Software Test Plan Document

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: 4/5/2018

Revised: 4/8/2018

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 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. 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 wired solution that performs a similar function, but it is bulky and still requires a computer that takes a lot of time to set up. 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 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 as follows:

- Allow for easy access with both a smartphone and tablet.
- Ensure that data can be accessed on different platforms as long as they are all running android.
- Download all of the engine data any place the plane has landed with only a smartphone or tablet necessary for the engine download.
- Allow the technician to review all of the data that was collected in a presentable fashion.
- 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. Unit Testing

The bulk of our application passes information via file parsing. Upon parsing the files the corresponding pages are populates 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. The following section outlines the parameters of valid data entry for each page and auxiliary elements they may contain.

Summary Page - The summary page receives its data from the file parser module and displays two simple tables with an overview of information. This information allows the technicians to investigate which pages they should investigate further. Tables 1 and 2 outline the boundaries of data entry for the two tables displayed on the page.

	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

	Test Input	**Valid Input**	**Test Output**
Number of Maintenance Conditions	ASCII Character String Special Characters Null Values Double Float	*Type:* Int *Ex:* 10	“Error: Invalid input type, check data file
Number of Events	ASCII Character String Special Characters Null Values Double Float	*Type:* Int *Ex:* 10	“Error: Invalid input type, check data file
Number of Chips	ASCII Character String Special Characters Null Values Double Float	*Type:* Int *Ex:* 10	“Error: Invalid input type, check data file
Number of Oil Bypasses	ASCII Character String Special Characters Null Values Double Float	*Type:* Int *Ex:* 10	“Error: Invalid input type, check data file
Number of Exceedances	ASCII Character String Special Characters Null Values Double Float	*Type:* Int *Ex:* 10	“Error: Invalid input type, check data file

Table 2: Equivalence Partition of the Maintenance Summary Table

The team plans to test the entries into these pages and trigger errors upon malformed data entries with a message informing the user of an issue within the data file. In this way the team can ensure that the tables are populated with valid data.

Exceedances Page - The exceedances page displays information inside a dynamic list that expands in parallel to the number of exceedance entries inside the parsed data file.

Upon selection of an exceedance a pop up containing further detail on that exceedance is displayed. Tables 3 and 4 outline the boundaries of data entries for the list items and their popups.

	Test Inputs	**Valid Input**	**Test Output**
Exceedance ID	Int Double Float Malformed Strings	*Type:* String *Ex:* EX901	“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”
Peak Value	ASCII Character String Special Characters Null Values Double Float Negative Numbers	*Type:* Int *Ex:* 10	“Error: Invalid input type, check data file”
Duration	ASCII Character String Special Characters Null Values Double Float Negative Numbers	*Type:* Int *Ex:* 10	“Error: Invalid input type, check data file”
Maintenance Conditions	ASCII Character String Special Characters Null Values Double Float Negative Numbers	*Type:* Int *Ex:* 10	“Error: Invalid input type, check data file”

Table 3: Equivalence Partition of Exceedances List

	Test Inputs	**Valid Input**	**Test Output**
Value	ASCII Character String Special Characters Null Values Double Float Negative Numbers	*Type:* Int *Ex:* 10	"Error: Invalid input type, check data file"
Description	Int Double Float Malformed Strings	*Type:* String *Ex:* This is an exceedance	"Error: Invalid input type, check data file"
Units	Int Double Float Special Characters Malformed Strings	*Type: String* *Ex:* PSI	"Error: Invalid input type, check data file"

Table 4: Equivalence Partition of Exceedance List Detail Popup

Similar to the Summaries Page the team plans to test the Exceedances list and their popups by implementing unit tests upon the entry of data into the page. Again an error message will be displayed to the user upon malformed data entry.

Events Page - The events page contains line graphs detailing the date, time, type, value, and units of an event. For example, say an oil bypass event was triggered. This page would display the duration and PSI over this duration of the oil bypass. Table 5 outlines the boundaries of data entry for that graph.

	Test Inputs	**Valid Input**	**Test Output**
Event Type	Int Double Float Malformed Strings	*Type:* String *Ex:* FUEL_LEAK	"Error: Invalid input type, check data file"
Start Time	ASCII Characters Strings	*Type:* Date Obj *Ex:* 2018-02-19	"Error: Invalid input type, check data file"

	Int Double Float Malformed Date Obj	14:10:43 (Year, month, Day - Hour, Minute, Second)	
End 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”
Values	ASCII Character String Special Characters Null Values Double Float Malformed Array	*Type:* Int[] *Ex:* {10,12,5,4,3}	“Error: Invalid input type, check data file”
Units	Int Double Float Special Characters Malformed Strings	*Type: String* *Ex:* PSI	“Error: Invalid input type, check data file”

*Table 5:*Events Page Equivalence Partition

Faults Page - The faults page displays a list of failures. Faults typically indicate a catastrophic failure within the system. Their description can provide insight to the cause of the problem and inform the technician on where to investigate further. Table 6 outlines the boundaries of data entry for this page.

	Test Inputs	**Valid Input**	**Test Outputs**
Fault ID	Int Double Float Malformed Strings	*Type:* String *Ex:* EX901	“Error: Invalid input type, check data file”
Date - Time	ASCII Characters Strings Int Double Float	*Type:* Date Obj *Ex:* 2018-02-19 14:10:43 (Year, month, Day - Hour, Minute,	“Error: Invalid input type, check data file”

	Malformed Date Obj	Second)	
Description	Int Double Float Malformed Strings	*Type:* String *Ex:* This is an exceedance	“Error: Invalid input type, check data file”

Table 6: Faults Page Equivalence Partition

4. Integration 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.

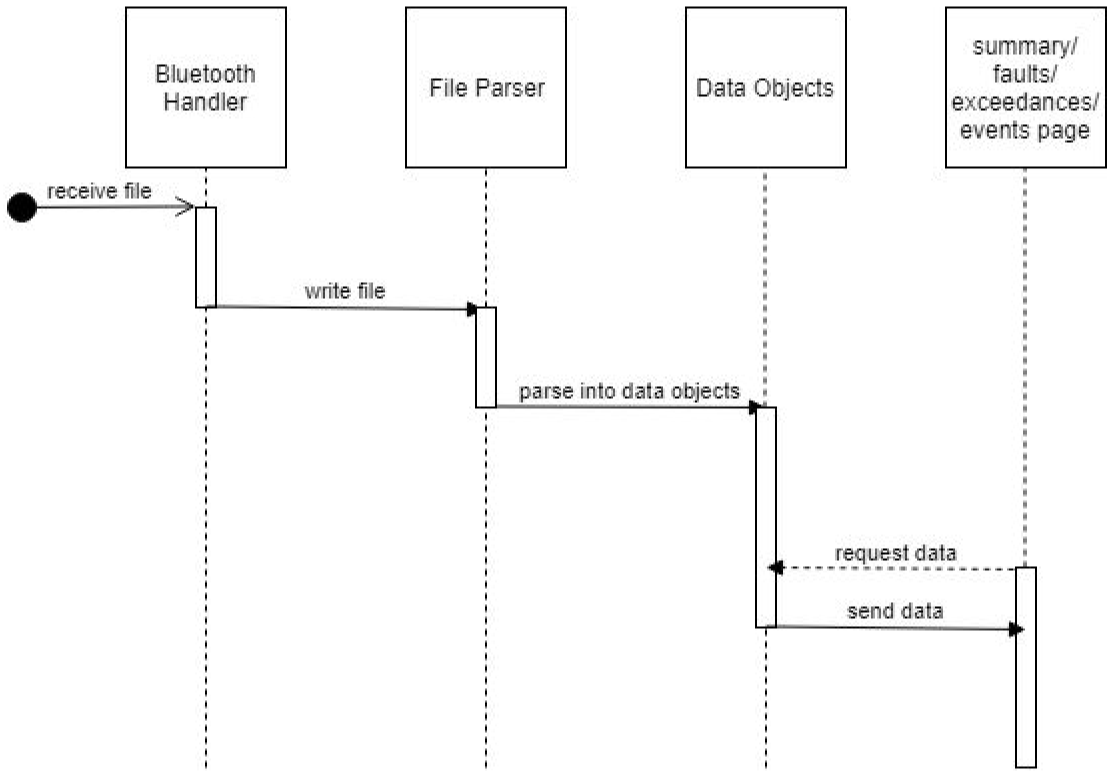

Figure 1: Sequence Diagram of Data Transfer

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 the aforementioned unit tests will be run on the input for each page.

4. Usability Testing

This section serves to outline testing to ensure users are able to access and use all of the appropriate functionality provided by the application. Unlike other sections, where testing was focused on output of specific modules and code robustness, testing here will focus on user interaction. As such, testing will ensure that users are able to easily connect and retrieve a download file via Bluetooth and find various data, as specified in specific tests.

In order to conduct usability testing, the team will test with various groups with varying levels of knowledge of technology. Each test group will be given the same scenario, involving tasks to be accomplished and data to be retrieved with the application. Afterwards, a survey will be 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 will use the following scenario to ensure that users are 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 will provide 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 three distinct groups to test and provide feedback on the application. These groups will consist of client/Honeywell members, students in the Advanced User Interfaces course, and other non-Computer Science CEFNS students.

The Honeywell test group should consist of both engineers who work with our client, as well as actual technicians who have a use for this application. Engineers working with our client will be able to provide feedback in regards to features and tweaks that may help the team that will be implementing our application officially at Honeywell. Additionally, technicians who use the current EEI application are crucial for testing, as they are able to comment on features that they find most useful while using the application, being the target group for this application.

Students in the Advanced User Interfaces course would also prove helpful, as our application is a UI based application. The goal is to streamline the data download process and provide useful information to technicians very simply. As such, this group will be able to provide useful feedback in regards to UI layout, navigation, and other pertinent information.

The final group will consist of non-Computer Science CEFNS students. These students will show some proficiency in application use, but will not sophisticated UI design knowledge. These students will be able to provide in regards to user navigation and ease of use. As these users will not be looking for specific features within the application, we anticipate to gain feedback relating to how well the application guides users and how easy it is to navigate.

With this plan for usability testing, the team anticipates to gain valuable user feedback, which is important for an application whose purpose revolves around a user interface. This testing should also allow the team an opportunity to polish the design aspects of the application, while still ensuring that core functionality and features are highlighted within the application.