Team Fire Scout

Software Testing Plan

3/26/2021

Sponsor:

Dr. Fatemeh Afghah
Alireza Shamsoshoara

Mentor:

Sambashiva Kethireddy

Team Members:

Nicholas Bollone
Matthew Briody
Jacob Hagan
Kenneth Klawitter
Drew Sansom

This document outlines how the Fire Scout System as a whole will be tested. Testing will include unit, integration, and usability testing.

Introduction **2**

Unit Testing **2**

Shared System Data 3

Drone Ops 3

Drone Data 4

Integration and System Testing **5**

Testing the SDRs 5

Testing with Sensors 6

Testing the GUI 7

Usability Testing **7**

Conclusion **7**

Introduction

Fire Scout is a joint computer science/electrical engineering project between two senior-level capstone teams at Northern Arizona University. The goal of the project is to create hardware and software capable of identifying and analyzing wildfires when attached to Unmanned Aerial Systems (UAS's), more commonly called drones. This solution removes humans from harm's way when analyzing fires while providing a cheaper alternative to aerial and satellite imagery.

Since the end of the development cycle is near the team has sought to start thorough and robust testing. Testing allows software developers to make changes to code while still having a way to measure its success. It not only allows the current Fire Scout Team to see if the code they have is working (although team members have been testing code independently throughout the project) but will provide a basic testing framework for future developers to expand upon.

Since Fire Scout is a joint project between computer scientists and electrical engineers involving specialed hardware including mini-computers and software-defined radios (SDRs), and since the software development emphasizes artificial intelligence, there will likely be fewer unit tests compared to other senior capstone projects. Instead, integration and system testing will ensure that the electrical engineering and computer science components work cohesively together. Additionally, the users' input is limited to only the options the computer science team gives them, so there will be little usability testing. The user has limited AI models that can run on the drone, and there is limited information the system is capable of collecting, so there is less need than other projects to test input.

Unit Testing

Unit Testing looks at small individual pieces of code rather than how those components fit together as a whole. By testing a method, class, or other pieces of code, developers can ensure that it is working throughout the whole program. If a part of the program using code

that has been unit tested is not working, the unit tests help rule out possibilities of what is going wrong. However, all of this assumes that the unit test was written correctly.

As discussed, a large component of Fire Scout involves integrating the electrical engineers' work with the computer scientists' so integration and system testing will be performed in more detail. Knowing this, unit testing will focus on the classes and methods that interact with EE's code or are called by the CS team throughout the system.

Shared System Data

The shared system data is the set of classes representing the data sent through the SDRs. This data is either written by the drone to be displayed on the GUI, or written on the GUI for the drone to process.

Drone Ops

The DroneOps class represents data written by the GUI for the drone to interpret as instructions. Testing this class is as simple as testing that each mutator and accessor method that writes or reads values. This is fairly trivial since there are only so many values that the system as a whole was designed to understand. For instance, when the user wants to turn image classification on, the classification option is set to "on" rather than "off" based on the GUI button they click. The below code snippet shows the python dictionary that holds options for the drone.

```
self.ops = {
   "classification": "off",
   "obj_det": "off",
   "segmentation": "off",
   "hd": "off",
   "thermal": "off"
}
```

Each one of the keys has a set number of acceptable values and methods to toggle those values. Those values are:

```
"classification": "off" or "on"
"obj_det": "off" or "on"
"segmentation": "off" or "on"
"hd": "off" or "on"
"thermal": "off" or "red" or "white" or "green"
```

Unit tests, therefore, are as simple as calling every method in the class that modifies the dictionary and seeing if the dictionary values changed correctly.

The next set of units/methods to be tested are the boolean checker methods that reexamine the dictionary. These are used on the drone to see if its local dictionary has been changed and if the drone needs to run or stop any operations. Unit testing will be as simple as changing each dictionary value (shown above) and verifying that the methods return the expected output.

The final set of methods are responsible for reading and writing the dictionary information to files on the drone and ground system respectively. These methods consist of helpers that convert the Python dictionary to a JSON, and back again, as well as callable methods that will read/write to paths defined in the class's instance variables. Testing these methods will simply involve calling them and seeing if the written data can be read from the file back into the dictionary. If data can be written/read or read/written, then the methods are working.

Drone Data

The DroneData class represents the data sent from the drone collected by EE's sensors to the GUI. This class was created as a simple way for the EE team to save sensor info in a format that could easily be sent through the SDR and interpreted by the CS team. Like the DroneOps class, much of this class consists of simple accessors and mutators, and testing them is as simple as seeing if each function successfully reads or writes the expected values to the Python dictionary. Knowing the testing for this class follows the same methods as the aforementioned class (checking to see if the methods read/write the expected values), it is sufficient to simply show the class dictionary knowing that all values will be changed and examined in testing:

```
self.sensor_data = {
   "compass" : 123,
   "lat" : 123,
   "long" : 123,
   "alt" : 123,
   "temp" : 123,
   "humidity" : 123,
   "co2" : 123
}
```

Integration and System Testing

Integration testing examines how well modules, classes, and other sizable pieces of code integrate together. While this is relevant for many projects, Fire Scout will instead focus on system testing which tests how well information and data flow through the system as a whole. This decision was made because the different software modules produced by the computer science team integrate together through the electrical engineering components. To see how well the software modules and code integrate together, the CS team has to send signals through the SDRs. This naturally turns into a system test.

Any small interaction between modules on either the ground or drone will have been tested by unit testing since the classes exist to provide ways for the EE team or the GUI to read and write data to specified dictionaries, JSON files, or paths with ease. If the methods in DroneOps and DroneData pass unit testing, then they can be called anywhere in the architecture with confidence they will accomplish their task, therefore they are integrated.

Finally, since the responsibilities of the Fire Scout Project are divided between CS and EE with clear guidelines there are some components that if not functioning properly are beyond the control and scope of the CS team to fix. The following system tests, however, go above and beyond what is expected of the CS team since they also test the success of EE's code.

Testing the SDRs

After the CS team writes data to a specified folder on either the drone or ground, EE is responsible for sending it through the SDRs. Through unit testing, the team will know that

reader/writer methods responsible for communicating with these folders work. To test if the SDR's themselves work and if receiving/sending signals work, the most natural way is to manually try and send signals through them. This testing will closely resemble walking the client through a prototype. The below workflow tests every possible user action that can be taken on the ground system, and tests if the drone responds to every action correctly.

1. Turn on the GUI
2. Turn on the Drone
3. Turn a task on with a button
    a. Classification
    b. Object Detection
    c. Segmentation
    d. HD feed
    e. Thermal Feed
    f. Kill all operations
4. Drone receives a signal
5. Drone runs the specified task
6. Drone sends data back
7. GUI displays data

If Steps 1 - 6 can be repeated with every task mentioned in step 3, then the team knows that the SDRs are working for all use cases and that the software on the drone and ground are integrated correctly through the EE's system.

Testing with Sensors

Electrical engineering is responsible for collecting information from the sensors attached to the Jetson Nano. Through unit testing, the team will prove that EE can successfully write and read to a file containing this information. After the file receives this data, the data needs to be sent through the SDRs to the ground system. From there, the ground system must display the information on the GUI. Testing follows a workflow similar to testing the SDRs.

1. Turn on the GUI
2. Turn on the Drone
3. The drone will automatically write its sensor data for the SDR to read
4. The drone SDR will send the sensor data
5. The GUI will automatically read the sensor data delivered by the corresponding SDR and display it.

Again, this is a very clear and concise test that looks at how well the system as a whole works and how well the computer scientists' code integrates together through EE's components.

Testing the GUI

The GUI itself trivially displays its elements or not. It also only uses methods defined in the DroneData and DroneOps classes which will be tested in unit testing. Therefore, we are sure that the GUI will integrate with other components of code on the ground since all it does is link buttons to methods. If a button can change the python dictionary through a method, then it is integrated. This is tested in the above "Testing the SDRs" system test.

Usability Testing

Usability testing will also be kept to a minimum. The goal of usability testing is to ensure that the end-users find using the product intuitive, and are pleased with how it operates. In the case of Fire Scout, all product functions are presented to the user in the GUI (mentioned in the SDR Testing section above). The GUI simply shows buttons to the user that can toggle these functions on and off. Our development of the GUI has and will continue to consist of showing our GUI to Ali and Dr. Afghah ensures that they are pleased with the workflow of the GUI and the system as a whole. However, to test that they are ultimately pleased with the GUI, we will give the GUI to Ali and Dr. Afghah, then without showing them how it works, ask them to use every feature on it. After listening to their feedback, we will make any adjustments as they see fit. This entire method not only allows us to work with them to develop a GUI that satisfies

their requirements, but then examines the end result of our development by giving them the chance to use it unassisted.

Conclusion

While the tests for Fire Scout may be limited, especially compared to other capstone groups, they are applicable for a joint CS/EE capstone project. The unit tests will provide the foundation for all future tests by ensuring that all class methods perform correctly. These tests will ensure that both the CS and EE teams can manipulate, store, and share data across the system as a whole. Next, Fire Scout's integration and system tests will show that the EE and CS components work together cohesively and test that the pipeline from the ground to the drone is working correctly. Finally, usability testing, while limited, helps Fire Scout meet the client's requirements by working directly with them to create the GUI, and then letting them test it to see if they are pleased with the end results. These tests will help show the current software is working, that it interfaces with EE's developments, and paves the way for future teams to expand upon.