Finding Hay in the Needle Stack: Doing the Right Thing at the Right Time

Final Report

Team Number: sddec19-09
Client: Collins Aerospace
Adviser: Thomas Daniels
Team Members/Roles
Austin Konsor: Software Architect/Chief Engineer
Mireille Mwiza Iradukunda: Test Engineer/Report Manager
Gooi Yin Hong: Lead Designer/Web Master
Yealim Sung: Project Leader/Meeting Scribe
Team Email: sddec19-09@iastate.edu
Team Website: sddec19-09.sd.ece.iastate.edu

Revised: 12/10/2019

Table of Contents

List of Figures

List of Definitions

SVN - Apache Subversion

STARWARS - Database that contains test result data for the script files in the SVN.

Test Case - Script file in the SVN repository

Test Event - Group of Test Runs ran on the Test Cases

Test Run - Single test ran on a Test Case

1.0 Executive Summary

Collins Aerospace has about 8000 common script files (test cases) that are spread across many directories within a SVN repository. These are python files used by Collins engineers. However, of those 8000 files, there may be many versions of the same test case. These will have the same file name with the same functions, yet some may work better than others. As of right now, the Collins engineers have no way of knowing which file works the best unless they were to manually search through all of the programs themselves. So the engineers must fix each functions everytime they find a mistake, when that mistake may have been fixed in another version.

In a database called STARWARS, Collins has a collection of test result data regarding each of the script files. These include a large number of test events that are run on the said script files which gives test results such as the time the test was run, the result (pass, fail crash), number of verifications, number of those verifications passed or failed, file path, and many more.

Our proposed solution is an application that will recommend and visualize the best file/function using the given SVN repository and STARWARS test result data. The user will have the ability to search a desired function to receive the results necessary to move forward on their projects efficiently.

2.0 Requirements Specifications

2.1 Functional Requirements

Analyze input files - The application is able to crawl through the directory of script files and find where each instance of a function occurs. It is able to run an edit distance function on two different files or functions to see how many changes are required to get from one of the files to the other. Finally it is able to visually show a side by side comparison of two different files or file/function combination in a new window to see what differences there actually are.

Visualize end result - The application is able to visualize multiple aspects of results for Collins Aerospace engineers to analyze. One has already been mentioned, which is a side by side comparison of files that shows the additions, deletions or changes between two files, such as a GIT commit would show.

Quick, accurate, and reliable output - The result should help the engineers to quickly and easily determine what to do and provide deeper understanding.

Offer suggestions - Given a desired function the application will provide enough information for the Collins engineers to analyze and make a decision in which file to move forward with.

2.2 Non-Functional Requirements

Maintainability - The Collins Engineers should be able to maintain it for their own future use possibly without the team's help.

Usability - The Collins Engineers should be able to modify it and implement it to fit their environment.

Compatibility - It should be compatible with the Collins system.

Security - The application should strictly be used by Collins Aerospace employees.

3.0 System Design & Implementation

3.1 Design Plan

Our design is implemented into two seperate pieces which are driven by two sets of resources given to us by Collins Aerospace. This is designed to be a desktop application built with Python 3 that will strictly run on Collins system, which is a Windows environment. The first part of the design is a file crawler which crawls through the SVN of script files and gathers various important information to be used. The file crawler runs right when the application is opened. It grabs specific information such as each unique function+parameters instance and every file path that it came from. We are able to use this information for our edit distance function, file comparison, and the second part of the design- the visual analysis.

The visual analysis comes in multiple forms available for Collins to use. The data involved in the visuals comes from the test result database called STARWARS. It holds the information of many test events that are run on the script files from the SVN directory. The interface of this system is built with Tkinter. A user will have the ability to search for a desired function they intend to work on, which will already be gathered from the directory crawler. The first part of the visual analysis comes in table form that shows historical data. Given the function, it will list each instance (file) it is located in along with the columns of average score (percentage), the date of the last test run on it, the branch the test results come from, total number of test runs on the file, and finally the edit distance value. The user will have the ability to sort each column, the ability to compare two files from this page, and also decide which file the edit distance is being calculated from. The second type of visualization comes in the form of a graph, built with the Matplotlib library. A user is able to select their desired function once again which brings them to a graph customization screen. In it they can select different ways to set up the graph, such as selecting the branch of test result data, which files to graph, and even what values should be in the x and y-axis. The graph will then show the selected data, which normally is each test run on the selected functions, separated by color for which script file they are in. This will help the Collins engineers decipher how well a function compares from one instance to another to give them the ability to make a further decision on their own development.

3.2 Design Objectives, System Constraints, Design Tradeoffs

Design Objectives:

- Create a simple to use application for Collins Aerospace engineers to use.
- Perform our directory crawler when the application starts running.
- Apply the visualization and data aggregation functionalities when directed by a user.

System Constraints:

- Must work in a Collins Aerospace system (Windows).
- Need to write code in Python 3.

Design Tradeoffs:

- We wanted to add as much functionality as possible for the engineers to use. In return the interface design suffers a little bit.
- That can also be said for using a simple library such as Tkinter to design our interface. It is simple to use, develop and maintain, however it doesn't have an extremely pretty look.

3.3 Architectural Diagram, Design Block Diagram -- Modules & Interfaces

3.3.1 Architectural Diagram

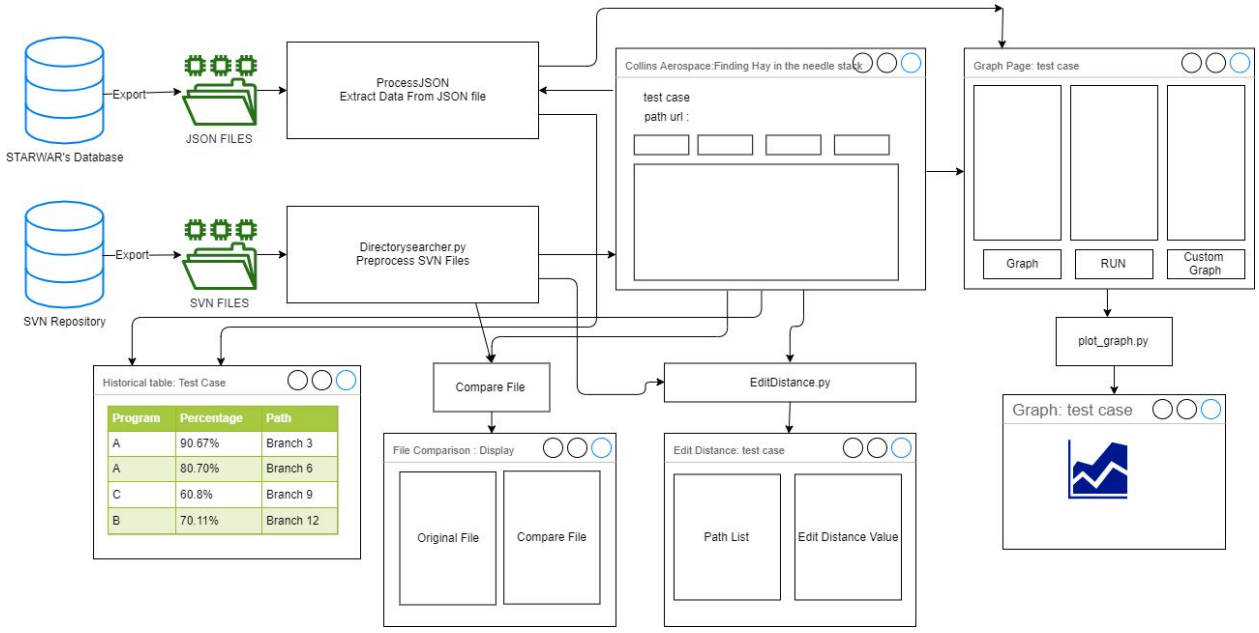

Figure 1: Architectural Diagram

3.3.2 Design Block Diagram

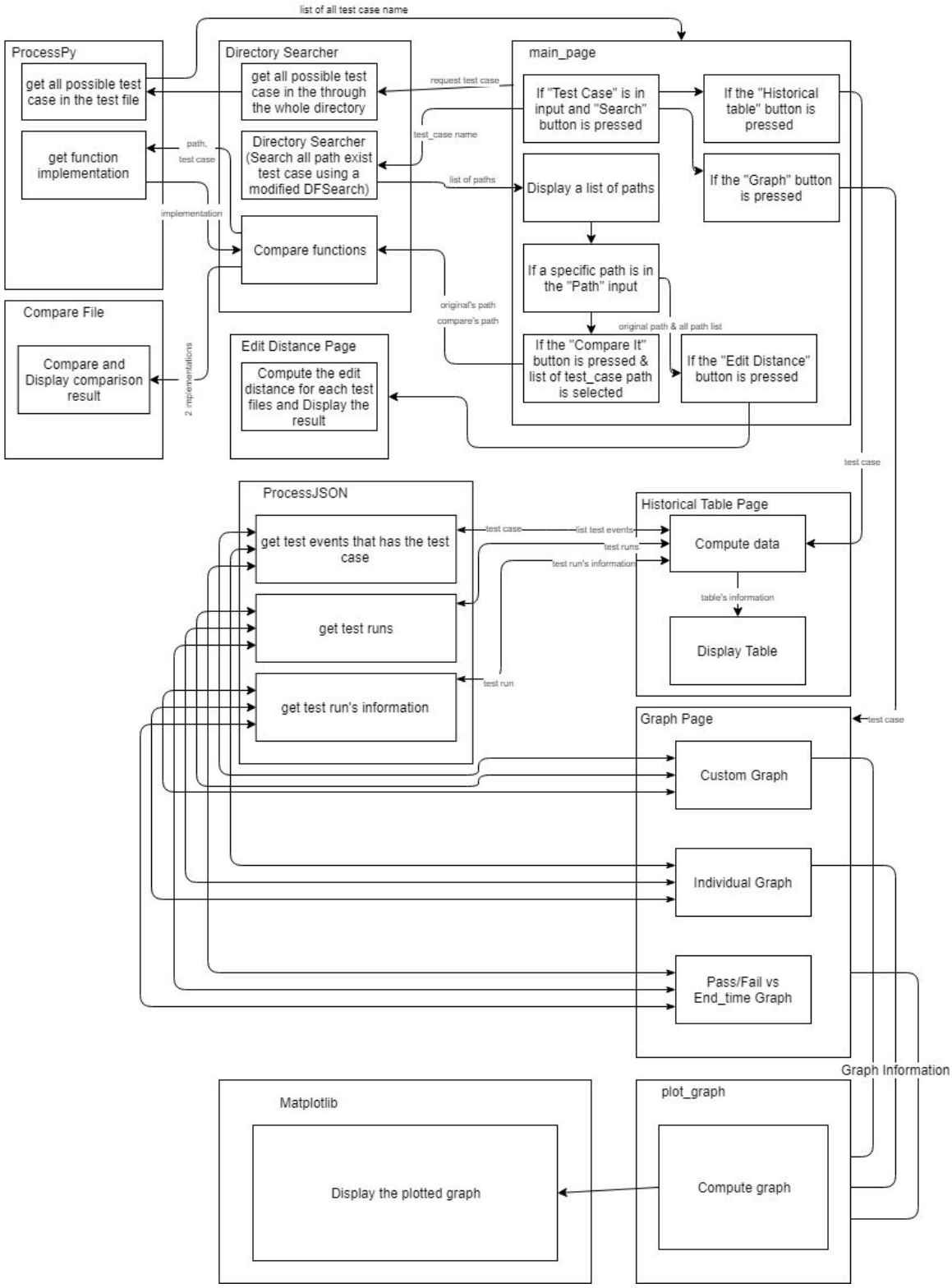

Figure 2 : Design Block Diagram

Interface

Main Page Interface

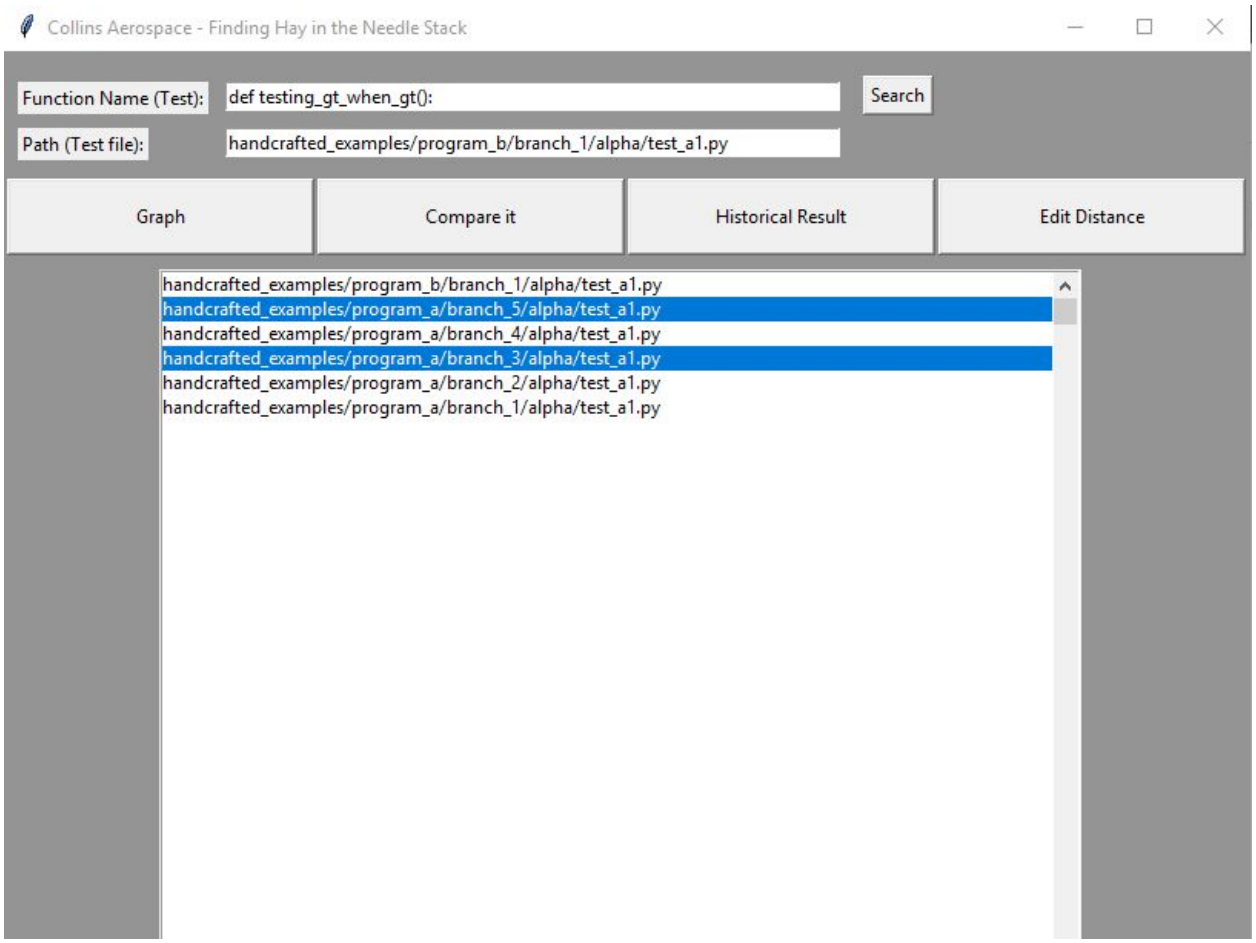

Figure 3: Main page Interface

Graph Page Interface

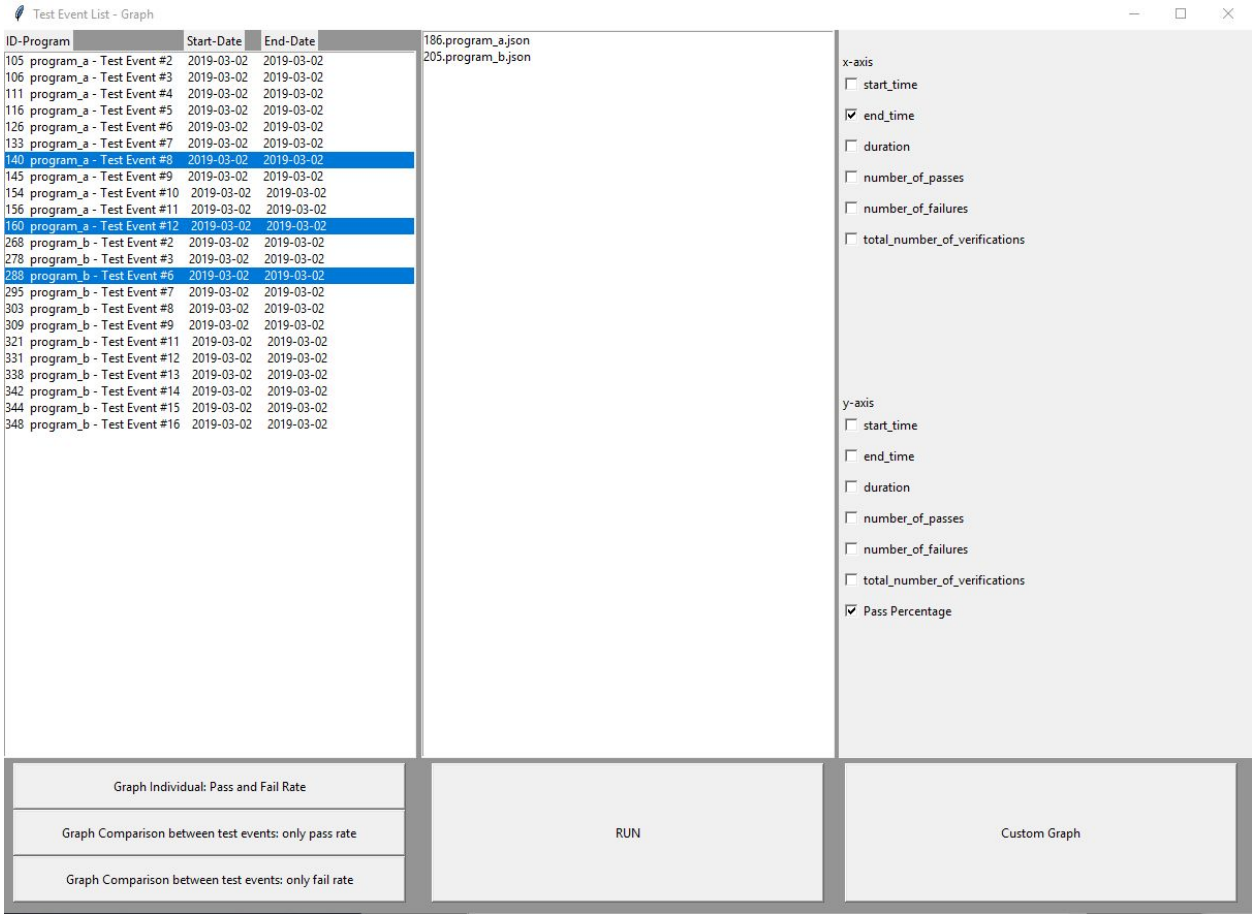

Figure 4: Graph page Interface

Graph

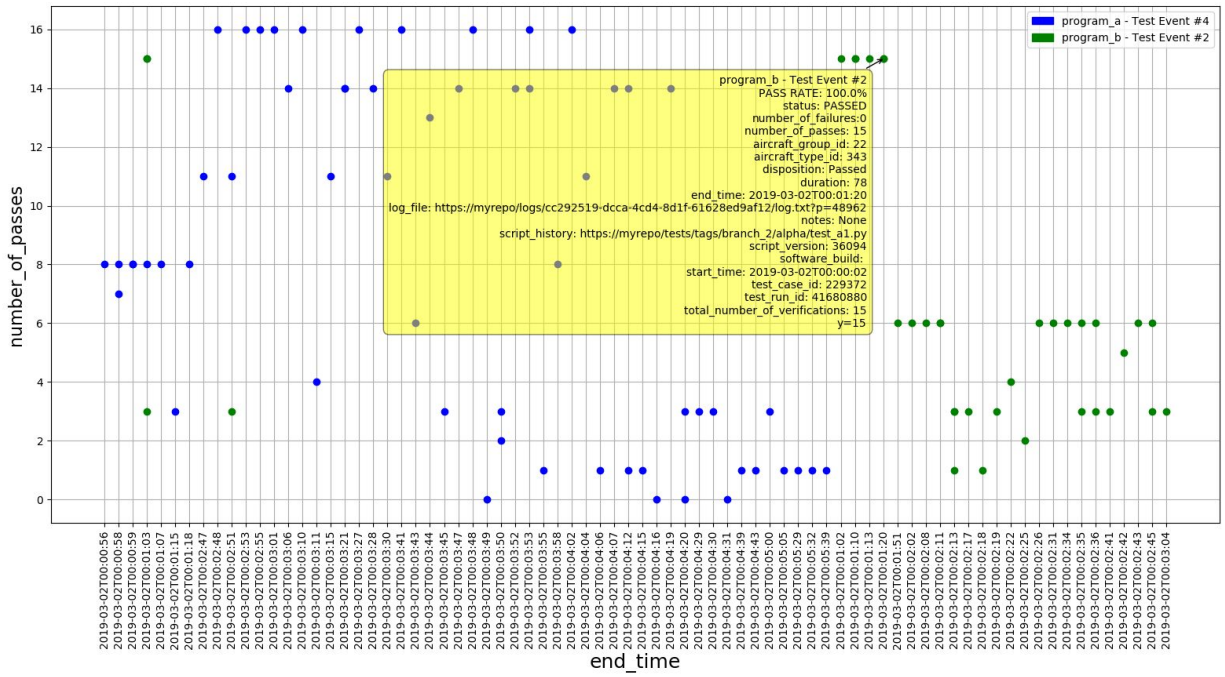

Figure 5: Graph Interface

Historical Table Interface:

Historical table: testing_gt_badly

	script_history	Program	Last Run Date	Overall Perc.	Test runs No.	Edit Distance
1	https://myrepo/tests/tags/branch_4/alpha/test_a1.py	186.program_a.json	2019-03-02 00:04:22	93.88	84	0
2	https://myrepo/tests/tags/branch_5/alpha/test_a1.py	186.program_a.json	2019-03-02 00:01:22	92.02	94	0
3	https://myrepo/tests/trunk/alpha/test_a1.py	205.program_b.json	2019-03-02 00:02:45	89.56	222	0
4	https://myrepo/tests/tags/branch_3/alpha/test_a1.py	186.program_a.json	2019-03-02 00:04:08	88.79	107	0
5	https://myrepo/tests/tags/branch_2/alpha/test_a1.py	205.program_b.json	2019-03-02 00:01:21	88.57	242	0
6	https://myrepo/tests/tags/branch_1/alpha/test_a1.py	205.program_b.json	2019-03-02 00:03:12	88.41	253	0

Figure 6: Historical table page interface

Compare Implementation Interface:

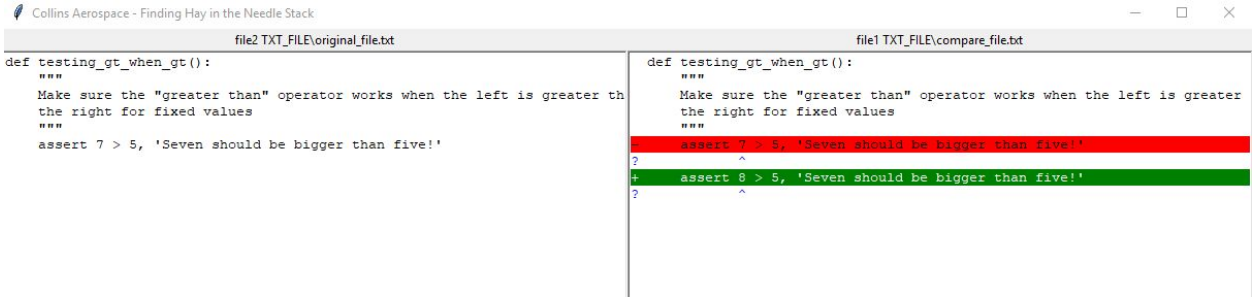

Figure 7: Compare Implementation Interface

Edit Distance Interface

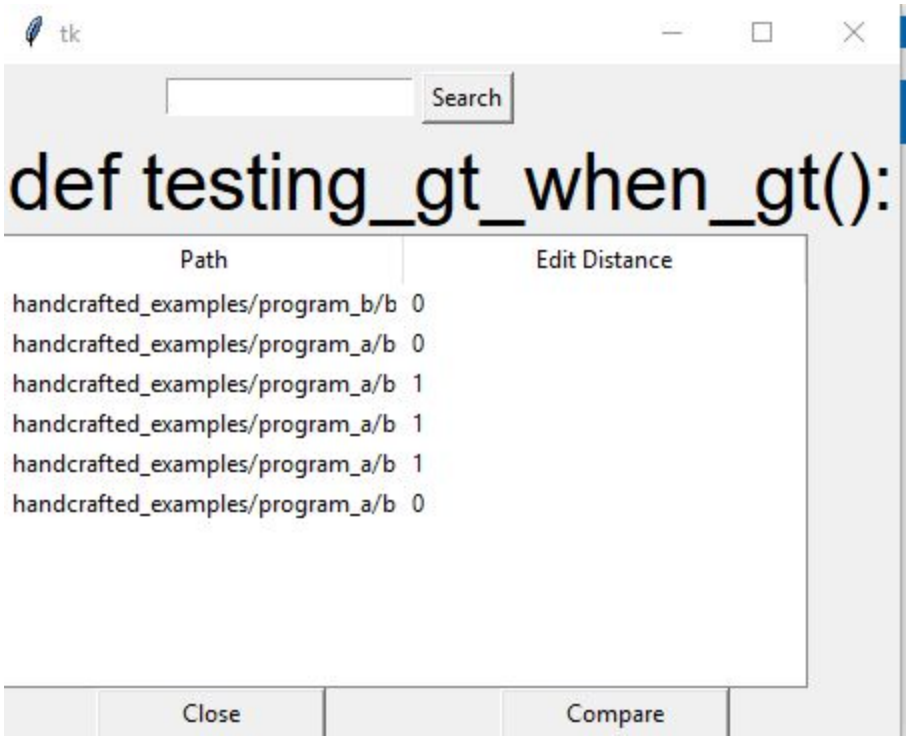

Figure 8: (Edit Distance Interface)

3.4 Description of Modules, and Interface

Modules:

Modules	Description
DirectorySearcher (figures 1 and 2)	The DirectorySearcher analyze the imported SVN directories by crawling through all python files using a slightly modified Depth First Search algorithm. The DirectorySearcher is used to find all the possible python files, and also to find all the possible function names. All those gathered function names can be found in a drop down list in the "Function Name (Test)" input that is in figure 3. It can also be used to find all the python test files path that has the specific function names.
ProcessJSON (figures 1 and 2)	ProcessJSON is a module for extracting data from the JSON file. Each and every functions in the ProcessJSON has their own responsibility of extracting different kinds of data such as: test_events, test_cases (function), test_runs' information, and etc. Most of these functions are heavily used to plot dynamic graphs, and make recommendation table. We are also planning to include edit distance value in the future.
Main_page (figures 1 and 2)	The main_page mainly consist of GUI implementation. Most of the GUI interfaces calls different parts of components such as: Historical Table Page, Graph Page, Edit Distance Page, Compare File Interface. The main_page has all function names information from the DirectorySearcher, and also require a function name to be able to call the other components to work.
Graph Page (figures 1 and 2)	The Graph Page mainly consist of GUI implementation. It has to be called from the main_page with a function name in order to work. This page gets all the information that the engineers want to plot the graph.
Matplotlib (figure 2)	After getting all information to plot the graph from the engineers, matplotlib will then process all the information and display the results.
ProcessPy (figure 2)	ProcessPy is a helper module for helping DirectorySearcher for finding all the function names, and also to find the code implementation for file comparison.
Compare file (figures 1 and 2)	The compare file module takes two different code implementations and compare the differences between them and display the results.
Edit Distance (figures 1 and 2)	The edit distance is also used to compute the differences of 2 code's implementation, but it will return as a value format.
Historical Table (figures 1 and 2)	The historical table takes in a function name and will then try to compute all information that ProcessJSON gather for the specific function name. It will then sort the information computed according to the pass rate percentage.

Table 1: Modules Description

Interfaces:

Interfaces	Descriptions
Main Page Interface (figure 3)	In the main page the "Function Name(Test)" input has the ability to drop down a list of function names for the engineers to choose. After selecting a function name, they can search all the python file that has the function name by clicking the search button, and it will display a list of paths in the list box. They can select multiple paths in the list box that can be used to compare files and etc. The main page also redirects to other pages such as: Graph Page, Compare page, Historical Result Page and Edit Distance Page.
Graph Page Interface (figure 4)	The Graph Page has two list box that the engineers could select, the test_event list and program_list, they could also select what type of custom graph they want simply by checking the checkboxes at the right for x-axis and y-axis.
Graph Interface (figure 5)	The Graph Interface is an interactive graph. By clicking on the points in the graph, it shows a note window with the information of the test runs, the note window are draggable for comparing different points in the graph. If the points are too cluttered, they can also zoom into the graph to have a closer look between the points.
Historical Table Interface (figure 6)	The historical table shows the best test case that has the highest pass rate from all program and test events. The historical table has 6 columns which are, script_history, Program, Last Run Date, Overall Percentage, Test runs No. and Edit Distance.
Compare Implementation Interface (figure 7)	The compare implementation interface displays how different between two codes implementation. If there is an additional code, it will be highlighted in green, and if there is any missing code, it will be highlighted in red.
Edit Distance Interface (figure 8)	The edit distance interface displays the number of edits it needs to get from one implementation to the other. In figure 8, some of the values are 1, which means that 1 line of code is different for the python test file.

4.0 Implementation

4.1 Implementation Details, Technology, Softwares Used

Since the project involves data visualization and directory crawling, it was fair to decide on using Python as the main programming code. The version used was Python version 3.7.1. Other third party library includes Pmw, yapf, pandas, matplotlib, edit_distance, mplcursors, tkinter, and tkintertable. For the version-control system, gitlab that was provided from Iowa State University was used.

4.2 Rationale for Technology/Software Used

Pmw - Pmw is a toolkit to build high-level compound widgets. Since it helps with consistent look and feel between graphical applications and is also highly configurable, the team decided to use this library. Version used: 2.0.1

Yapf - Yapf is a formatter for Python files. To crawl through multiple Python files from different locations, Yapf seemed to be the suitable library to use. It was used to specify the formatting style for the name of the file. Version used: 0.28.0

Pandas - Pandas is a software library for data manipulation and analysis. By using this library, it was effortless to manipulate numerical tables and time series for the data visualization. To convert the JSON data to graphable dataframe, Pandas library was necessary. Version used: 0.23.4

Matplotlib - Matplotlib is a plotting library using the numerical mathematics extension NumPy. In order to generate various graphs for the visualization, matplotlib was necessary as it provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits. Version used: 3.1.2

Edit_distance - Edit_distance is a Python module for computing edit distances and alignments between sequences. In order to get a quick glance on how different one function is from another, Edit_distance library came in handy. Version used: 1.0.3

Mplcursors - Mplcursors provides interactive data selection cursors for Matplotlib. It was used for the interactive customization on selection cursors. By using this library, the users are able to click on different data to see the test event number, pass rate, result status, number of failures/passes, and the timestamps. Version used: 0.3

Tkinter - Tkinter is the standard Python interface to the Tk GUI toolkit. It is known for being able to output the fastest and easiest way to create the GUI and is flexible for the users.

Tkintertable - Tkintertable is the standard GUI toolkit for Python. It allows interactive spreadsheet-style tables to be added into the Tkinter application. By using this library, generating table visualizations was possible. Version used: 1.3.2

4.3 Applicable Standards and Practices

- P16085 - ISO/IEC/IEEE International Draft Standard - Systems and Software Engineering - Life Cycle Processes - Risk Management
- P15026-4 - IEEE Draft Standard - Systems and Software Engineering - Systems and Software Assurance - Part 4: Assurance in the Life Cycle

4.4 Resources

- Mock SVN repository. Collins (Branden Lange) built a repository for us to replicate what their SVN repository would look like. This was used to build our prototype and practice our directory crawler, file comparison, and edit distance functions. They had to do so because of security purposes.
- Mock STARWARS (test result data) json files. They also gave us two json files filled with test result data for us to build our product with, especially in the area of visualization.

4.5 Related Products, Literature

Some of the same functionality, but not completely similar:

- dupeGuru is a duplicate file finder for Windows. Although it is normally used to delete the extra copies of a file, it can also be used to just identify the duplicates and where they may be found. It is one of the best of it's kind. That is the extent of its usage though, it is unable to parse through a file to see if it has the same functions or any of the other necessary requirements for our project. (Lifehacker.com, 2019)
- Code Compare is a more complex software which takes the duplicate finding to the next steps. It is able to compare two files to see if certain lines of code match, with the ability to merge them together if needed. This could be used to make sure they have the same functions within the python files, and if so, are the functions implemented the same or will they produce different results. Again, this is as far as the usage goes as far as our product is concerned. It would be useful for comparing and merging files. However, this product costs $49.95. (Devart Software, 2019)

5.0 Testing, Validation and Evaluation

5.1 Test Plan

For this project, the main testing plans were integration testing, regression testing and user testing. User testing was mainly conducted at the research park since their data was not allowed to be on any machine other than the Collins Aerospace's machines. As for the non-functional test plan, the plan was to use Python version 3 for our program to be compatible with the already existing systems.

5.2 User-Level Testing

We had a demo session where we tested our tool against Collins Aerospace's real data because prior to that we were only using mock data provided by them due to corporate regulations. After this session we got feedback on what the client would like changed. Prior to that we had a demo with them but with mock data, we got good feedback as well on what could be added to the tool.

5.3 Integration Testing and Regression Testing

Since we had different modules in this project, we did integration testing to identify issues that came up from putting together different units. After new additions we also conducted regression testing to make sure no issues came up from new edits.

5.4 Evaluation

Our testing throughout the semester was successful for the most part, while integrating there was not much trouble linking different units, other than one issue described below, and the regression testing ensured there was no issues coming up from adding new features and when they were present we fixed the bugs.

However, when performing the user-level testing with Collins' real data, we found a bug that failed some of our functionality. As stated in Appendix III, the problem we ran into was that the key used for a test run in our mock data was the actual name of the function (ex. def greater_than()), while the key used for a test run in their real data is the test case id (ex.112341). Although it is not fixed yet, as the mock data has not been changed at this exact time, once changed it is an easy fix to gain back full functionality.

One other issue arose, as described in Appendix III also, while integration testing. We want to combine the edit distance values and the historical data table but just creating a new column for it. The problem, however, is that the paths for our mock SVN repository does not match with the mock test result data, as it uses the actual url of the SVN (which we do not have access to) rather than the local path of ours. For now, we keep these functions as two different pages, with documentation on how to combine them when we hand our product to Collins Aerospace.

Appendix I – "Operation Manual".

The project will be distributed as a source code file. The codes will be available on gitlab repository for easy distribution for the clients. After cloning the repository, the user should make sure that they have Python version 3.7.1 or higher. In order to set up, the user should be familiar with Pip installation. Installation of Pip can be done by downloading get-pip.py file to a folder on your computer:

curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

Then run the following:

python get-pip.py

Since pip is installed now, you are required to install the following libraries using pip install:

pip install pmw, yapf, pandas, matplotlib, edit_distance, mplcursors, tkinter, and tkintertable

Now you may open your favorite text editor for Python-our team used Visual Studio 2019-and run main_display.py. The alternative way to do this would be to open a command prompt for Windows/command shell for Linux/unix command prompt for Mac and change directory to where the project is cloned, and to run:

python main_display.py

Figure 3 would be the window that you will see after running the Python file, with empty text boxes.

First, write the function name, in the 'Function Name (test):' box, that you would want to see the results for. The program has an auto-fill ability to help you with the options. Click the search button after selecting/typing one of the functions. It will then display different Python files that has the selected function in it. Double-click on one of the files that you would want to see the result of. If you want to compare that file alone, you are good to go. If you would like to compare it with another file, click on the Python file that you would want to compare and proceed to the next step.

The first action button is the 'Graph' button. From the middle section of the graph window, select one of the JSON files that contains the test results from STARWARS database and hit the 'RUN' button. That will give you several test events on the left side of the graph window. You can either select one test event and hit the 'Graph Individual: Pass and Fail Rate' or select multiple test events and hit 'Graph Comparison between test events: only pass rate' or 'Graph Comparison between test events: only fail rate'. You may also generate customized graphs by controlling the x-axis and y-axis with various factors on the right side of the graph window and hitting the 'Custom Graph' button. The graph is interactive in a way that it will show you the event number, pass rate, status of the test result, number of failures, number of passes, aircraft group id, aircraft type id, disposition, duration, end time, log file path, notes, script history location, script version, software build, start time, test case id, test run id, and total number of verifications of the point you may click. This pop up information will close if you right click on it. By utilizing the buttons on the bottom left corner, it is possible to zoom in/out of the graph, save the graph, and customize the shape, font, or size of the output. Refer back to figure 3.

The second action button is the 'Compare it' button. This action button is straightforward. It will compare the selected function in different files, which would also mean that the user is required to select more than two Python files. If you refer to figure 7, the red highlight would mean the deletion of the original file, and the green highlight shows the added/changed part of the code.

The third action button is 'Historical Result' button. This action can be done by simply clicking the button. It will show all the edit distance, last run date, overall percentage, program name, test run number, and script history of all the files with the selected function name as you can see from figure 6.

The fourth action button is 'Edit Distance' button. By simply clicking the button, it will display a window that will show all the distance of files that contains selected function. For example, 0 means the functions are the same. The original values are based on the distance of the top-most file path. If you would like to see the distance compared to a different file, simply click on that box to highlight the desired path and press 'Select'. The Treeview will then update with the new values. Refer back to figure 8.

Appendix II: Alternative/ other initial versions of the design (and why they were scrapped in favor of the current version).

- Initial design for the project was to make it as a Windows application. However, the team did not want to risk the possibility of having compatibility issues with the Collins environment.
- We had originally planned on having an intermediate Mongo Database to store test result data (also described in Appendix III), however we found it unnecessary with our current functional requirements. If in the future they wanted to keep a track record on specific functions or files, this may come in handy, but we were able to save time and system efficiency by pulling data straight from the test result data.

Appendix III: Other Considerations.

What we learned

- The availability of resources (data) is very important
    - With all of the security restrictions at Collins Aerospace and their data, we were unable to see or use any of their real working files and test result data. If we could look at it, it was encrypted into nonsense. So we needed a way to replicate that data so we could still use it in order to build our prototype and test various aspects to see if we're on the right track. We tried to make our own given a stub of what a test run will give us. However it was hard to replicate the examples of the type of data we're actually getting without seeing an example and it was also hard to replicate the size of what we'd actually need to be working with.
    - Collins decided to give us mock data to work with, one being a mock repository with python files to replicate the SVN and another being mock test result data in the form of JSON files. This helped a lot to work with a larger size sample and work with actually accurate data (what we were hard coding wasn't great), however it came with a couple issues, some that we are still dealing with. One, the file paths in the mock test result data didn't line up with the repository they gave us. The test result data gave an SVN path, while ours was local. This came into play when trying to compare files, so the edit distance and comparisons had to be completely separate from all other data because of this, which is functionally inconvenient. The second issue, which was just found recently, is that in the

test result data, they were classifying test runs under the actual name of the function being tested on, when really it should have been the test case id. So when we tested on actual data of theirs, it failed.
    - This isn't being said to throw Collins under the bus, as if we were actual employees of theirs this wouldn't be an issue because either we would have access to those SVN and STARWARS repositories, or we'd have the ability to build our own mock data given access to examples. This could also have been solved if the mock data was given to us earlier so we could identify the problem sooner. Again, not trying to bash Collins Aerospace, but now we understand the importance of being able to access such data to build and test, to identify risk/problems earlier and make sure we continue to be on the right track.
- Communication is key
    - We had great communication throughout these past two semesters, having meetings every week and asking questions back and forth. There were a couple aspects we could've done better however. One being defining technical terms. We had tried to do so in one of our initial meetings, however when we changed our project leader, we sort of threw that out the window. We should've came back to it again to reassure those terms and possibly create better language as some of our conversations became confusing because of that.
    - Another way our communication could've been better is our own ability to challenge the Collins team. By challenge I mean offering alternatives to solutions or asking if some aspects are completely necessary. We assumed their initial idea of the solution was the only way to do it. Such as having an intermediate database to store data from STARWARS when we could save a step and just grab and analyze the data straight from the test result database. We ended up asking this late in the semester, but could've saved a lot of time and thought by asking earlier.
- Start practicing with new libraries early, specifically when working with multiple together
    - We did learn how to use most of the libraries fairly early actually, unless we hadn't decided on using it yet. However, it was combining them together that took some time to learn. Such as working with Matplotlib and Tkinter together became more difficult than just using them individually.