Design Document

Engine Data Analytics Tool

Sdmay 20-06

Client:
Collins Aerospace

Faculty Advisor:
Lofti Ben-Othmane

Members:
Zak Frisvold
Thomas Haddy
Ryan Radomski
Will Sartin
John Powen
Jamie Raught

Team Website:
http://sdmay20-06.sd.ece.iastate.edu/

Executive Summary

Engineering Standards & Design Practices

- Secure and ethical coding
- No Free Software Foundation Licensing
- No Dead Code
- No Compiler Optimization
- MVVM Design Pattern

Summary of Requirements

- The Trend Data Analyzer (TDA) shall be a Windows compatible executable file
- The TDA shall accept any .csv and .xls files as input
- The TDA shall recognize if the .csv input file is valid and provide a notification on failure
- The TDA shall parse the input file into selectable data fields
- The TDA shall convert the ASCII data into ARINC 429, Network Data Object (NDO), and MIL-STD-1553
- The TDA shall provide capability to create user defined graphs, charts, and tables
- The TDA shall provide data field names for predefined data per the EIDS_DCU_Collins_ICD.xlsx
- The TDA shall provide capability for user defined data field names
- The TDA shall allow the user to export the parsed data to either .csv or .xls
- Prior to exporting the parsed data, the user shall be allowed to filter what's included in the data set
- The TDA shall allow for playback of trend data once imported
- The standard playback shall be sequencing the data points at a 1 Hz update rate
- The TDA shall allow the user to see the graphical depiction of the EIDS dials during playback

Applicable Courses from Iowa State University Curriculum

- Com S 309
- Com S 327
- Com S 228
- Com S 311
- SE 339
- SE 329

New Skills/Knowledge acquired that was not taught in courses

- C#
- WPF Applications
- MVVM Architecture
- ARINC 429, Network Data Object, MIL-STD-1553
- Collins Aerospace coding standards

Table of Contents

1. Introduction

1.1 ACKNOWLEDGEMENT

A special thanks to Zach Wright and William Johannes from Collins Aerospace for presenting such a challenging and rewarding project, as well as always making time for us. We'd also like to thank our Faculty Advisor, Lofti ben Othmane.

1.2 PROBLEM AND PROJECT STATEMENT

General problem Statement:

Collins Aerospace engineers have access to copious amounts of engine data from the digital instrument panels in their C-130 airplanes. This data details the status and performance figures from the entire center gage cluster during operation. Current tools to analyze these large data sets are underwhelming and unrewarding. There is a need for a robust analytical tool for aircraft engineers, technicians, and operators.

General solution approach:

The tool we are creating will be used by employees of Collins Aerospace to review flight data, show performance of engine parts over time, and diagnose issues within the aircraft. Engineers will be able to quickly review the data collected, make more informed design decisions, and utilize the data more effectively. It will have more tools, be faster, more informative, and easier to use than its predecessor. The final product will be an application capable of showing the data in a friendly user interface, for both technical and non-technical users.

1.3 OPERATIONAL ENVIRONMENT

The application will be run on a Windows environment (currently requiring Windows 10 and above). It will be a standalone application, capable of reading in .csv or .xlsx files, then parsing and displaying that data to the user. The application will not require internet access, and can write transformed data to .csv or .xlsx files. The application should be capable of taking in undefined field names, and let users create and define them. The application should also be able to parse partial data from incomplete data sets.

There are no hazardous working conditions to report for the project, although it will be constrained to Collins Aerospace Coding Standards.

1.4 REQUIREMENTS

The following requirements specified below were given to us from the clients at Collins Aerospace. These guidelines reflect several discussions and team meetings with the clients and are expected to be followed throughout the project lifecycle.

Functional

- The Trend Data Analyzer (TDA) shall be a Windows compatible executable file
- The TDA shall run without any internet connection
- The TDA shall provide capability to create user defined graphs, charts, and tables
- The TDA shall convert the ASCII data into ARINC 429, Network Data Object (NDO), and MIL-STD-1553
- The TDA shall parse the input file into selectable data fields
- The TDA shall be able to report anomalies and provide quick access to visualizations
- The TDA shall allow the user to export the parsed data to either .csv or .xls
- The TDA shall allow for playback of trend data once imported

Non-functional

- The parsing needs to run quickly, with a worst case of 10 minutes
- The Trend Data Analyzer (TDA) shall accept any .csv and .xls files as input
- The TDA shall recognize if the .csv input file is valid and provide a notification on failure
- The TDA shall provide data field names for predefined data per the EIDS_DCU_Collins_ICD.xlsx
- The TDA shall provide capability for user defined data field names
- Prior to exporting the parsed data, the user shall be allowed to filter what’s included in the data set
- The standard playback shall be sequencing the data points at a 1 Hz update rate
- The TDA shall allow the user to see the graphical depiction of the EIDS dials during playback

1.5 INTENDED USERS AND USES

The end users of the Engine Data Analytics Tool will be flight engineers, aircraft maintainers, operators, and quality assurance support specialists at Collins Aerospace. The application will provide the end users a tool to review engine data post-flight, spot any engine anomalies, warnings, or alerts, analyze engine performance, and visualize the data.

1.6 ASSUMPTIONS AND LIMITATIONS

Assumptions

- We are receiving clear and consistent data from a DCU (data conversion unit, a black box)
- Data will be formatted to fit ARINC-429, Network Data Object, or MIL-STD-1553
- Application does not require an internet connection to run
- Application will be run on Windows operating system

Limitations

- Written only in the C# programming language
- Handles data in only .csv and .xlsx files
- Can only analyze data from a C-130 airplane

1.7 EXPECTED END PRODUCT AND DELIVERABLES

Standalone parsing program for reading engine data (Deadline: 2/1/2020)

- The parsing program will read engine data in a standard format. This will require reading the input file, examining the hexadecimal values, and then parsing them in such a way that it is portable to many different CPU architectures in a performant fashion.

User interface for parsing, saving, and visualizing data (Due: 3/1/2020)

- The user interface (UI) will be in the form of a windows desktop application. The interface is responsible for the workflow of the engineer or quality assurance specialist. They will need to be able to conduct all of their reporting duties with no third party software. The UI will require scalability and customization for its graphs, charts, and other visual representations to give the end user more power on analyzing the engine data.

Visualization toolkit (Due: 4/1/2020)

- This toolkit will come in the form of an embedded workflow inside of the UI. This is responsible for general visualizations in the program. It will also support the ability to be invoked by other sections of the system. For example, a module that saves a snapshot of visualization data should be able to invoke the visualization toolkit.

2. Specifications and Analysis

2.1 PROPOSED DESIGN

From the problem statement, the goal of the project is to come up with a standalone software application that can quickly parse ARINC-429 data from C-130 engines and display the results for our clients at Collins Aerospace. After speaking with our clients, it was decided the solution would incorporate the following important requirements: A Windows executable file, accept any .csv as input, and interactively allow users to create graphs, charts, and tables for data analysis (The full listing of the requirements can be found in 1.4 Requirements).

Our proposed solution will be written in C# with a back-end with a purpose to efficiently parse the engine data, and a front-end with an emphasis on manipulating visuals and making the data clear to read for our end users. For parsing the data, the main goal is performance. The clients discussed with us that parsing the engine data could take anywhere from a couple minutes to well over 15 minutes of data parsing with their current application. The proposed solution will need to be considerably faster. For the front-end, it was decided that it too will be written in C#. The visuals will have the ability to be manipulated with an emphasis on showing abnormalities in the given data for the engine. It needs to be easy to use, informative, aesthetically pleasing, and efficient on display. One major requirement our solution must adhere Collins Aerospace coding standards which includes no compiler optimizations for our program.

The design will include a visual graphs front-end, and a data parsing back-end all as a standalone application. Since the application needs only to run on a Windows operating system, it will all be written in C#. The concept sketch below will show the general idea of how the application will function.

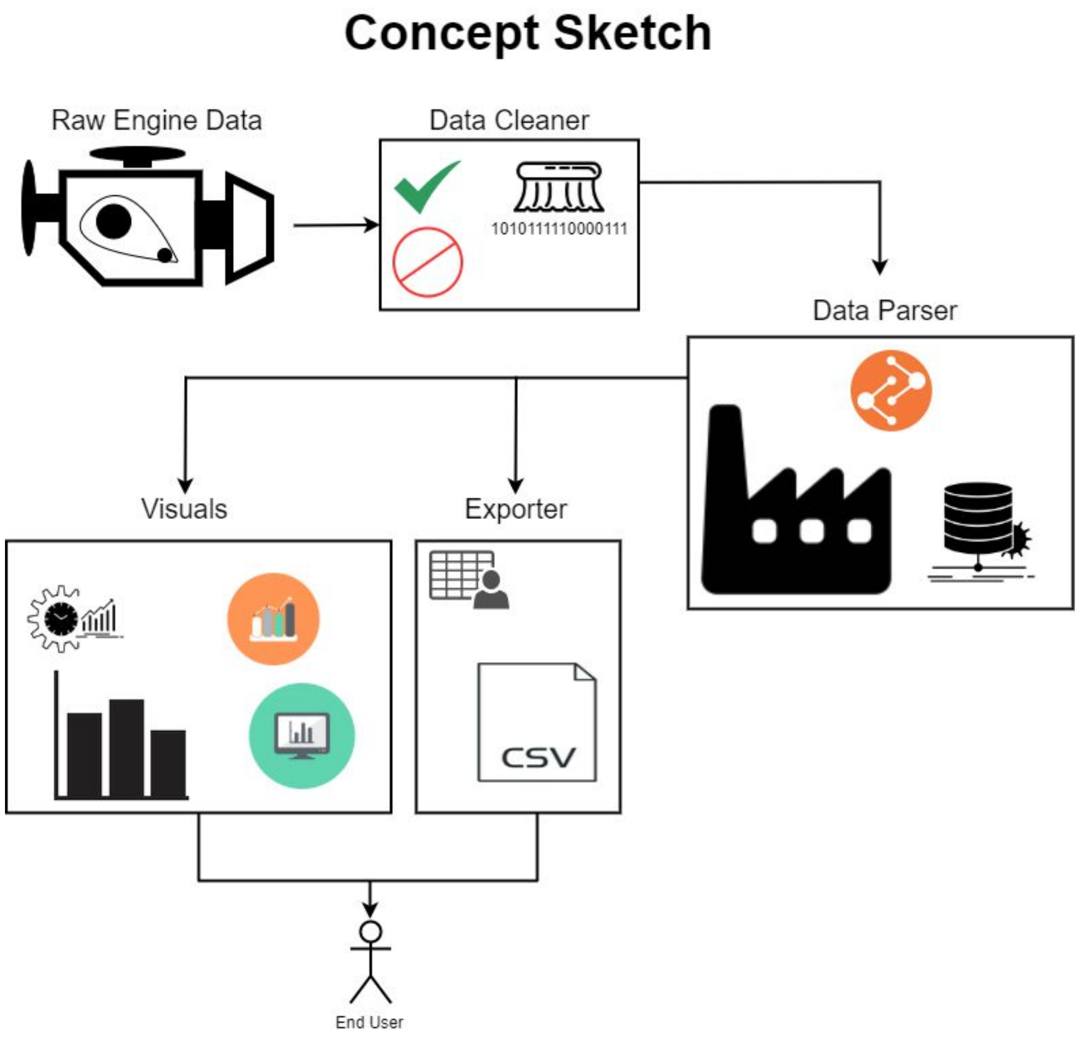


Figure 1

From *figure 1*, the engine data will be the input to the program. Then, a validity check and general cleaning of the data will occur. Once the data is valid and cleaned, it is moved to the parser algorithm which reads and formats the data for easy access. The front-end will take this newly formatted data and display the initial results in a series of charts, tables, and graphs that interactive displays the input. Lastly, the end users can manipulate the visuals to their needs, export the data if needed, and analyze the data for any abnormalities, warnings, or failures that weren't initially found or considered.

2.2 DESIGN ANALYSIS

Overall, the project so far has had successful tests, demos, and documentation. Instead of our group jumping into a potentially new language head on, we decided to start simple with a test application written in C++. This prototype could parse the .csv test file given to us from Collins. After researching the bit fields of the data, we concluded that the 32-bit data elements input to the parsing algorithm can be carried over to C# if necessary. From testing, the data given to us from Collins Aerospace had an average runtime of 12.344 seconds for parsing 100 files. This test was done without fully optimizing the algorithm, and it showed us that our data parser will be able to easily perform to what we expect in terms of runtime.

Having simple testing done allowed us to dive right into the problem, and its success paves an ideal path to actual implementation when the time comes. Along with the parser, the given .csv got converted to .xml to test possible data formatting. This test made it a lot easier to read and organize the data to give each bit more information. For demoing, a small GUI was created that was written in Javascript with HTML and CSS. This gave our clients a good idea of what we understood about the bit fields, and it provided a space for discussing how the application should perform during actual implementation.

For the following semester, some aspects to keep in mind are performance, scalability, and usability. Performance comes into play during the parsing of the data, and its strengths lie in multithreading, given input being 32-bits, and binary data representation. As for scalability, the client wants the program to have multiple UI capabilities and perhaps have the application work on a variety of airplane engines. In terms of our work so far, scaling the application is a weakness because we haven't tested any different kinds of data. Usability is in a good state from the demo. We found that representing the data is not difficult. The complexity comes in from implementing multiple visuals that can be changed from the end user. Also, usability could be a volatile aspect, because the client may want either minor or major changes to the front-end. Our client will want this part to encompass all of their user's needs.

2.3 DEVELOPMENT PROCESS

For the development process, we will be following along with an Agile methodology, with a close implementation of Scrum. The project size is relatively small so waterfall would have been a good methodology except for one key principle. Each meeting, the client brings something new to the table. With waterfall, it would be harder to adapt to new changes depending on the stage at which our team was working. Clearly, applying a scrum agile approach best fits the needs for our project. *Figure 2* highlights the key factors for implementing the methodology to the project.

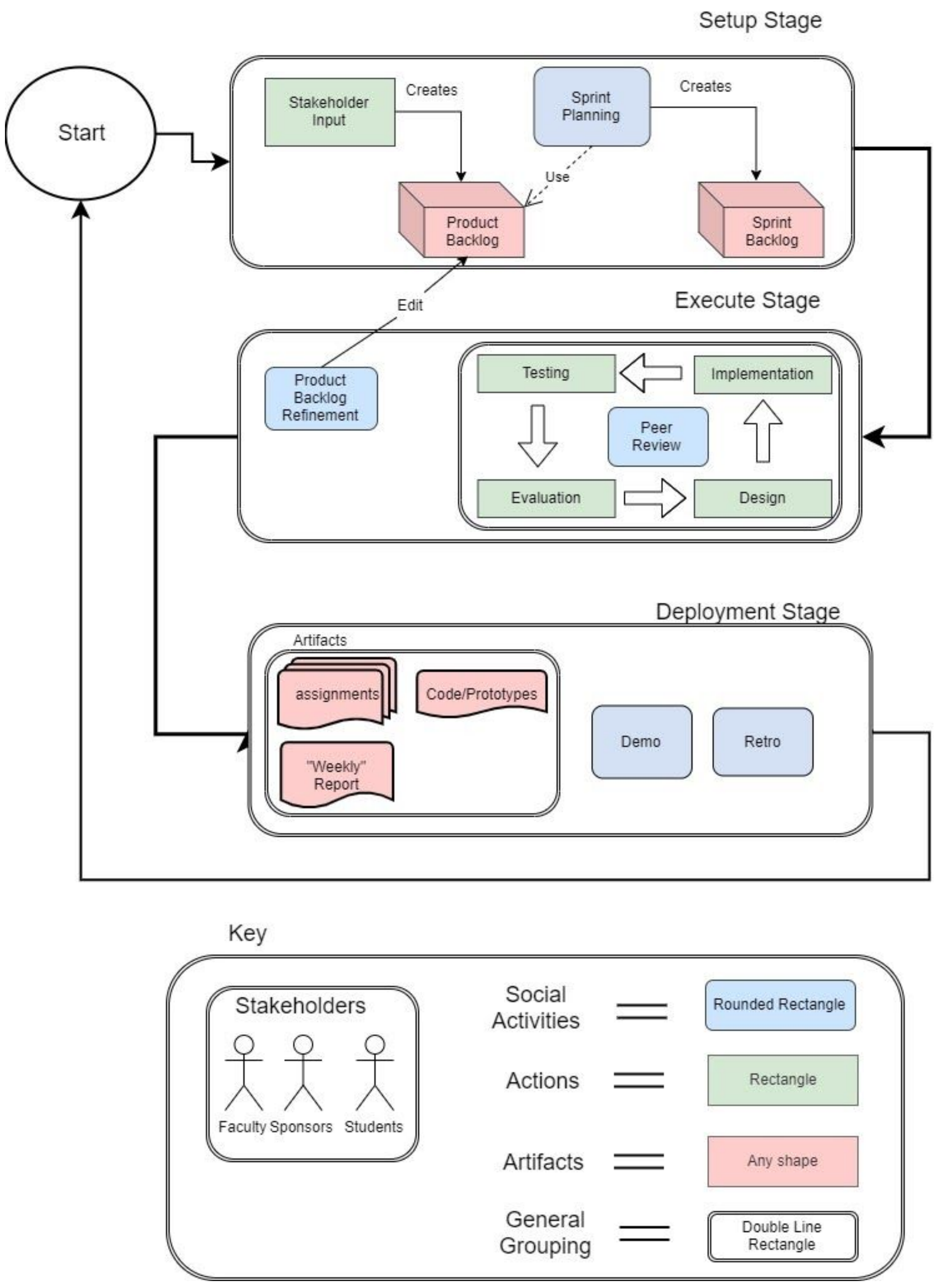


Figure 2

Looking at *figure 2*, the setup stage is responsible for the formation of product and sprint backlogs given from the client and team meetings. These are all recorded in our project's gitlab with the *Issues Board*. The execute stage is where our team takes the issues and attempts to resolve each one. This can be done by first designing the solution, implementing the idea, testing the resulting solution, and evaluating its performance and correctness. If all of these pass, the developer should create a merge request where it will be peer reviewed, and if it passes, will be merged into production.

The deployment stage is responsible for the documentation and presentation of the project. This is crucial for two reasons: client suggestions and course grade. For progress to be made, the client needs to be up to speed with the project's status so they can make accurate and timely suggestions, critiques, and modifications for the next sprint.

2.4 DESIGN PLAN

The design of our application can be described using a use case, functional decomposition, module, and system block diagram. Together, these diagrams show the design planning involved with user functionality, system communications, and functional capabilities.

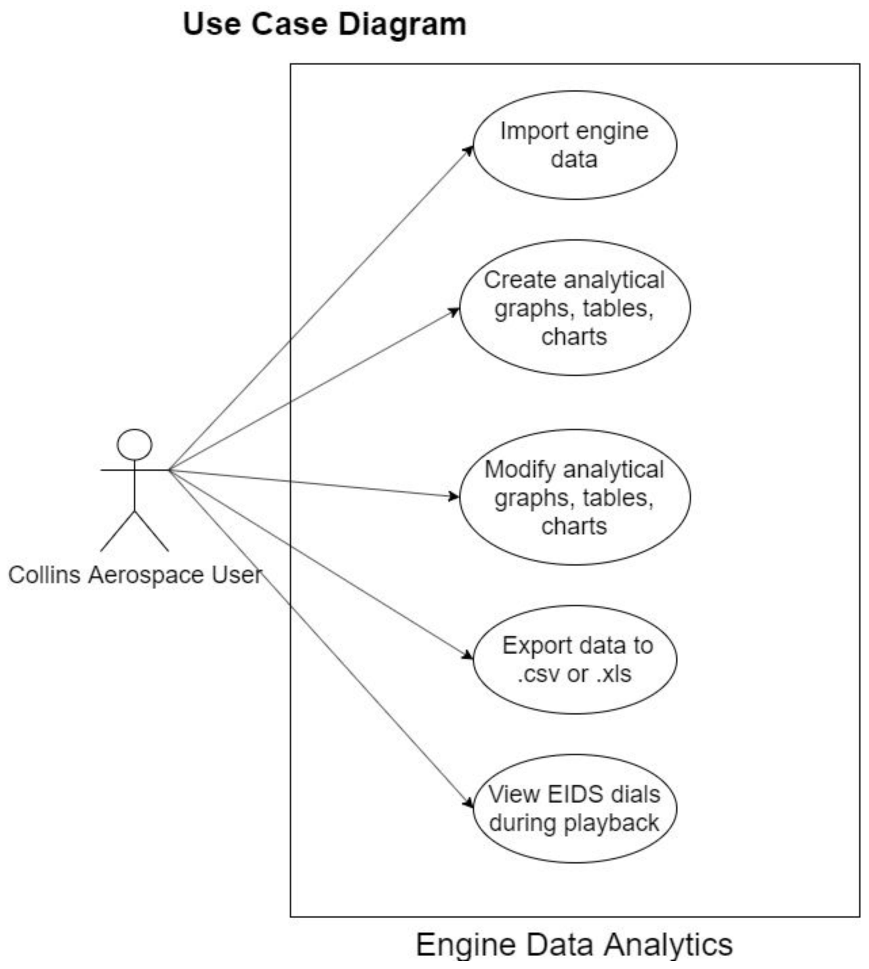

Use Case Diagram
Collins Aerospace User
Import engine data
Create analytical graphs, tables, charts
Modify analytical graphs, tables, charts
Export data to .csv or .xls
View EIDS dials during playback
Engine Data Analytics

Figure 3

The use case diagram describes what the end user will be able to interact with. There is only one type of user as the application is only built for the maintenance members at Collins Aerospace. The description of each box is pulled apart more with the functional decomposition (*Figure 4)*.

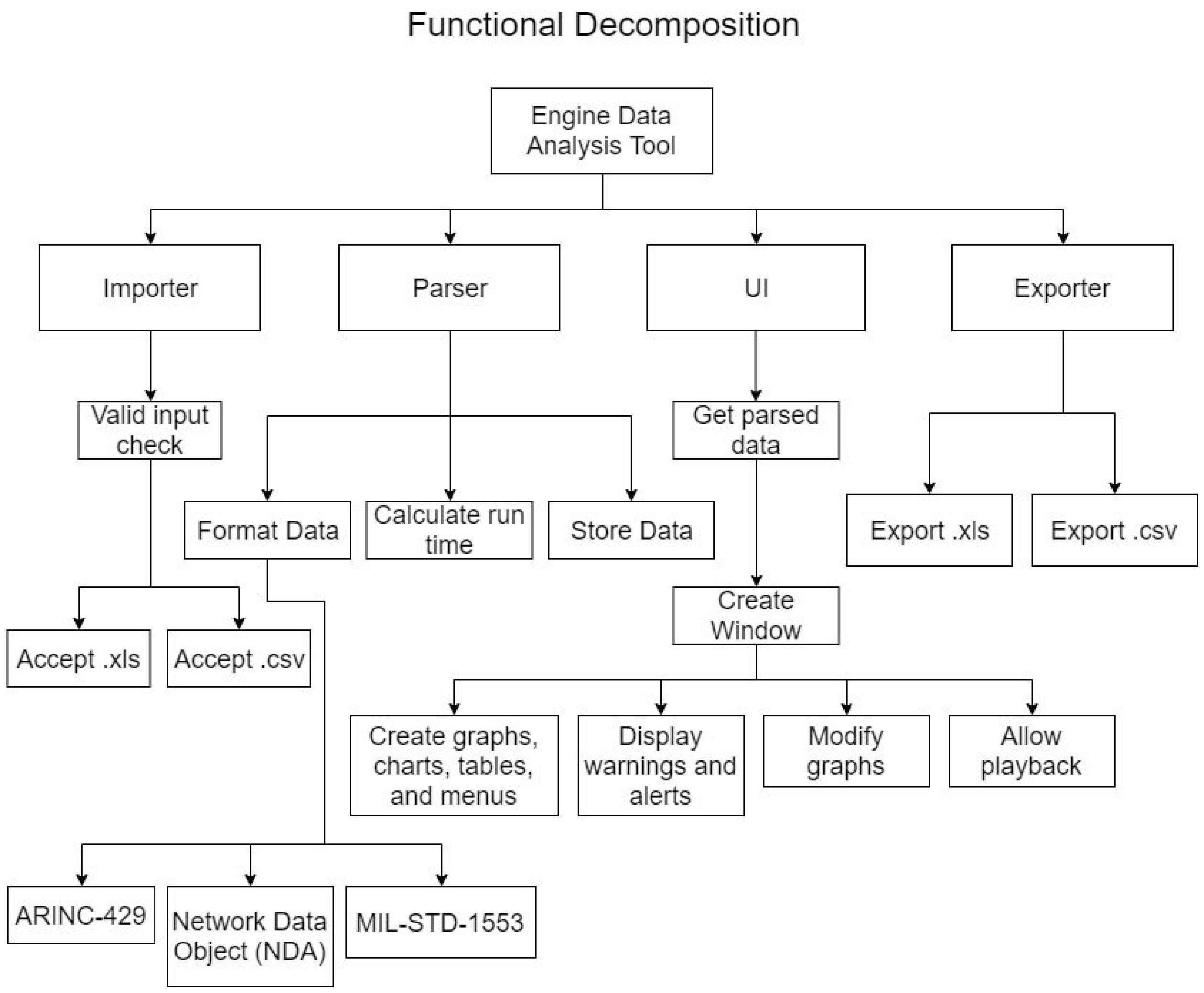


Figure 4

The functional decomposition shows the different capabilities the analysis tool will ultimately have. It does not show the specific function implementation, but rather, the high level functionality a group of functions hope to achieve. At the top level, the basic functionality includes the importer, parser, UI, and exporter. Then the diagram branches out to incorporate all of the functional and non-functional requirements. An example would be implementing the parser to format the data into ARINC-429.

System Block Diagram

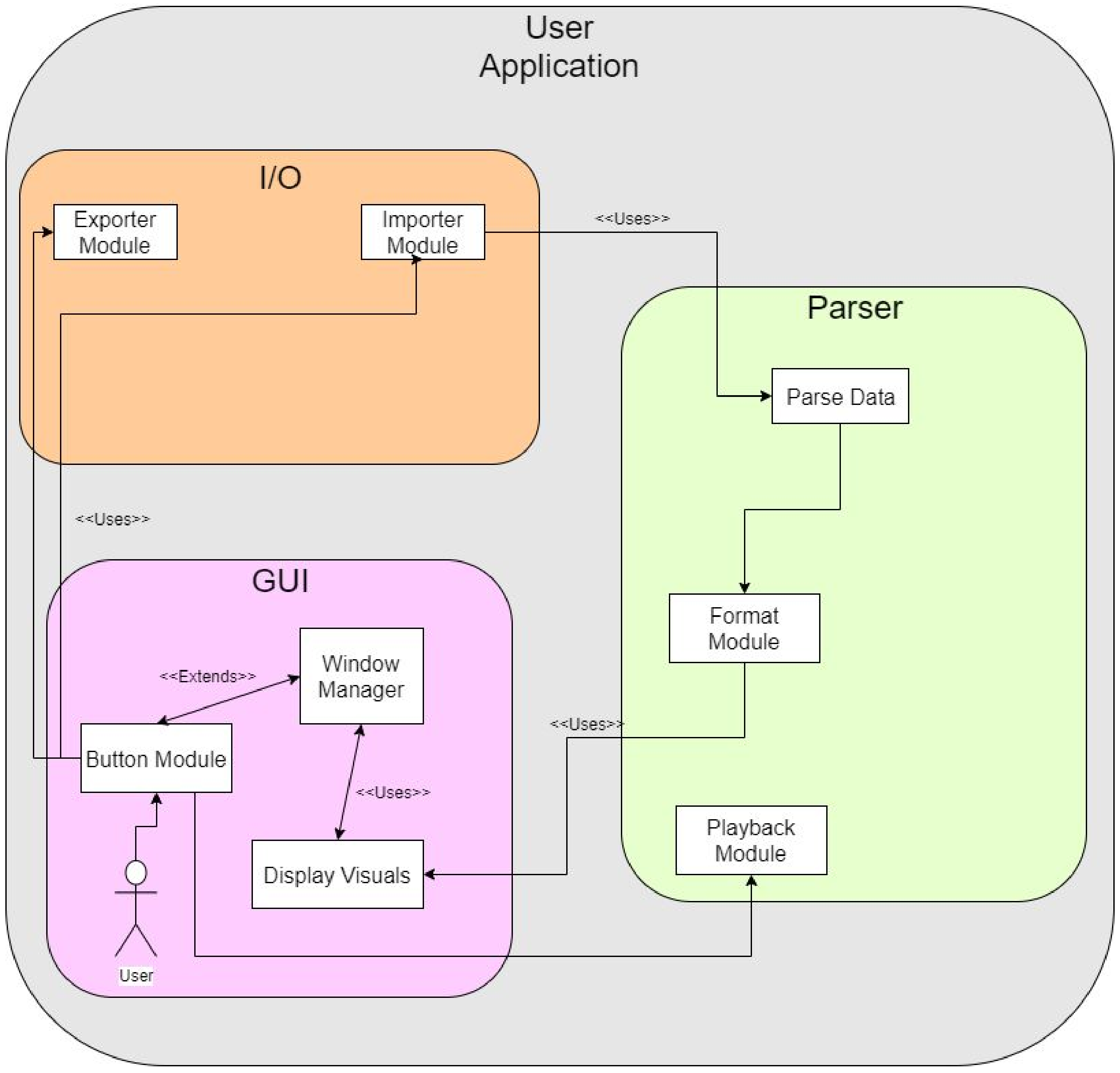


Figure 5

The System Block Diagram shows how the three main modules, I/O, Parser, and GUI, communicate with each other. The user interacts with the GUI using button presses, which are then sent to the respective module to be executed. Once complete, the information is sent back to the GUI to be updated. The system is relatively small, but each module contains complex logic for completing a task. The imported file to be parsed could contain more than 12 hours worth of data coming from the engine that is to be parsed. Along with that, the GUI will be responsible for manipulating the visuals to the user’s needs which will require numerous features and processing to render the given visual.

UI Example

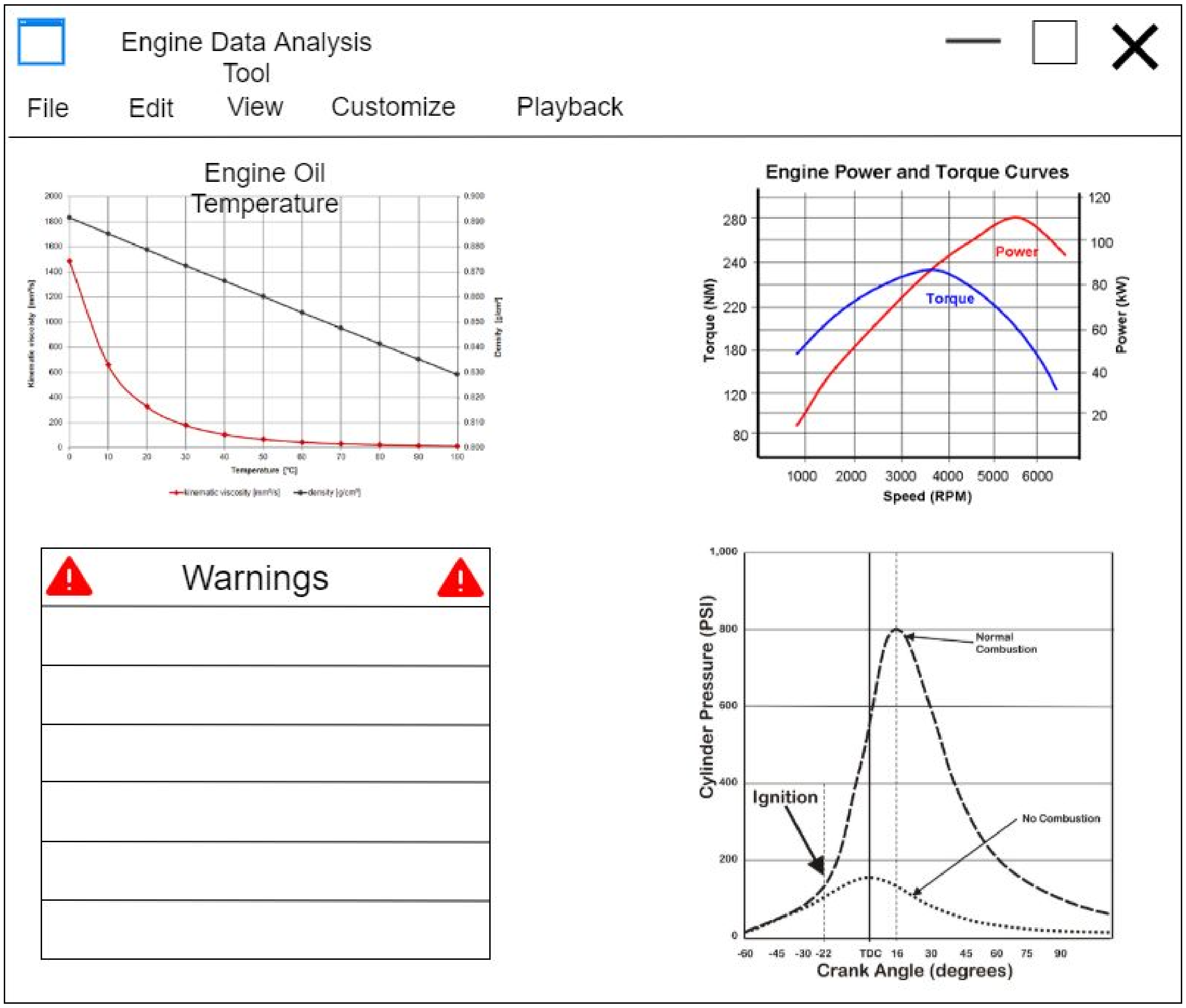


Figure 6

The User Interface (UI) will consist of various graphs, charts, and tables for the purpose of analyzing the formatted engine data from the parser. If the user clicks on the top panels (File, Edit, etc.), it will have a drop down consisting of features such as import data, export data, and modifying the graphs. From the requirements, there will be another feature that allows for playing back the data into one of Collins simulations. While the simulation is not necessarily required (Stretch requirement), it will provide the end user with another unique tool for visualizing the data that was imported from the C-130 engine. Accompanying the UI is a table that displays any critical failures or warnings coming from the data. This part will be especially important to the end user as the ultimate purpose of the program is for analyzing, detecting, and diagnosing engine problems.

3. Statement of Work

3.1 PREVIOUS WORK AND LITERATURE

Collins Aerospace currently has a tool to parse engine data files but they have found most of their users have opted not to use this because of performance issues. They state that parsing files can take hours with the current method they have. With our application, we will be able to parse these files in seconds.

3.2 TECHNOLOGY CONSIDERATIONS

When considering technologies, we had some requirements that influenced our decisions. We were assigned to create an application that can run from a .exe file on a windows machine. This along with our performance requirements led us to decide that creating a C# application. We wanted to utilize the and performance benefits that C# brings when compared to other languages such as Java or Python.

3.3 TASK DECOMPOSITION

Our project can be broken into three main categories of tasks: paring engine data, analyzing parsed engine data, and visualizing the analyzed engine data for the user. Each of our requirements pertains to one of these categories. These tasks will be implemented individually, but will interact as one. After each task is implemented, we will have one cohesive application as our final product.

3.4 POSSIBLE RISKS AND RISK MANAGEMENT

A couple concerns with our project are the limited amount of data we are able to receive from our client as well as any struggles that may occur with interfacing between the front and back ends of our application. As far as costs, materials, and equipment we are able to do all our development on our personal computers or computers provided by Iowa State University, leaving us with no risk in these departments.

3.5 PROJECT PROPOSED MILESTONES AND EVALUATION CRITERIA

Some of the milestones for our project include:

- Creating prototype data parser

- Creating prototype user interface
- Creating base level application (first release)
- Implementing features detailed in client requirements
- Implementing client defined stretch goals (time permitting)

Confirming these milestones will be fairly straightforward as we will release our application to our client at each milestone and allow them to verify that it meets their standards. Through agile development practices we will be able to take any feedback from our client and implement it in future releases.

3.6 PROJECT TRACKING PROCEDURES

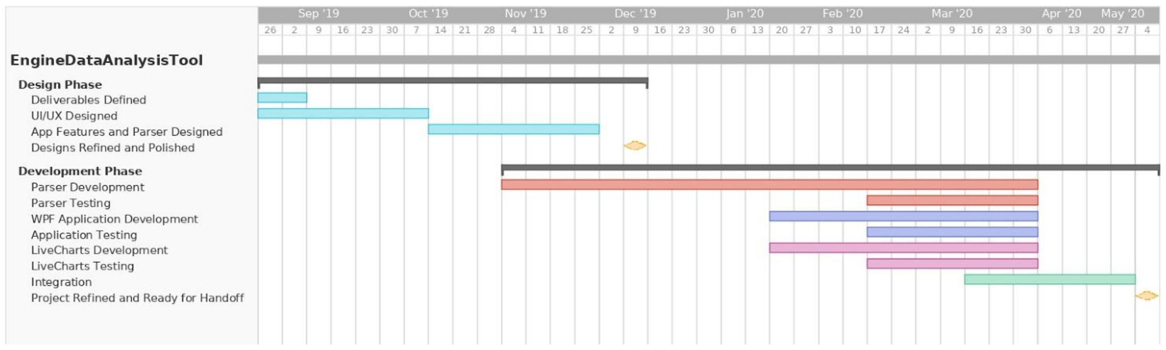
We will be tracking our project progress using the git issue tracking system.

3.7 EXPECTED RESULTS AND VALIDATION

At the end of our time working on this project, it is expected that we will have an application that takes in a .csv file from a C-130 airplane and allows engineers to analyze it. To confirm these solutions work at a high level, we will ensure our program is able to analyze any and every data file given to us from Collins Aerospace to their standards. After completing the base of our application, we will implement stretch goals to be determined in conjunction with the development team and Collins Aerospace engineers with consideration of our current program state and remaining time in the semester.

4. Project Timeline, Estimated Resources, and Challenges

4.1 PROJECT TIMELINE



Our Design phase will take up the majority of the first half of the project timeline while Development will take up the rest. We have divided our Designing and Development tasks into Parser, Application, and Charts design time and will focus in each sector to maintain a familiarity with the process throughout. Our issue tracker will be with GitLab and will keep track of the week-to-week operations and tasks whereas the Gantt chart shows how our timeline will proceed for each phase.

4.2 FEASIBILITY ASSESSMENT

The parsing portion of the project has proven to be highly feasible. Prototype parsing implementations have been made with satisfactory results. These prototypes have demonstrated the capability of reading in ARINC-429 data and put it into native C++ data structures.

Drawing static graphs of engine data is likely to have a great degree of feasibility. This code will be written in C# using Microsoft standard development kits. This is not suspected to take an enormous sum of time.

Drawing customized graphs will be moderately feasibly. This will require the most bandwidth of communication with our clients as well as the highest technical challenge.

4.3 PERSONNEL EFFORT REQUIREMENTS

Task	Hours	Description	Explanation
Generic Parsing	12	Write an ARINC-429 parser sdk that can be used for any ARINC-429 data	There exists working prototypes for this already. The extra time will be for polishing it and testing this as it will be the core functionality.
Customized ARINC Fields	8	Extend generic parser to also read in custom fields from Collins Aerospace	This will be easy once the generic parsing is done. All that needs to be done is to transform a subset of the fields after they are already partially parsed.
Initialize GUI	4	Build out starting GUI that works on Windows Desktops	This will be following a getting started page for a Microsoft GUI sdk
Base Workflow	8	Add navigation to the GUI for various features	This requires some forethought of what the client needs from the project.
Base Graphing	20	Put in feature for most important graphs to our client	This has a lot of potential to go wrong. It will also require our team members to learn new libraries.
Base Graphing Extensions	24	Fill in any base graphs that the client asks for	This could take a bit longer as the client could ask for many graphs.
Design Custom Graphs	10	Come up with a mechanism to allow users to create their own graphs	This could potentially be difficult. The user will need to have an intuitive interface to describe a graph, then this will need to be serialized and later parsed. Finally, a graph needs to be drawn from the configuration.
First Custom Graphs	20	Create a prototype custom graph from the design	The implementation will likely take longer than the design of custom graphs because flaws in the design may emerge and a lot of custom code will need to be written.

Extended Custom Graphs	80	Create any more graphs that the client asks for	This will take the rest of the year. The client may ask for more graphs as long as the project persists.

4.4 OTHER RESOURCE REQUIREMENTS

This is not applicable to the project. No physical materials are required. The client has assured us that they already have windows computers that will be able to run the code. The machine that collects the engine data is already finished and working. No licensed software or persistent servers are required as this is a desktop application.

4.5 FINANCIAL REQUIREMENTS

This section is not relevant to our Senior Design project.

5. Testing and Implementation

Unit Testing Cases

We will unit test our data parser for accepting ranges of data sets. Differing in volume, quality, and other data input modifiers, that can effectively test our parser. We will also want to check our data transformer and GUI to make sure it holds up under different data sets being loaded in.

Integrity Testing Cases

Our parser should perform no differently for large or small data sets, except in time completion. We will want to check our data and make sure that data is not corrupted when loaded from larger files, as well as make sure our small data sets do not fill in blank values or expect non-null fields.

User Testing Cases

Our GUI should be easy to use and intuitive to any new user. We can test this by introducing sample data and asking sample users to navigate to a data field, load files, or perform some other operation that actual users will want to do.

Continuous Integration Test Cases

We will reuse test cases from the above categories to ensure our enhancements against any regression.

Test Cases and Results Format

Our repository will contain our test cases alongside our code for easy portability to any developers station. The results will also be backed up with noticeable results, abnormal results, and goals for results. Our communication channel, Slack, will be used to notify of any application breaking updates or test cases. We hope to have our CI/CD up and running to automatically run applicable tests to new merges to master.

5.1 INTERFACE SPECIFICATIONS

We will be testing our file reader, as well as our file writer alongside different versions of Windows applications. This will ensure a compatibility range that will be suited to our clients needs.

5.2 HARDWARE AND SOFTWARE

We only require software testing, making our project much more capable of being in TDD for portions of the development process. With knowing what expected output should be, we can create tests before development begins and navigate the development process always with the end goal in mind.

5.3 FUNCTIONAL TESTING

We will have Unit, Integrity, and Acceptance testing covered. All test cases will be developed alongside the project, with a focus on test cases that relate most to what is the next deliverable.

5.4 NON-FUNCTIONAL TESTING

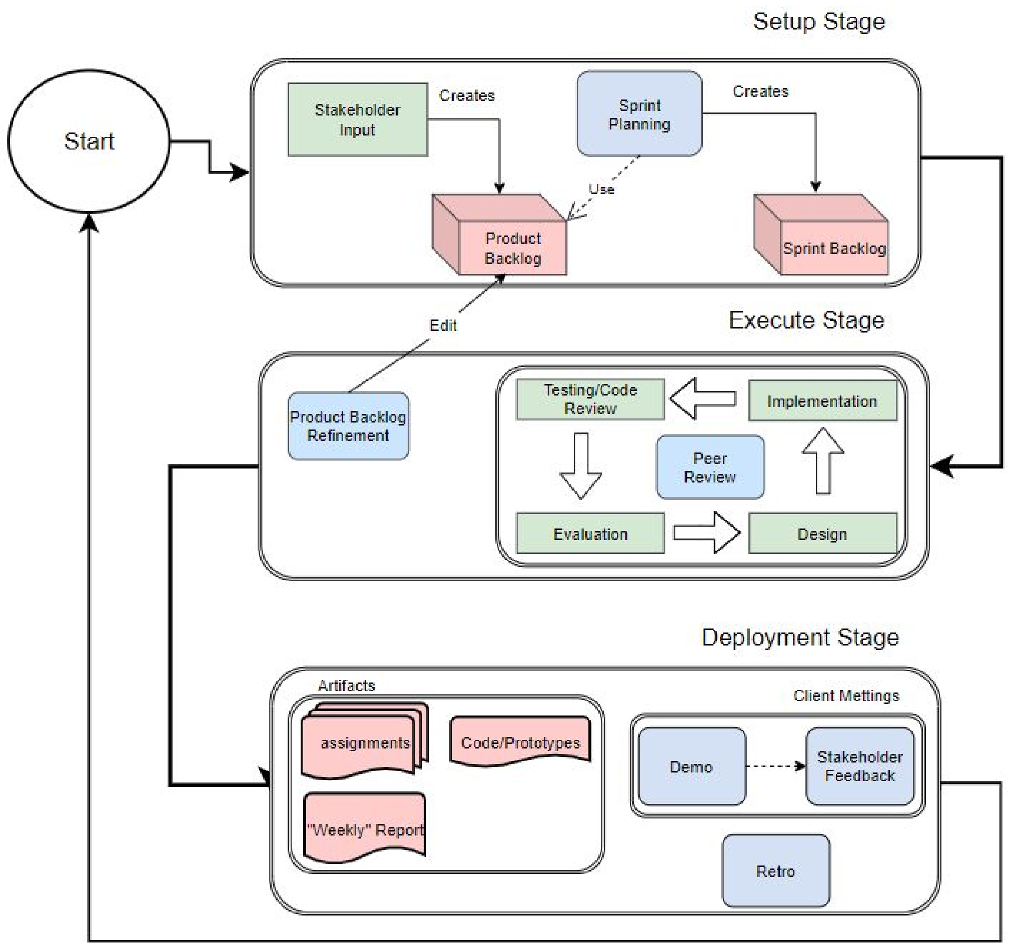
With our project development being broken into 3 distinct parts, data parsing, data transformation, and data visualization, we can test for performance, security, usability, and compatibility at each step in the development process.

5.5 PROCESS

In our development, we successfully followed the Agile methodology with a similar interpretation to Scrum. This allowed us to get quick feedback from our client each time we implemented a new feature.

- Parsing was tested using C# unit tests
- Front end was periodically demoed to the clients

– Flow diagram of the process:



5.6 RESULTS

Success

- Successfully implemented unit test suite for parsing and exporting
- Parser runs underneath their goal time, and eliminates unnecessary overhead
- We learned that C# does not have built in support for buffered reading of text files integrated with LINQ. Text Files must be read eagerly.

Challenges

Integration - We didn't integrate everything until the end and we found out that we created different kinds of applications in our IDE Visual Studio. It took a week to get everything fully integrated with each other.

Backend issues - For parsing the biggest issue was a lack of specification for parsing the data. No one was using the data because of how much effort it took to extract it, thus it had fallen by the wayside. A few weeks into our Spring Semester, our Collins representative found a schema file which made us completely rewrite the backend data parser. In the end it was a lot better.

Exporting Issues - One of the stretch goals was to have the option of exporting to .xls format. The data varies so much though, and there are so many ways to represent the data in charts or tables. We could not find a good way to represent each data type without writing a lot of code for all 328 different data types. Not to mention a way to include any possible future data types. So we kept it exporting as .csv and .xml since they are robust.

6. Closing Material

6.1 CONCLUSION

Work done so far includes obtaining the requirements form the client, implementing a testing prototype, and documenting the project plan, design, and testing for the following semester. After getting the requirements, our team was able to start testing with a prototype parsing application written in C++ to get a better idea of the performance of the data parser. The client greatly appreciated this, and started giving us additional ideas for us to implement when the time came.

Our development process will start with completing the data parser, handling ambiguous data, and allowing the creation of undefined data fields. We will then move on to the User interface for parsing, saving, and visualizing data. Finally it will culminate into the software application toolkit, that can take an input, display the data in a visually-pleasing, graphical manner to the user, and export the changed data/variables.

The most important factor in our development process will be holding each other responsible for their tickets, and ensuring integration of each new part added to the application. We want to produce the best product possible for not only our experience, but also the benefit of our clients.

6.2 REFERENCES

ARINC Protocol Tutorial. GE Intelligent Platforms, 2010.

Docs.microsoft.com. (2019). *Getting Started (WPF)*. [online] Available at: https://docs.microsoft.com/en-us/dotnet/framework/wpf/getting-started/ [Accessed 9 Dec. 2019].

Tableau Software. (2019). *Tableau Desktop*. [online] Available at: https://www.tableau.com/products/desktop [Accessed 9 Dec. 2019].

6.3 APPENDICES

Two manuals were used to learn C# and C++ cited below.

Liberty, J. (2002). *Learning C#*. Beijing: O'Reilly.

Meyers, S. (n.d.). *Effective Modern C++*.

Appendix I - Operation Manual

Welcome to the Engine Data Analysis Tool Application. Hopefully you can find this tool as fulfilling to use as it was to create.

The application has 3 pages, the Home-page, Export-page, and Analysis-page. The Home-page is the initial startup screen that the user will see. The Export-page is used to handle exporting files with specific conditions. The Analysis-page shows an in-depth look to what the data holds.

To Import for Analysis (From Home-page):

- First enter a path into the **File Search** textbox that starts with “C:\”. This can be a path anywhere on your computer (we initially set it to the C drive, so not entering anything will go with the default “C:\”).
- Then click the **Import** button to open up a File Explorer and select the CSV file you would like to import. Once imported the application will redirect to the Analysis-page.

To Analyze a File (From Analysis-page, follow steps above first):

- The simple layout allows for an intuitive grasp of the data controllers and resulting charts.

    - **Select Data Fields** will give you a drop down of fields which you can select from, to see that fields data.
    - **Engine 1..4** checkboxes give the user the option to see a specific engine's performance (if not applicable then checkboxes will not affect the charts) and select or deselect to see differences across engines.

To Export a File:

- The first step is to click the **Export to Parsed CSV File**. This will bring you to the Export-page.
- The Export-page has many options for the user to export the file with the parts that they require in the format that they want.
    - **Export As** checkboxes can both be selected to export as either CSV or XLS filetype.
    - **Source** is the file that is being parsed once Browse is clicked (this textbox is not a field you can type a path into and must be navigated to via file explorer).
    - **Destination** is the folder that the parsed file will be written to.
    - **Data Fields Selected** are the fields that the user can select to indicate parse only these types of fields.
        - **Select All** will select all fields.
        - **Clear All** will clear all fields.
    - **Cancel** will take you back to Home-page.
    - **Export** will export the file decided upon in **Source** with only the **Selected Fields** parsed. You will get a notification if it succeeds or fails and can find the resultant file in the **Destination** folder you specified.