Final Report

Version 1.0

Team Members: Joseph Griffith, Robert McIntosh, Brandon Samz, Corban Stevens

Project Sponsor: Gary Matsch and Harlan Mitchell

Faculty Mentor: Austin Sanders

Created: 5/09/18

Table of Contents

1. Introduction

Every day, over one hundred-thousand flights are scheduled across the globe. With so many flights occurring everyday, accidents are inevitable. In 2016 alone, there were sixty five accidents on commercial airlines resulting in ten deaths. Accidents can happen for a number of reasons, most of which are out of the control of aircraft operators and engineers. However, it is the responsibility of the aircraft engineers and operators to minimize the risk of failure as much as possible. We are focusing our concerns on the needs of the engineer, and any engineer will inform the best way to see a problem in your system before it happens is to collect ample operation data.

Technology today is data driven, and, as software developers, data is paramount to building and maintaining successful solutions. Our team (BlueSky Group) has been tasked with assisting our client in their goal of obtaining more data. Gary Matsch and Harlan Mitchell with Honeywell Aerospace develop turbine engines and engine control systems for a myriad of private jets. These engines and their connected systems generate data every flight, data that is paramount to the reliability of their product. Currently, this data is downloaded from a computer that monitors the engine's performance through a wired connection. Honeywell Aerospace technicians periodically connect to this Engine Control Unit (ECU) and retrieve the data as often as they can. However, this process does not happen frequently enough. The cumbersome process of physically connecting to a computer and downloading this data on location greatly limits the amount of flight data that Honeywell can collect. Developing a wireless solution to this problem would allow Honeywell to collect ample amounts of data, but this solution would not bring immediate profit to their company. In other words, there is no business case to solve this problem. This is where our team comes in; our clients, Gary Matsch and Harlan Mitchell, see this as a very cumbersome and unnecessary process in an age when everything is wireless. To assist our client in proving to his superiors the importance of this solution, we at BlueSky Group will develop a prototype that solves this problem. With this problem solved Honeywell will be able to market their solution to anyone who owns a plane and sell this product. Currently there is already a wireless solution out there that performs a similar function, but it is bulky and still requires a computer and a lot of setup time. With our solution Honeywell will be able to complete with this other wireless model of data transfer and expand their already large client base.

Our prototype will take the form of a mobile app that uses Bluetooth to connect to the Engine Control Units, download their data, and process it. Currently, Honeywell technicians use an archaic Engine Management System called EEI to process flight data. Once they have downloaded the data from the ECU, this system displays the data in a way that can be analyzed

by the technicians. Our prototype will emulate the functions of this system on a mobile platform, allowing for ease of access to this data. Our prototype will allow for a secure and easy way to download flight data on a much larger scale meaning greater reliability for the products Honeywell Aerospace creates.

We will also ensure that our solution will meet the requirements specified in our requirements document. The key functional requirements that our solution will be able to satisfy are allow for easy access with both a smartphone a tablet to ensure that data can be access on different platforms as long as they are all running android. The engine data should also be able to be downloaded any place the plane has landed with only a smartphone or tablet necessary for the engine download. Our application will also be able to download all of the data from the ECU to ensure they we get all of the data that we need to represent an accurate picture of how the engine is performing. Our application will then allow the technician to review all of the data that was collected in a presentable fashion. The application will also communicate with the ECU over a Bluetooth connection to ensure the the engine data can only be downloaded locally and transferred in a secure measure.

Our key performance requirements are as follows. Our application will download data with a lossless protocol. This will ensure that when we are downloading and displaying our data that it is accurate and the technicians are not going to make mistakes due to misinformation. Our application will also be able to do the download within 1 minute. With this goal we will make sure that our solution is much better than the current version of EEI in which the download currently takes around 10 minutes and requires a lot of setup. Our application will also not report incorrect data. Data that is shown will be displayed in correct formats as expected by technicians, and it will be displayed in a way that is readable in understandable. Lastly, our application will be able to establish a connection between the phone and laptop in under 15 seconds. This window of time will once again ensure that our application will be better then the current version of EEI as it is now where a technician has to walk into the plane and manually connect wired into the plane in order to start the download.

The environmental requirements for our application follow as such. Our application will run on android OS. We chose this platform because it has been the easiest platform for our team to develop on. This being said our application will be able to run on all android devices. This will ensure that no matter who is running our application then if they have an android device they will be able to run our application regardless of what actual device it is. As specified by our sponsor our application will also be able to mimic the basic functions of EEI but in a different layout that is much more readable than what is currently available. This will allow technicians to read the data they collect much more easily than before. Our application must also be able to download at least 5 dummy data sets in order to compare the data on them. This would simulate the collection of multiple data sets over time, so being able to compare the data from these files is a must for technicians working on these aircraft.

2. Process Overview

In order to develop our application our team used a waterfall design process. This allowed us to finish the necessary steps it took to complete this process in the correct order and go back and make changes if needed. Our team first started by acquiring the requirements from our Honeywell sponsor. Once we had these requirements we began to work on our prototype. Once we were finished with our prototype we moved into full scale development of our application using the requirements we have previously gathered. Once we had completed our first version we continued to test and refine our design until we had something that we felt like our client would be happy to receive. When we were developing our application we used a few tools in order to help with the overall development of the project. The first major tool that we used was android studio. Using this was a big help to us because when it comes to making android apps android studio provided a lot of built in help for compiling the application as well as helping with the design of the gui used in the app. The other major tool that was used was GitHub. GitHub provided us with a way to have version control when developing our application, as well as having built in support with android studio making it very easy to upload different commits to our project.During development in order to make sure things were running smoothly our team met once a week as a team to develop and discuss tasks they had completed earlier that week. We also met once a week with our project mentor to make sure we were always on task, and we met once a week during the seconds semester with our project sponsor in order to make sure we were fulfilling all of the requirements that we had promised to fulfill.

3. Requirements

In this section, we will outline the functionality necessary to effectively meet our client's needs. All of the functional, performance-based, and environmental requirements will be expanded on and relate to the following set of top level domain requirements.

3.1 Functional Requirements

1. *Top Level Domain Requirement:* Engine download application is accessible to users with a smartphone or tablet.
    - 1.1. *Accessible with a smartphone:* Most users of the engine download application will be using a smartphone. Smartphones are much smaller than tablets, so less detailed data will be displayed.
        - 1.1.1. The application should have a navigation menu so that different screens can be easily navigated to.
        - 1.1.2. The application should support scrolling so that it can be navigated easily.
    - 1.2. *Accessible with a tablet:* Tablets allow the user to see more of the data without having to scroll or go to another window. They are also very helpful when the technician has to view graphs in detail or manipulate something on a densely

crowded screen.

1.2.1. The application should scale and take advantage of increased screen size on a tablet.

1.2.2. The application should have an expanded view of the graphs so that greater detail is displayed.

2. *Top Level Domain Requirement:* Engine data can be downloaded any time or place the plane has landed, with only a smartphone or tablet running the engine download application.

2.1. Application should be able to download flight data regardless of internet connection.

2.1.1. Download should be possible as long as the user is within 10 meters of the aircraft.

2.1.2. Application should list available engine control units to download data from.

2.1.2.1. Application should keep track of recently accessed engine control units.

2.1.2.2. Application should list the engine control units in a formatted string of numbers, which are randomly generated and assigned to each ECU.

3. *Top Level Domain Requirement:* Application should be able to download all of the data from the ECU.

3.1. Application should display progress of the download.

3.1.1. Application should verify when a download is completed.

3.1.2. If a download fails to complete the user should be notified.

3.1.2.1. If download fails to complete on the first run user should be able to restart the download.

3.2. Application should be capable of storing all of the engine's flight data.

3.2.1. If device does not have enough available storage to complete download the user should be notified.

3.3. Application should display the file size of the ECU's current available flight data.

4. *Top Level Domain Requirement:* Application should allow for review of engine data, with functionality similar to EEI.

4.1. The application should function similarly to the EEI software provided to the group.

4.1.1. The application should have engine data that includes the engine serial number, the aircraft id, the download date and time, engine position, and the operator who performed the download.

4.1.2. The application should include engine maintenance data that includes the number of maintenance conditions, number of exceedances, number of

events, number of chips, and the number of impending oil filter bypasses.

4.1.3. The application should include statistics on the engine and the ECU.

4.1.3.1. Information regarding engine usage and engine landing should be included.

4.1.4. The application should show data regarding different systems for aircraft takeoff, climb, and cruise.

4.1.5. The application should show a list of faults that occur with a timestamp so that the technicians will know when they occurred.

5. *Top Level Domain Requirement:* The application will be able to establish a secure connection with the laptop for testing using Bluetooth. In order to achieve this both the laptop and the application must abide by protocols established by Bluetooth. The laptop will act as a server that will provide the data that has been gathered from the ECU, and the the application will act as the client which is going to be receiving the information sent by the server.

5.1. *Establishing connection as a server:* Our laptop will operate as the server for our Bluetooth connection. In order to do this first it will have to maintain an open Bluetooth server socket. This server socket will listen for incoming connection requests, and provide a connected Bluetooth socket after a request is accepted. When the Bluetooth socket is acquired from the server socket the server socket will then be discarded in order to make sure that our laptop will not accept any more connections.

5.1.1. In order to get a Bluetooth server socket our application will call the method listenUsingRfcommWithServiceRecord(). This will automatically write a new service discovery protocol database entry on the device. The UUID of the app will also be included in the entry which establishes the basis for the Bluetooth communication

5.1.2. To start listening for connections, our program will call accept() on the server socket

5.1.3. Our program will then call close() in order to close the connection to make sure that other devices cannot join the same connection.

5.2. *Establishing connection as a client:* The application will operate as the client in this project. The application will initiate a connection with the laptop that is accepting an open connection on an open server socket. Our application will create a Bluetooth device object that represents the remote device. It will then use that Bluetooth device to acquire a Bluetooth socket and initiate the connection.

5.2.1. Our application will initialize a Bluetooth socket object that will allow our application to connect to a Bluetooth device. The UUID that is given in this creation must match the UUID that is used by the server or else the connection will not be allowed. This will be made sure because the UUID

value will be hard coded into the application

5.2.2. Our application will then establish a connection by calling connect() which will initiate a lookup to find a remote device with a matching UUID. if the lookup is successful then the connection is created.

5.2.3. If the method times out then it will throw an IOException which we will then handle by letting the user know a connection could not be completed.

3.2 Performance Requirements

This section details the performance requirements for this project. These performance requirements specify how the functional requirements are expected to perform in aspects such as speed, accuracy, etc.

1. Application downloads data with zero data loss or corruption.
   1.1. It is important that all engine data received is accurate so that the application does not misinterpret or fail to report important data.
      1.1.1. Bluetooth connections manage data using packets similar to TCP. As such, the underlying Bluetooth protocol handles this requirement by ensuring that each packet is received and is not corrupt, and resends the packet if not. However, during the test phase, this requirement may also be tested by manually hashing the download file on each end and comparing the hash.
2. Application downloads data within 1 minute.
   2.1. One of the main focuses of this application is to speed up the data download process. As such, we want the application to download the data as fast as possible, while still providing some leeway due to unavoidable connection issues.
      2.1.1. This time was determined using the 5 MB max file size of the data to be downloaded and average download speeds of Bluetooth 3 or 4. Verifying this requirement is met should involve averaging download times from multiple downloads that minimize outside factors (signal interference, phone distance from microcontroller, etc).
3. Application reports no incorrect data.
   3.1. As previously mentioned, it is important that the application is highly accurate with data to ensure that issues aren't mistakenly diagnosed or missed, as this could result in major engine failure or unnecessary maintenance.
      3.1.1. As download data will be generated for testing purposes, the team should be able to verify that the application properly displays data by manually parsing it and comparing with what is displayed in the application. This

requirement is verified by ensuring that all data in the testing process is correctly reported.

4. Application will establish a connection between the laptop and the device in under 15 seconds.
    4.1. In order to have an application that is focused on speeding up the data recovery from the ECU, speed is a key factor in everything that this application needs to do. When it comes to establishing the Bluetooth connection, the application should be able to establish the connection within 15 seconds.
        4.1.1. The application will be able to establish a connection within 15 seconds because we will be operating our Bluetooth connection in the way that is directed by the Bluetooth documentation. This will ensure that the application is establishing a connection as soon as possible because it is operating in the intended way.
        4.1.2. The application will also be able to establish a Bluetooth connection in under 15 seconds because the application will be able to pair with the laptop because the laptop will be looking for a Bluetooth socket which can only be provided by our application. This Bluetooth socket will be initiated once the application is prompted to connect to a device. This means that the devices should automatically pair once they have been once they have been initially paired with each other.

3.3 Environmental Requirements

This section details the environmental requirements for this project. These are requirements that are imposed upon the project due having been specifically asked for by the client or as a result of chosen technologies.

1. The application will run on Android OS.
    1.1. As specified by the project, the application must be mobile. This means that it must be able to run on a mobile phone. As not all members of the team have Mac computers, and thus are not able to write iOS applications, we have decided to write the application for Android OS.
2. Application should be able to run on all Android devices.
    2.1. Application should transfer user interface seamlessly between devices
        2.1.1. Application interface should be scalable between tablets and mobile smart phones.
    2.2. Application interface should be able to display in both portrait and landscape mode.

3. As specified by our project sponsor the application will be able to mimic the basic functions of EEI in a different layout that is newer than what is currently being used.
    3.1. The application will mimic the functionality of EEI. From our analysis of the current EEI, the most important feature that we need to implement is the functionality to analyze the data and identify if there are any immediate faults that need to be recognized. For example, if a sensor has failed and is in need of repair the EEI will be able to recognize that and report that to the user.
        3.1.1. The EEI is able to identify when a sensor has failed and is in need of repair because, if a sensor is broken, then it will not be reporting data. When this is happening, it either means that the sensor or its connection needs to be repaired or that it is just broken altogether. The application will be able to identify these conditions and report them back to the user.
    3.2. The application will be able to display the ECU data to the user.
        3.2.1. The application will display data to the user through use of a library with the ability to display the data in a graphical format that will be appealing to the user.
4. At least 5 dummy engine data files should be created and be able to be downloaded by the application.
    4.1. The file type must be consistent across these 5 files.
    4.2. The fields and units of measure should be consistent with those found in an actual .dld file.

4. Architecture and Implementation

In this section we will illustrate the finer points of our systems architecture. The following sections are separated by the GUI pages within our application. Each module defines how the information is displayed and where its information is coming from. In this manner we will present the inner workings of our application. Followed by an overview of our implementation our project.

4.1. Module: Faults Page

The faults page is responsible for displaying faults read from the ECU data in an easy-to-read table format. This table should display the FF Fault ID, description, and time of each fault. Generally, this module interacts with the FileParser module by reading the list of faults that are stored into a Fault class. This list of faults is then used to build the table.

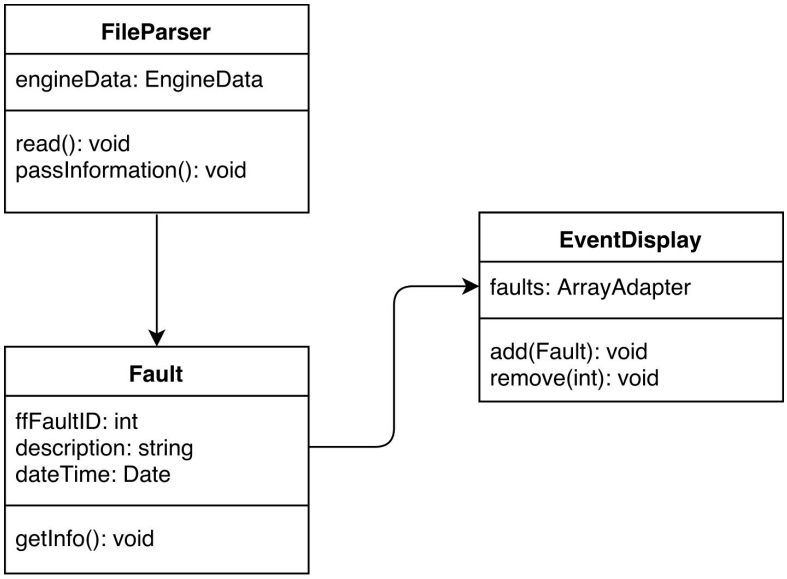

Figure 3: UML Class Diagram for Faults page.

Due to the nature of the faults screen, there is not much the user can interact with or modify. In general, the public interface is the EventDisplay, which displays fault data in a table format for the user to view. These faults are displayed using an ArrayAdapter, which will create the table that displays the FF Fault ID, description, and time of each fault. Should the list be long enough, users will be able to scroll through the list to view the full list of faults.

4.2. Module: Events Page

This component of the application will fill the role of displaying all of the events recorded by the ECU. These events are one of the major things that technicians look at when they are trying to diagnose what is wrong with an aircraft. These events are anything that the ECU deems a significant change to the engine while it is running. This means that if there is a change in anything that is not recognized as “normal” by the ECU it will flag what is happening and start taking information down about what is happening the time it started and the time it ended. This component will fit in with the larger architecture because it is one of the main pages that we will have available for users to utilize while they are analyzing data from that they have just downloaded. This module will also receive data from the file parser class which will be given by way of passing a message and creating a new class every time a new one is read from the file.

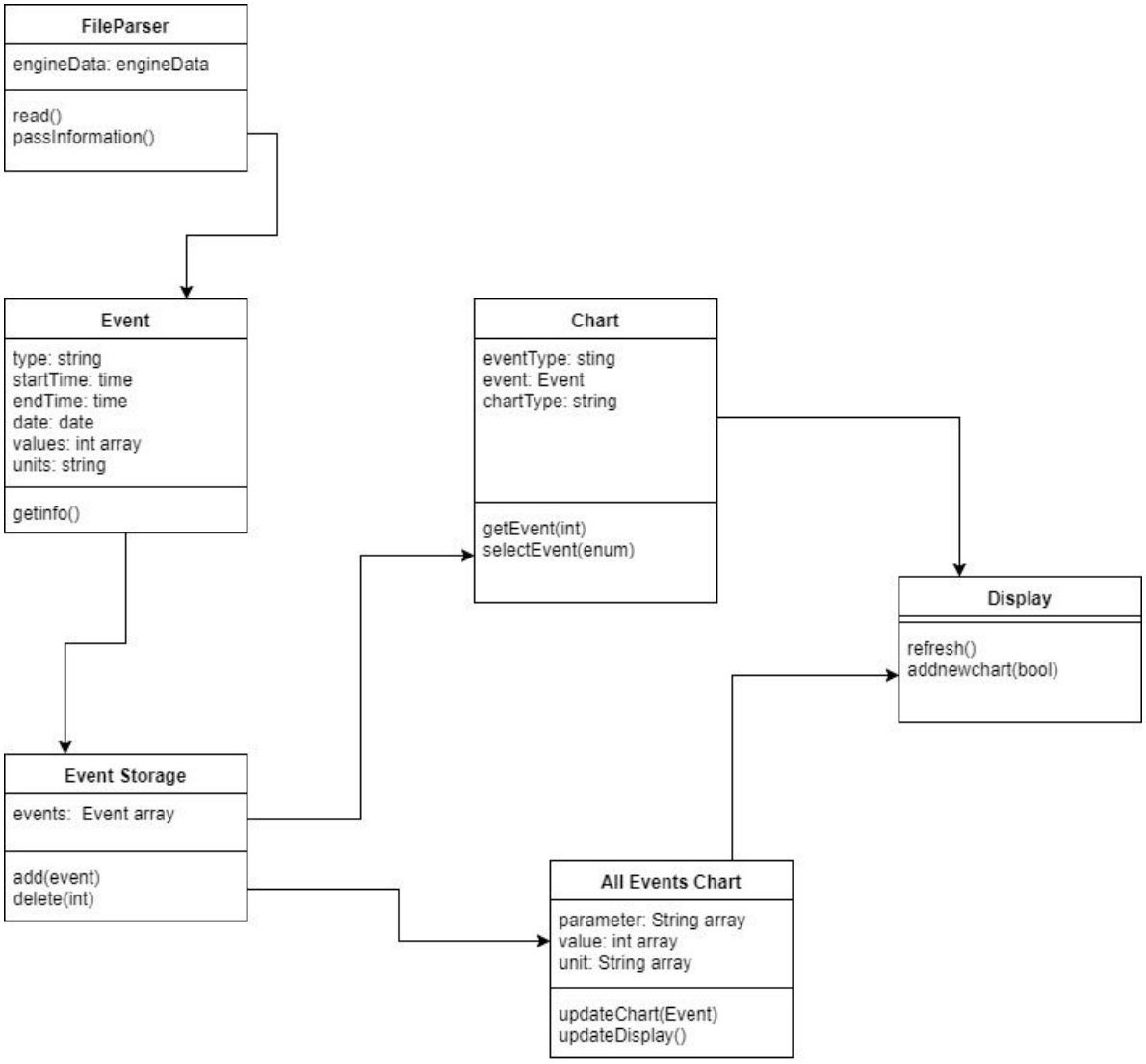

Figure 4: Events UML

Our public interface that the users interact with will be with the display, the chart, and the all events chart. When it comes to interacting with each of these, the user will not be inputting any information of their own on this page. The user will only be able to select what they want to do based on the options that they are given. This means that the parameters for any of the functions involving display will only have a few set options. The only interaction with the All Events Chart will be a scroll bar. This chart just displays all of the events meaning what they are, the abnormal value received, and the units that go along with that value.

The next part of the module that the user will be able to utilize is the individual charts that display a graph of the event. These graphs will display the value in question over the time of the event occurring. The chart will start blank and the user will be able to select from the available events which one they want to display. This will use the selectEvent function. This function will have an enumeration of all the available events for the user to choose. Once the user chooses one the chart will then display this data as a graph to the user.

The last part of this module that the user will be able to utilize is the ability to select the chart currently being displayed. Since we do not know if the user running our application will be on a mobile phone or a tablet this will allow the user to select what information they need to view. The user will interact with the display class here and choose if they want to view a different chart if they do then the display will refresh bringing up that chart to the user.

4.3. Module: Summary Page

The summary page offers Honeywell technicians with a basic overview of engine data and engine maintenance data. The summaries page also allows the technicians to add comments to the data from this page,

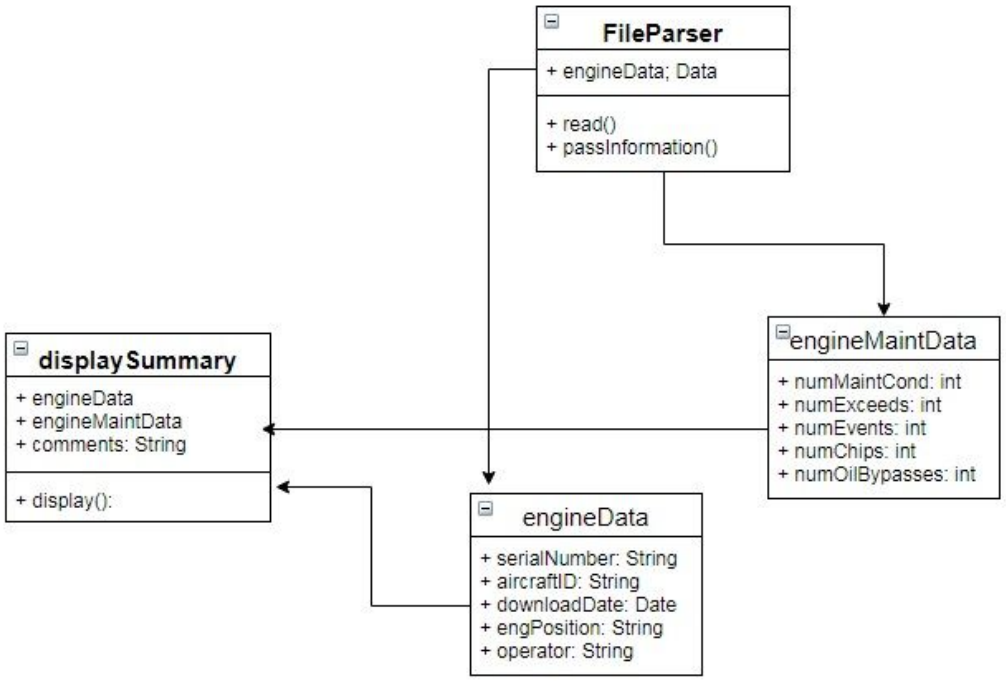

Figure 5: Summary UML

The summaries page contains two tables, one for engine data and one for engine maintenance. The engine data is akin to a configurations page, it contains info on the type of engine and aircraft the technician is analyzing. The engine maintenance table contains the real summary of the page. From this page the technician can instantly see if any serious problems occurred during the flight. If the technician notices that the number of exceedances and events is more than zero, he knows to visit those pages and investigate further

4.4. Module: Bluetooth Connectivity

The Bluetooth module for our app is responsible for connecting to the Linux Virtualbox as well as finding and displaying the local available Bluetooth connections. It uses one small utility file named ConnectionThread. When the user interacts with this page they see a list of available connections and connections remembered by their device. They can tap on any of the devices on the list to establish a connection.

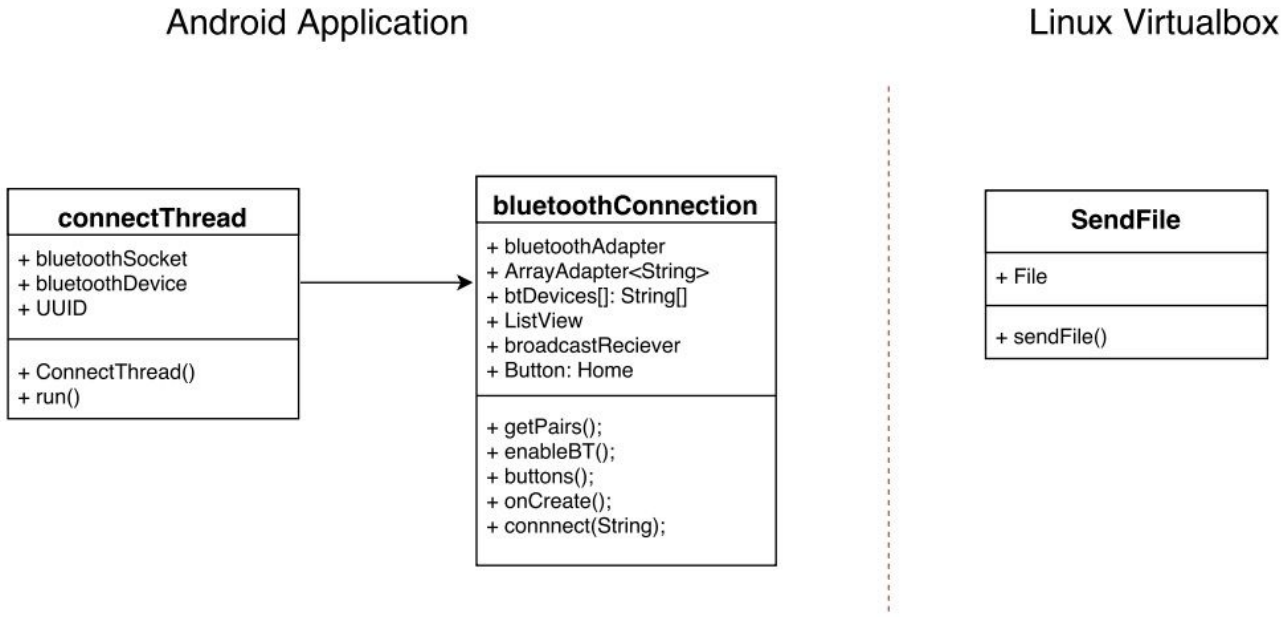

Figure 6: Bluetooth Connection Module

Before the BluetoothConnection class can initiate a connection there are a number of steps that need to be taken. First, the class calls the function enableBT() which prompts the user for the apps permission to use the host devices Bluetooth capability. The next step is to fill the list with the devices that the host device has paired with in the past. The function getPairs() accomplishes this task by querying the host device for an array of remembered devices, when the host device returns the device info the getPairs() function adds them to the ArrayAdapter which is then displayed in the list. The final step of setting up Bluetooth is to search for new devices, this is accomplished by the broadcastReciever. As the broadcastReciever finds new devices it passes them to the onRecieve() function which adds them to the list in a similar fashion to the getPairs() function. When a user selects a device from the list the buttons() function passes the devices name and mac address to the ConnectThread() function. This function stores the host devices mac address and uses these two addresses to establish a connection by calling the run() function. Finally, when a connection is established the Linux Virtualbox automatically sends its file via the Bluetooth socket so it can be stored and parsed.

4.5. Module: Exceedances Page

The exceedances module in our app will show if a parameter for the engine reached above a certain threshold. Technicians will be able to see what these exceedances are and view the details

regarding them. These details include the time and date that an exceedance occurred at and duration that exceedance occurred. The data for this page is obtained from the FileParser.

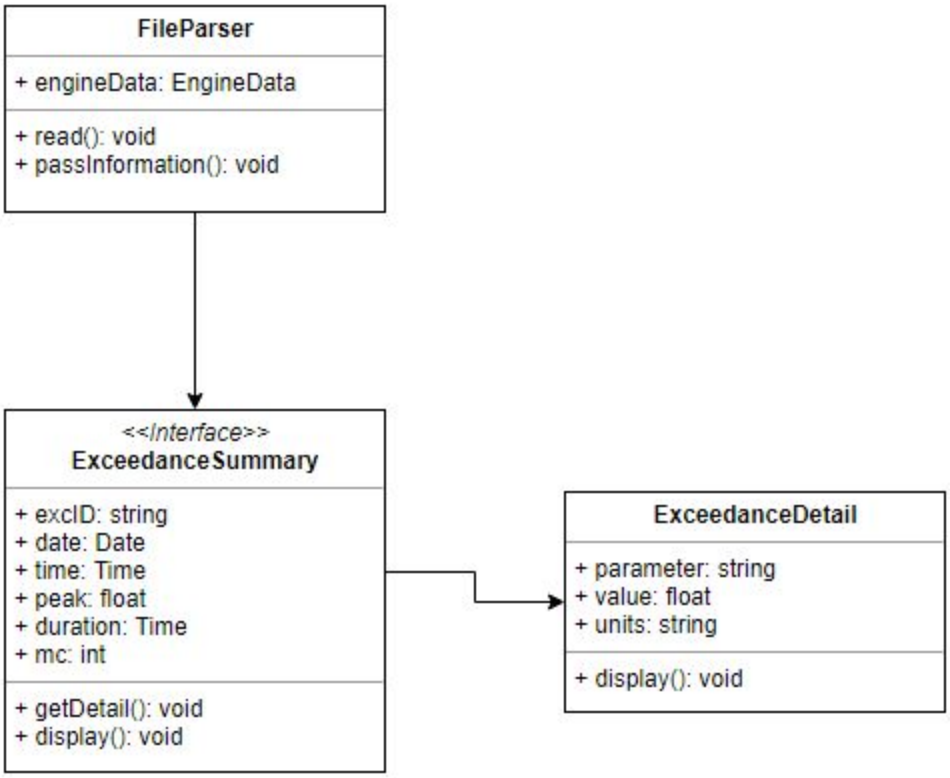

Figure 7: Exceedances UML

There is not that much that can be changed about this page. The only interactable object on this page will be for if a technician wants to view the details of an exceedance. There will be a scrollbar on the ExceedanceSumary and the ExceedanceDetail pages if the items for these lists become long enough.

5. Testing

The purpose of this section is to introduce tests that could be used to test how well data is transmitted and used throughout the different modules that make up the application. The testing here will focus on if the data changes at all when it is sent in its original form to the various modules that are in the application. Successful transmission between modules will show that all of the modules are working correctly together when they have to handle the same data.

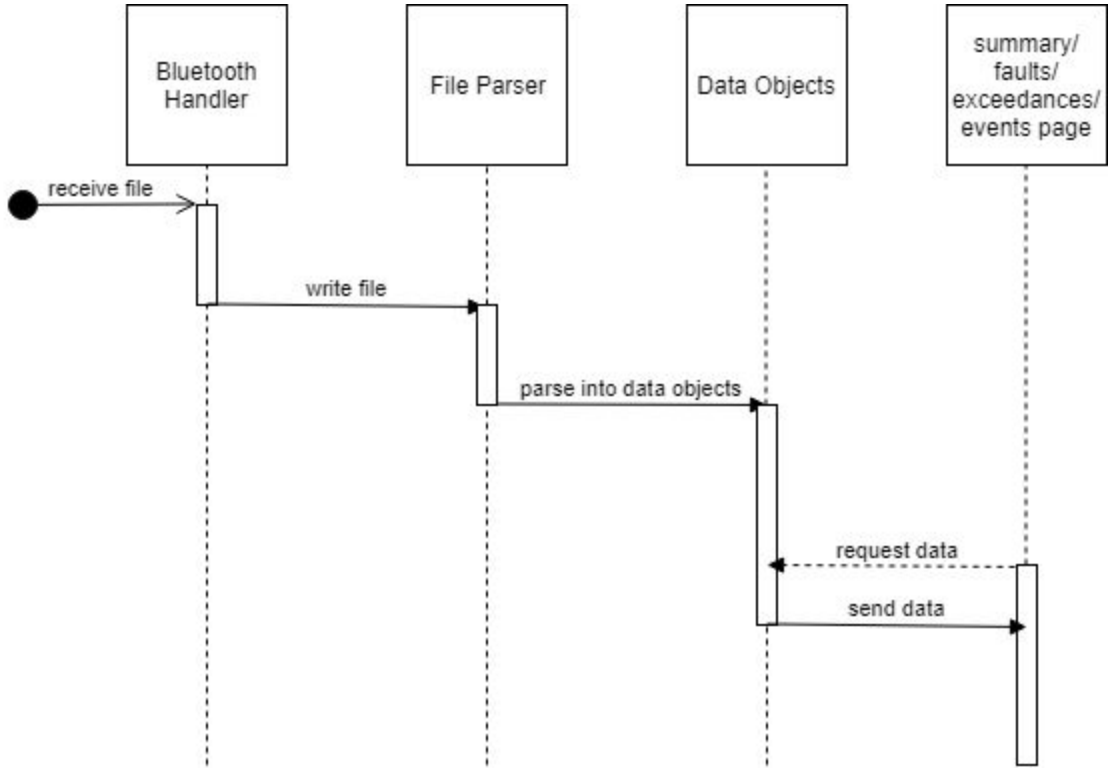

Figure 8: Sequence Diagram of Data Transfer

Bluetooth Handler: The Bluetooth handler will receive the file that will be used throughout the application as its source of data. This file will be sent over a Bluetooth connection to the file parser. At this stage, testing will focus upon ensuring the data file was not corrupted, fragmented, or incomplete. Corruption tests will entail catching any I/O Exceptions,File Not Found Exceptions, that may occur. Fragmentation tests will be conducted via ensuring that the data file has not reordered its data (i.e. does engine maintenance data follow engine data). Finally we will test the completeness of the data file by comparing the downloaded file size to the original file size.

File Parser: The File Parser receives the file from the Bluetooth Handler. This file will then be parsed into the predefined Data Objects for easy use in the other parts of the application. To test this part of the application, the tester will check to see if the application has parsed the data correctly from the corresponding yaml file. This will entail catching any IO Exceptions, Null Pointer Exceptions, or Class Not Found Exceptions that may occur.

Data Objects and Pages: The Data Objects will receive the parsed data from the File Parser. It will then wait for the various other pages to request data from the Data Objects. The Data Objects will then send the requested data to the appropriate page. To test their validity unit tests will be run on the input for each page.

Upon parsing the files the corresponding pages are populated with the necessary data. Because the methods of displaying the data are nearly identical across all four pages we will focus our unit testing upon the data entry into these pages, using Android Studios built in unit testing functionality. An example of the unit tests performed on the summaries page is depicted in figure 9. On the left hand column is a list of invalid data for each entry, in the center column is the correct data format, and the right hand side displays what the application returns on an incorrect entry.

	Test Input	**Valid Input**	**Test Output**
Engine Serial	ASCII Character String Null Value	*Type:* Int *Ex:* 84576285	"Error: Invalid input type, check data file
Aircraft ID	Special Characters Malformed String	*Type:* String *Ex:* C7-ABA (Registration Prefix - Designation)	"Error: Invalid input type, check data file
Date - Time	ASCII Characters Strings Int Double Float Malformed Date Obj	*Type:* Date Obj *Ex:* 2018-02-19 14:10:43 (Year, month, Day - Hour, Minute, Second)	"Error: Invalid input type, check data file
Engine Position	ASCII Characters Int Double Float	*Type:* String *Ex:* Left, Right, Center	"Error: Invalid input type, check data file
Operator Name	Int Double Float Special Characters	*Type:* String *Ex:* John Doe	"Error: Invalid input type, check data file

Figure 9: Summaries Page Unit Test

As a final iteration of our testing we decided to implement usability testing. In order to conduct usability testing, the team tested various groups with varying levels of knowledge of technology. Each test group was given the same scenario, involving tasks to be accomplished and data to be retrieved with the application. Afterwards, a survey was conducted asking users to provide feedback on various portions of the application. Along with this, the team will also record and

review footage of users interacting with the application, to view reactions and issues in the moment.

The team used the following scenario to ensure that users were able to use the application and access all functionality:

- *As users of this application, you are tasked with taking on the role of an engine technician to identify any major issues in the engine data, as displayed by the application. Complete the following tasks and provide feedback in the survey at the end.*
    - *First, connect to the test engine control unit, labeled "Test ECU".*
    - *Navigate to the summary page and determine the number of exceedances and events recorded in the data, as well as the engine's serial number.*
    - *Navigate to the exceedance page, identify a single exceedance by its ID number, as well as its parameter, value, and units.*
    - *Navigate to the faults page and identify a single fault by its ID and description.*
    - *Now, navigate to the events page and select a single event to view a chart. Identify this event by its ID number and provide the start and end time for this event, as well as values at these times.*

After this scenario has been completed, the team provided the test user with the following questionnaire:

1. *Describe the overall experience in regards to page navigation. Was navigating between pages easy, difficult, or somewhere in between? What features would help to improve navigation between various pages?*
2. *Describe your experience with data display. Did you feel that important information was sufficiently highlighted? Was there information that didn't seem important that was too prominently displayed? How did you know when information seemed important?*
3. *Provide any thoughts in regard to user interface. What did you like? What did you not like? What features or changes to the UI would you like to see?*
4. *Please discuss any further changes or improvements you feel would be beneficial to the application.*

With this scenario and questionnaire, the team will pick multiple members from two distinct groups to test and provide feedback on the application. These groups will consist of students in the Advanced User Interfaces course, and other non-Computer Science CEFNS students.

6. Project Timeline

With the module design in mind, the team was able to plan out the schedule for implementing each module. The focus was to ensure that important modules and modules that are relied upon by others were implemented first. The second focus was to schedule tasks that can be worked on in parallel as much as possible, in order to maximize group productivity. Below is the Gantt chart outlining the implementation plan and timeline.

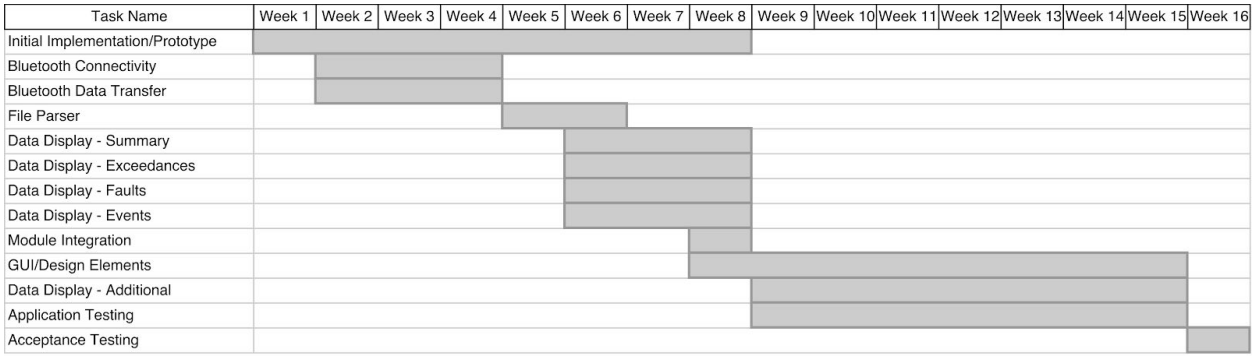

Figure 10: Gantt Chart for Implementation Plan Timeline

The initial eight weeks of the semester consisted of implementing the application into a working prototype. The initial focus (spanning Weeks 2-4) ensured that Bluetooth connectivity and Bluetooth data transfer worked correctly, as these modules cannot be tested via emulator. Once the Bluetooth functionality was working, the team finalized a format for data files so that the file parsing module could be implemented starting in Week 5.

After work on the file parsing had begun, the team was able to begin working in parallel on the various data display screens which spanned Weeks 6-8. As these screens only received data from the file parse module, and not each other, they could be implemented in parallel without any conflict. Once these various pages where working, we began to integrate the various modules in Week 8, which consisted of ensuring that each page was accessible and that data was being passed properly to each module.

Once basic functionality was implemented, the team began refining the GUI and other design elements beginning in Week 8 and continuing through Week 15. For the most part, this consisted of tweaks that made the application look and feel better, so they had a lower priority than modules that provided actual functionality. Similarly, the team began extending and adding functionality beginning in Week 9 and continuing through week 16. Functionality added in this stage was not the team's main focus and was added (time permitting) throughout the testing stage.

In Week 9, the team began the testing phase, which will last through Week 15. The application was thoroughly tested to ensure that all requirements are met. Each module wal also tested to ensure that all output was accurate and provided expected results. Testing was be performed while keeping acceptance testing (Week 16) in mind, to ensure the final product passed the acceptance testing.

7. Future Work

In order to improve the application, there are a few key features that could be implemented or improved in the future.

The team's initial focus was the Bluetooth data transmission and display. As such, the GUI could be vastly improved from this initial implementation. Currently, the application displays data in a very basic display. Improving this display would allow the technicians who will be using this application to navigate more easily and find the necessary data faster.

The team also focused on only displaying data that seemed to be the most important. There is, however, more data that could be displayed. Implementing this extended data display would be useful in the future to ensure that all necessary information is displayed.

Overall, there are still more features that can be implemented in this application. The goal is to ensure most of the important information and functionality from EEI is implemented in this application.

8. Conclusion

Many people rely on aircraft to get from one place to another in a timely manner, so making sure that they are safe to use is a critical task for many people. Engineers and maintenance personnel need to make sure that the airplanes that they are working on do not have any mechanical failures or faults that could result in an accident. To do this, technicians currently use a piece of software called EEI to help identify problem areas. Unfortunately, this piece of software is old and cumbersome. Blue Sky Group will provide the owners of EEI, Honeywell, with a mobile application prototype to aid with the modernization of EEI. The main task that Honeywell wants our team to accomplish is to be able to download a file from a Linux computer onto a mobile device using Bluetooth. To accomplish this task, our team will be using a python script on a Linux virtual machine that will send a file over Bluetooth. This file will then be received by a mobile device running android. This file will then be parsed and the data that it contains will be used to create graphs and to fill tables so that airplane technicians will be able to easily read the data and figure out what is wrong with the aircraft.

9. Appendix

- Hardware

Our team developed on a mix of windows and mac machines with the only minimum requirement to develop for our application being the ability to run android studio. As long as the machine you are on is able to run android studio you are able to work on this application.

- Toolchain

- Android Studio: This was the most important tool that our team used when developing. As we used android studio to do all of our coding and compiling. We did not use any external plugins or addons, so as long as you are running the most current version of android studio you will be able to use that to work on this project.
- BlueZ: This is the tool we used in order to let our python script use Bluetooth from the linux virtualbox. Having this library allows python scripts to set up Bluetooth server sockets which is how we were able to get our application to connect to the virtualbox and then have the virtualbox send the data over Bluetooth.

- Setup: for android studio
    a. Install the latest version of android studio and run it. Make sure to set everything up to the default values so no special installation is required.
    b. Download a copy of our project from the GitHub given to project sponsor
    c. Go into android studio and open up the project that was just downloaded
    d. Save the project in a directory of your choosing
    e. Congratulations you are ready to start developing on our application
- Setup: for linux virtualbox
    a. Install virtualbox by oracle and use default settings
    b. Download the latest version of ubuntu Linux
    c. Install that version of linux onto your virtualbox
    d. Setup your linux account with personal preferences
    e. Ensure pip is installed (sudo apt install python-pip)
    f. Ensure python distutils are installed (sudo apt-get install python-dev)
    g. Ensure BlueZ is installed (sudo apt install bluez)
    h. Ensure libbluetooth-dev is installed (sudo apt-get install libbluetooth-dev)
    i. Install PyBluez (pip install pybluez)
    j. Edit the file using “sudo gedit /etc/systemd/system/dbus-org.bluez.service” and append “-C” to the line ending in “bluetoothd”.
    k. Reboot run the command “sudo sdptool add SP”
    l. Congratulations your environment is set up on your linux virtualbox

- Production cycle
    a. Pull the latest version of the application from the GitHub
    b. Make a change to the code
    c. Plug an android testing device into the machine being used via a cable
    d. Make sure usb debugging is turned on on the device
    e. Go up to the run tab
    f. From the deployment window select your device for testing
    g. Then hit run

h. Note if the app compiles and builds correctly if it does not then fix the error if it does run then continue to test your change
i. In the terminal window you can see what happens if the app crashes or if any errors occur
j. Once you are satisfied with your change you can commit your version of the project to the GitHub so that others may pull it down and continue development.