM (stroboscope)	RPM (Arduino)	Accuracy
100	124	120	96.774%
200	196	183	93.367%
300	307	309	99.353%
400	404	407	99.263%
500	505	504	99.802%
600	590	591	99.831%
700	708	708	100.000%
800	816	816	100.000%
900	902	900	99.778%
1000	1019	1014	99.509%
1100	1100	1098	99.818%
1200	1191	1194	99.749%
1300	1312	1311	99.924%
1400	1391	1389	99.856%
1500	1514	1515	99.934%
1600	1593	1593	100.000%
1700	1722	1719	99.826%
1800	1804	1802	99.889%

Appendix D: System Accuracy Testing (Continued)

DC:

Goal RPM:	RPM (stroboscope)	RPM (Arduino)	Accuracy
100			
200	167	177	94.35%
300	310	309	99.68%
400	423	423	100.00%
500	511	510	99.80%
600	617	615	99.68%
700	702	702	100.00%
800	809	807	99.75%
900	915	915	100.00%
1000	1012	1011	99.90%
1100	1115	1113	99.82%
1200	1191	1191	100.00%
1300	1302	1299	99.77%
1400	1414	1413	99.93%
1500	1509	1509	100.00%
1600	1611	1611	100.00%
1700	1707	1707	100.00%
1800	1815	1815	100.00%
1900	1897	1896	99.95%
2000	1996	2001	99.75%