Code Review Documentation

Carter Brezinski - 200391111

Capstone Project - F.L.O.A.T.

April 9, 2022

1. Coding Overview

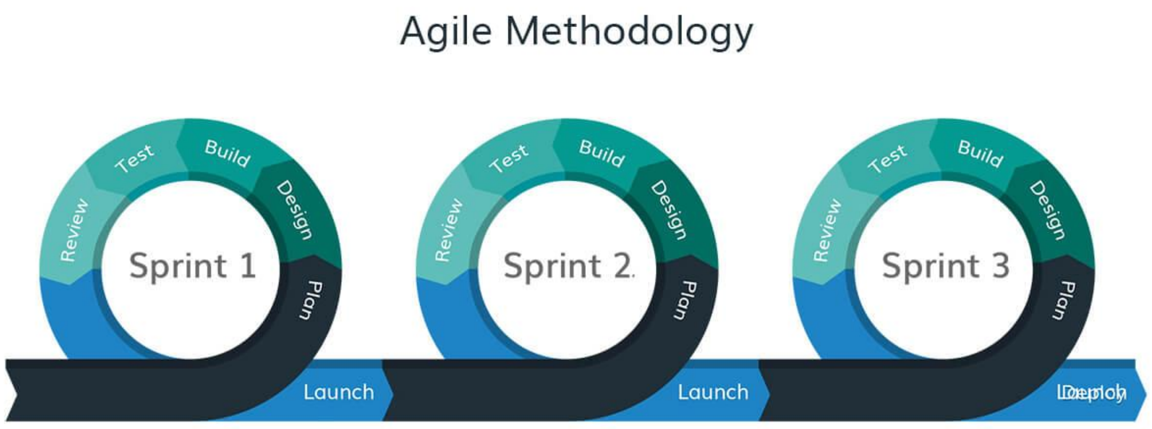
Compared to other capstone projects designed this year, our project covers multiple different environments. There was Jon's area of work which covered web development and the website, and my area of work encompassed both in-the-field work with the Raspberry Pi, and the analysis work performed in Google Colab. In this code review I will explain the engineering practices and design patterns I used and followed throughout the design and development process, and the significant changes I made to my portion of the code over the past seven months.

Table of Contents

2. Engineering Practices

2.1. Agile Methodology

From the very early stages of planning and research in our project, Jon and I agreed that we would follow the Agile Methodology for our individual developments. As our project changed in size and scope over time, so did our code to accommodate these changes. In my case, this involved consistent reviewing of code, refactoring of code to accommodate necessary changes found during tests, and the removal of repeating and redundant code as per the YAGNI* and DRY** principles we've learned in previous classes.



[1]YAGNI - You aren't gonna need it

[2]DRY - Don't Repeat Yourself

2.2. Clean Code

For myself, I always aim to develop code that is writable, reliable, and easily readable. When I develop code, I want it to be writable from the perspective that when I'm coding I know exactly what the code is doing, and in the same breath to be readable so that someone who has very little coding knowledge can deduce what is happening within the code, as writing code does not just end at a complete compilation.

Typically, since my code can become complicated and difficult to understand, I will include comments to explain to users what is occurring in certain blocks of code. Specific examples of this can be found in my ipython notebook for the image detection analysis. Many of these included Yolov4 and Darknet function calls that are not the easiest to read or understand, hence the insertion of explanations, for both myself and external users.

2.3. Frequent Testing

In the later months of the project, I conducted frequent cycles of training and testing of the code to ensure it would operate as expected. For the image, webcam, and video analysis, I ran the necessary files and models to have each version work, and I would test them against the datasets that they required to run correctly.

During this cycle of testing and training, if the accuracy of results was too low, or if the algorithm was unable to detect the objects in question, I would work back through my development process up until that point to determine the cause of the issue. This would result in me asking myself questions similar to the following:

- Why might these objects not be detected currently?
- Is the model trained for this type of object being tested against?
- Are the files being called upon properly formatted for this dataset?
- Have I trained the algorithm long enough?
- Am I using the best weighted model?

Typically, after asking myself these questions and verifying with the implemented files, I was able to come to a concise answer as to why certain objects were not being detected during analysis, solve the issue, and continue testing efficiently.

3. Design Patterns

As mentioned in the Project Overview, the proof-of-concept project that Jon and I worked on did not follow what would be considered a traditional software development project. I cannot fully speak to Jon's portion of code; however, for my code, there were very few design patterns I was actually able to implement in the development of my portion of the project.

3.1. NULL Object Design Pattern

For the Python scripts intended for data collection on the Raspberry Pi, on occasion the hardware chosen would either produce improper readings, or zero readings. As a result, I have documentation and check statements to ensure that if the sensors provide no reading, it will provide a reading that can be inserted into the csv file safely without causing the runtime to fail.

An example of this can be seen in the code below. This is code used to extract data from the humidity and temperature sensor:

```python
try:
    # Grab temperature and humidity from the sensor
    # It isn't always 100% accurate, hence the try and exception cases.
    temperature = sensor.temperature
    humidity = sensor.humidity
    print("Temp: {:.1f} *C \t Humidity: {}%".format(temperature, humidity))
except RuntimeError as e:
    # Reading wont always work! hence the error message
    # Directly plugging in 0's into the original print statement as
    # an indicator that the sensor wasn't behaving as intended.
    print("Reading from DHT failure: ", e.args)
    print("Temp: {:.1f} *C \t Humidity: {}%".format(0, 0))
    temperature = 0
    humidity = 0
temp = temperature
humid = humidity
```

When reading from the temperature and humidity sensor, if the sensor misbehaves, typically an error will be thrown and the runtime will cease. This is an issue, due to the fact that these scripts are supposed to run until a user manually stops the runtime. To solve this issue as seen in the code above, the format printed to the user's console view displays zeroes to indicate a malfunction in the sensor. In addition to this, the values of temperature and humidity are set to 0, so that they can safely be displayed in the csv file written into later in the program.

3.2. Builder Design Pattern

In the Google Colab development, for the training and implementations of my models, I was able to reuse a large majority of my code specifically for compiling the images and annotations data off of the google drive, and applying them for training and testing. I would consider this to be a functional example of the builder design pattern, as it takes the provided set of images and data files and makes them into a working final product to be used in training and testing.

4. Significant Development Changes

4.1. Developing within the Raspberry Pi

In the final meeting with Christine, she made it apparent that in order to ensure our project was notable, gathering additional parameters including GPS and water turbidity was recommended. She suggested talking to our industry reference Matthew Palmarin, for his input and recommendation with respect to including an additional sensor. This additional sensor did not take long to implement, and enhanced the data gathering portion of our project to appear more professional.

With the Raspberry Pi, I periodically gathered parameter data from the water & nearby environment, and consolidated this information into a csv file. This csv file would then be passed to Jon's website. On the website, the GPS data would be converted into a linemap, similar in style to that of Google Maps, and is easily viewable on the website by operators or other individuals.

4.2. Changing Datasets

In the second semester of the capstone project, I undertook significant changes to the foundation of how the project functioned up until that point. Originally, Jon and I worked with the TACO Dataset, an open image dataset of waste found in various outdoor environments (1). In the beginning of our project, we were able to use a demo model provided on the Github page of the TACO Dataset.

This format worked until I began research into developing our video detection aspect of the project. The reason behind this format no longer worked for two reasons. Firstly, the pretrained demo model had a set of file types that the video detection commands would not compile with. Secondly, the TACO dataset company stored and called upon their images during runtime.

It was because of both of those reasons, that I needed to make the change and develop my own dataset and model, to properly be able to interface with the yolov4 and AlexeyAB Darknet architecture (2). The solution to this was to compile a unique dataset of objects using Google's Open Image Dataset. With this open image dataset, I was able to select specific objects to download images, and the number of images I required to download for training and testing, and how they would be formatted. This was a drastic change in the development of the project, but it allowed for us to have the successful final version with working video detection.

Reference Links:

1. Taco Trash Annotations Dataset: http://tacodataset.org/
2. Alexey AB Darknet Architecture: https://github.com/AlexeyAB/darknet