RAMPART

Remote Aerial Mission Planning and Radio Tracker

Software Testing Document Version 1.0

April 3, 2020

Team Members

Eric Gault

Samuel Gilb

Keller Mikkelson

Nathaniel Zeleny

Team Sponsor: Dr. Michael Shafer

Team Mentor: Mahsa Keshavarz

Table of Contents

1 Introduction

Our project is motivated by the need to better develop a way to gather locational information from Very High Frequency (VHF) tags that are used to track a plethora of different animals throughout the world. The most common way this is done is by personally going out into the wild in order to gather this data using a handheld receiver. However, our client wants to develop a way to attach a receiver to a drone and have that drone collect the information in a faster and safer nature. The goal of our application is to merge a flight planning software as well as our clients own software into one. Our client's software currently contains a File Transfer Protocol (FTP) system, a User Datagram Protocol (UDP) system, as well as a configuration file manager for creating configuration files. The reason we need to test this program is because we will be working with expensive components on and including the drone, as well as collecting data so we need to ensure what we are gathering is correct.

Our testing for this software is going to heavily involve Unit Testing, Integration Testing as well as Usability Testing. All three types will be treated and testing equally as all of them are very important for our software. Unit Testing will be used for the configuration file manager to ensure that the data is being handled correctly. This is very important as if the data ends up being wrong, it can waste hours of our client's time or cause a crash which can end up costing money. Integration Testing will be used to ensure that all of our additions to the program work together and any of our changes do not cause any problems with the base functionality of the application we are building upon. Finally, Usability Testing is important because our project is being developed mainly for our client to use, therefore he needs to be able to use our software to its fullest. Our software is going to be rigorously tested because we are working with expensive components so any problems that occur that we do not catch in the testing phase can cost our clients money and time.

In the rest of this document we will go into more detail about each of our testing regimes. First we will start with Unit Testing, after that we will talk about Integration Testing and finally we will have Usability Testing.

2 Unit Testing

Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. For our unit testing we will be utilizing QT Test to test our units. This library is extremely helpful with testing QT functions and since our project is written in QT it was an obvious fit. Our rationale for not testing the units in the user interface is that the units don't provide useful boundaries and the tests would be based upon our visual assent instead of some measurable metric. The units we are testing fall into 3 major categories: The FTP (File Transfer Protocol) Module, UDP (User Datagram Protocol) terminal management, and The Configuration Data Generation/Handling Module. In the sections below we explore each of the units contained within these three and cover how we will be testing them.

2.1 FTP Module

In order to test the module constructed for the purpose of interacting with the companion computer's FTP server on the drone, we must look at what interactions we are doing with the FTP. Users need to be able to connect to a valid FTP server, go through login procedures, and receive the server list tree. Then, users should have file upload and download capability with the server. In addition to these steps, the FTP server returns a state that shows what process it is currently doing. These steps and states are what need to be covered in this module's unit testing. Here are the units that will be tested to cover the mentioned steps and states:

- **Valid server address**: This makes sure that there is at least some type of network address being accessed for safety of the user and security of the program.

    - Must be able to access all IP classes (A through E)
        - These are classifications that determine what an IP address is based off of the size of a company or expected traffic.
    - Address must have 4 bytes separated by a "."
        - Valid test value: 104.248.178.78
    - Each byte is bounded at a min of 0 with a max of 255.
        - Valid test value: 104.248.178.78

- **Login**: This makes sure that we get proper access to the FTP.
    - Qftp object state must be connected or login.
        - Valid state values: 3 and 4
    - Login can be a specific user or anonymous:
        - Valid user and pass: rampart:rampart
        - Valid anonymous: anonymous:
            - An anonymous login has no password, so after the ":" it is left blank.

- **Qftp command**: This is a value that is returned when the Qftp object is executing a command given by the server or user. If it has gone out of bounds, then that means there is some fatal error with the server.
    - Qftp object must have a command value from 0 to 14
        - Valid command value: 7 (this is the change directory command)

- **GET file**: This process is executed when a user wants to download a file from the FTP server onto their local machine. It requires asynchronous computation, meaning the process can be split up between other processes. Because of this, it is preferred that the current Qftp command be 8 (the GET command) but it's not required for the entire duration.
    - Qftp starting current command must be 8
    - Qftp current command must range from 0 to 14
        - Invalid command values: 3, 4, and 5
            - These are commands that are for establishing or removing the server connection and cannot be done during a transfer process.

- **PUT file**: This process is executed when a user wants to upload a file from their local machine onto the FTP server. It requires asynchronous computation, meaning the process can be split up between other processes. Because of this, it is preferred that the current Qftp command be 9 (the PUT command) but it's not required for the entire duration.

- Qftp starting current command must be 9
- Qftp current command must range from 0 to 14
    - Invalid command values: 3, 4, and 5
        - These are commands that are for establishing or removing the server connection and cannot be done during a transfer process.

2.2 UDP Terminal Management

The UDP terminal management system handles the receiving and displaying of UDP heartbeat messages. First we receive the message from the drone with the receiver method which handles any potential malformed packets and only sends valid input to the display method. In the display method we process the messages to provide the user with useful feedback.

- **Receiver Method:** The receiver method listens for UDP Heartbeat messages sent from the drone. Upon receiving one of these messages the receiver function inspects it to make sure the message was received correctly. It then passes the message to the Display method. Our Unit tests will cover the various spectrum of messages. We will test messages that contain too little data, the correct amount of data, and too much data.
    - Message range
        - 1-15 chars is invalid
        - 16-32 chars is valid
        - 33 chars and above is invalid

2.3 Configuration File Generation/Handling Module

The final set of unit tests we will be going over regards the generation and handling of the system's configuration files. These files indicate what kind of VHF tags the drone will be scanning for and the filtering settings that will accompany the scan. The file generated is essentially a basic text file that the drone's companion computer parses for each important piece of information. Ensuring that each file generated by our program contains the proper format and the exact settings requested by the user is a key step to consistent and accurate data gathering. On top of this, the program should be able to import the settings from a previously-generated configuration file. Therefore, two unit tests will be used here: one to check that our program generates valid configuration files, and another to ensure that the correct settings and formatting are used when importing an existing configuration file. These unit tests are:

- **Generate Config File:** Creates a configuration file based on the settings input by the user. First, the unit test will check to ensure that the values input by the user are within a valid range. If the values input by the user are invalid, a configuration file should not be generated. If the values input by the user are valid, the unit test will parse the newly-generated file in order to ensure that the formatting is correct. Each parameter within the configuration will be tested and will consist of three equivalence partitions. These partitions will test a value below the valid range, within the valid range, and above the valid range. These ranges, or boundary values, are as follows:
    - Tag Frequency Range (MHz) : 30MHz to 300MHz
        - Valid test value: 149.803 MHz
    - RF Gain (dB): (Min - 10dB) to (Max + 10dB)
        - Valid test value: (Max + 0) dB
    - IF Gain (dB): (Min - 10dB) to (Max + 10dB)
        - Valid test value: (Max + 0) dB
    - BB Gain (dB): (Min - 10dB) to (Max + 10dB)
        - Valid test value: (Max + 0) dB
    - Radio Sampling Rate (MHz): 24 MHz to 1800 MHz
        - Valid test value: 300 MHz
    - Pulse Duration (s): 0.01 second to 1 second
        - Valid test value: 0.1 second
    - Pulse Repetition Rate (s): every 1 second to every 10 seconds
        - Valid test value: every 5 seconds
    - UAV Telemetry Sample Rate (MHz): 0 MHz to 64 MHz
        - Valid test value: 24 MHz

- **Re-Use Config File:** If the user wants to re-use or alter a pre-existing configuration file, our program allows them to do so. When the user selects the file they wish to import, the program will parse the given file and pull the various required configuration file parameters. The unit test for this will use the same three partitions as the “generate config file” unit test listed above, but it will also include a check to make sure that each value filled into the blanks of our GUI are correctly pulled from the given configuration file.

3 Integration Testing

Integration testing is done after the individualized unit tests for each module by determining if the modules communicate correctly between each other. These kinds of tests are done because it is a beneficial way of guaranteeing a degree of cohesiveness between our modules instead of just gambling on the fact that they worked on their own environment so they must work together. To figure out what we wanted to test in integration testing we looked at a UML diagram of our project and chose critical sections of communication that could break our code and decided to test these important sections. The key components we decided to address were the UDP messages being sent to the terminal in the User Interface, The correctness of files being sent from config file module to the FTP module, and the Network communications of the FTP module.

3.1 UDP Messages

For our UDP messages we want to make sure the terminal display in the UI matches up with the messages sent from the UDP module. To test this we will be utilizing a program created by our client that generates fake heartbeat messages and we will save the messages and compare with the terminal display to make sure that the messages display properly.

3.2 File Integrity

Our FTP and configuration file unit tests are tightly coupled. Both of the configuration file unit tests are used to confirm that valid configuration files are generated when the user needs them, but the FTP unit tests ensure that we are sending and retrieving data to the correct drone’s companion computer. When combining these unit tests, we will check to make sure that valid configuration files are generated and sent to the companion computer using said unit tests. However, to ensure the integrity of our files are preserved throughout this process, our integration test will generate the files from scratch, send them to the correct companion computer, and then retrieve the file in order to check it for errors. If the file retrieved from the companion computer matches exactly with the one that was initially sent, the main part of our integration test will be considered a success.

3.3 Network-based communications:

Finally we want to make sure our upload and download via FTP are up to snuff. We send a configuration file to the drone to set up its flight plan. Upon arrival to the drone we want to check the file for correctness so we will be writing a small tester program to generate test files and send them to our upload method so we can verify that our upload is valid. To test our download file we will download the generated files that were previously uploaded and compare to assure that the download is working the same as the upload.

4 Usability Testing

Usability testing is the evaluation of an application based on the users the application is intended for. The goal of Usability Testing is to figure out how well the end user can effectively use and understand the software. The way this is generally tested is to set out a series of tasks for a new user to do and see if the user is able to accomplish those tasks set for them. As they try to accomplish these tasks, the reviewers who set up this testing take notes on how difficult or easy it is for the tasks to be accomplished so they can determine if and how something could be changed to better suit the users. Before we dive too deep into the details of our testing plan, we would like to start by explaining some variables that aided in these decisions. The first is who the software is for, our project is being developed for our client and to be used by him as well, and as of now he has told us that the UI (user interface) is not a concern until all of the functionality exists. Therefore the current UI has not been changed as it was being ported from his software into the flight planning software. Therefore we do not expect there to be any difficulties from our client, however we will still do testing to ensure that this is true. Since this project is tailored

to our client, he is the only person in which Usability Testing applies to and therefore will be our sole tester.
To begin talking about our plan, we will have our tester try and do a number of tests that focus on the main functionality of the software. Our tester will be our client due to what was stated earlier with him being the sole user for this product at this stage. The way we will gather feedback on these test cases will be to take notes as our client works through the task, as well as have him verbally say what he is doing as he does it so we can understand the thought process as well. After each test we will also talk with him to see his thoughts on any ways things could be changed to better fit his needs. We plan on having these tests on a Tuesday where we already have a meeting with our client and is early enough in the week in which if there were any changes needed we could get them back by the end of the week. Due to the current COVID-19 situation, all of our meetings are now taking place online, therefore we can no longer have him use one of our development laptops to test the software so we will have to send it to him and help him get it setup on his machine.
The first test we will have our client do is create a configuration file, the second test would be to import a previously created file. We will also have a test for uploading and downloading files from the ftp server, and our final two tests will be to have our client start and start the User Data Protocol (UDP). These tests cover the biggest three parts of our software and are all vital to the end result.
We believe that the Usability Testing can be completed within one week due to the lack of UI changes as well as our client's want for the software to work well first and look nice later.

5 Conclusion

In conclusion, we will be heavily relying on Unit Testing, Integration Testing, as well as Usability Testing to ensure that we develop a bug-free application that our client can rely on. Our client has often stressed that this application needs to not just work, but be functional and reliable. So we know how important this is to him and therefore important to us. The reason our Unit Testing is going to work is because we are running numerous tests to ensure that all of the data is correct and valid. The reason our integration testing is going to work is because We are covering the major venues of failure in our code and making sure that our current code is either robust or we can make the necessary changes to ensure our program functions as expected. The reason our Usability Testing is going to work is because our software is specifically being developed and used by our client. With the lack of general UI changes to focus more on functionality and reliability, we are sure that doing a small amount of testing with our client is best. Overall we are confident in our testing choices and are happy with the progress we have made so far.