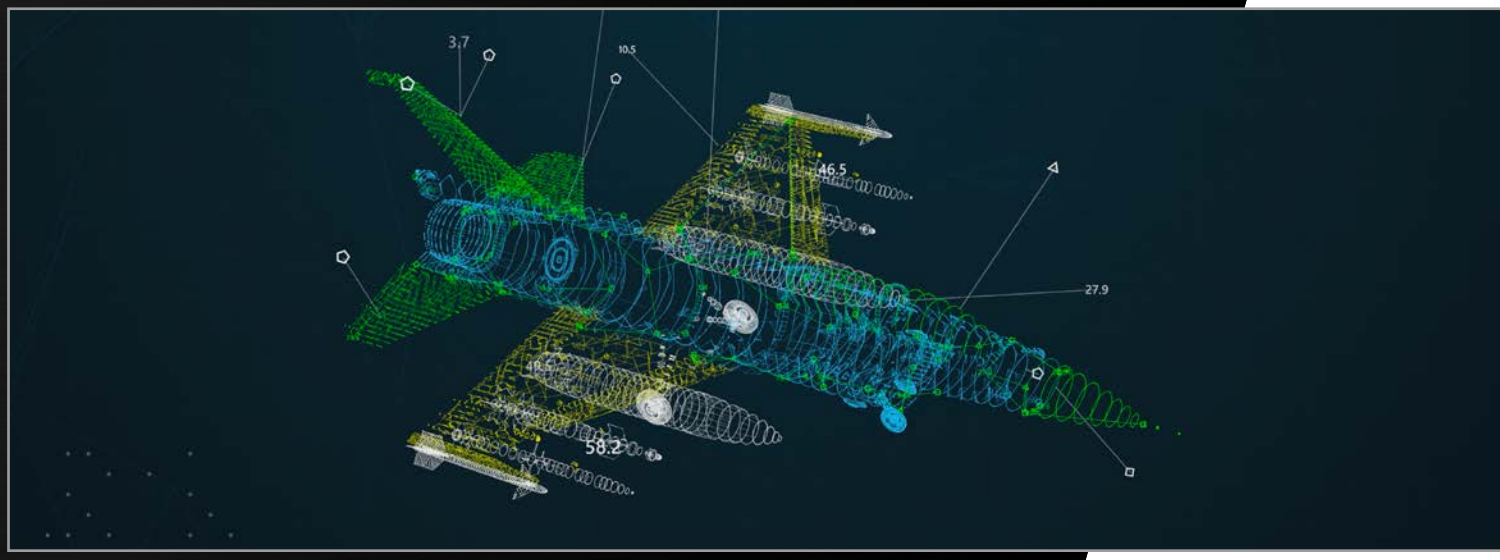
The Challenges & Advantages of a Unified Approach to Post-Flight Test Analysis

Post-test analysis in the flight test community tends to be a fragmented and project-specific endeavor. Engineers commonly spend enormous time and manpower building custom tools that satisfy the current project's needs. However, these tools are often not reusable by subsequent projects. If a set of common core requirements across many post-test projects can be established, it will be possible to engineer a generic, reusable tool. Such a tool can significantly reduce the time to conduct post-test data querying, analysis, and reporting. This white paper highlights these common requirements and suggests solutions that Curtiss-Wright has codified in a new software tool to meet the industry's needs.

Contents



curtisswrightds.com
ds@curtisswright.com

The problem with program specific tools

Modern flight test programs generate significant amounts of data for every flight, which must be interrogated for various reasons, including final reporting. Typically, tools to perform the necessary data search, analysis, and report generation are custom developed for each program.

Existing tools import data from an offline file format, perform custom analysis, and produce project-specific data plotting/results. The imported data is often sourced from another system, requiring the user to request the data by time, flight, and parameter. The turnaround time to receive the requested data can be anywhere from hours to days in a worst-case scenario. These tools often contain little or no search capabilities, no access to flight metadata, or the ability to browse the data in an ad-hoc manner.

Generally, little focus is given to making these systems more generic, mainly due to time constraints and development costs. As a result, subsequent projects rarely (if ever) can take advantage of these existing post-test tools due to a lack of documentation, support, and tool flexibility. This leads to new projects being forced to repeat the process of building their systems, which costs additional time, money, and manpower. The resulting system tends to be very similar to its predecessors in terms of project-specificity and lack of expandability. As the cycle repeats, older post-test systems are rendered obsolete while new ones are created without being able to take advantage of the work that went into their predecessors.

Defining a unified post-test system

Despite the custom nature of program-specific systems, there is some commonality between them that can be used to create a more generic, robust, “unified” post-test system that could benefit multiple flight test projects. The biggest challenge of such a system is to bring together the collection of requirements gathered from multiple flight disciplines into a single application that can be used throughout the flight test community as a whole. The following sections will examine the overlapping requirements necessary to make a unified post-test system and some approaches to satisfying those requirements.

Data querying

The first common requirement is the ability to search large amounts of data easily and quickly. Many flight test programs have years’ worth of flights stored on their servers, the sum total of which can add up to multiple terabytes of data. In some cases, engineers need to see search results visually to determine their investigation’s direction. Additionally, data queries can become quite complex as engineers dig deeper into the data to find exactly what they’re looking for.

The capability to search flight data across a combination of maneuvers, flight conditions, and flight metadata is necessary to allow engineers to complete their analysis effectively. Many existing systems currently use some sort of formal, structured programming language to query the data (e.g., Structured Query Language (SQL)). While this works, it requires the engineer to know the precise syntax and logic of the language. Alternatively, some systems offer a way to build these formal language queries through a series of user interface dialogs. While these methods provide a means of querying the data, they can be slow and cumbersome processes as the search queries become more complex and specific in nature. The sheer volume of data being searched, coupled with the broad range of topics to query, makes it clear that an intuitive and efficient searching tool is one of the most crucial aspects of any post-test system.

Data analysis

The second common requirement is the need to perform analytical calculations on data from multiple flights. Much overlap exists across disciplines in terms of a "standard" library of mathematical functions required to accomplish this task, including categories of mathematical calculations such as elementary mathematics, linear algebra, discrete math, and statistics. A unified post-test system that hopes to accommodate multiple flight disciplines needs to provide these functions as a starting point.

Additionally, discipline-specific mathematical functions are required to round out this library and provide the engineers with a complete mathematical package that satisfies their needs. For example, flying qualities engineers typically require functions that focus on maneuvers such as doublets, push-over pull-ups (POPU), and wind-up turns (WUT) to compute moment coefficients, damping, gradients, and stability metrics. On the other hand, structures engineers tend to focus on maneuvers such as elevated G rolls, buffet and forced excitation to compute items like peak loads and modal characteristics. This trend of discipline-specific analysis requirements continues across numerous other flight disciplines.

Since each discipline requires its own specific set of mathematical functions, the task of providing a "complete" data analysis package becomes quite tricky. Ultimately, it's impossible to provide every conceivable function that a flight test engineer might ever need for their given discipline. This dilemma yields yet another requirement – the ability to allow engineers to write their own custom mathematical functions and easily incorporate them into the existing library used by the system.

Regardless of whether these data analysis functions come from a pre-written library provided by the system or if the engineers themselves write them, the source code must be open for viewing by all users of the system. Functions cannot be provided as a "black box" as this prevents those using the system from seeing exactly how the calculations are being made, thus reducing the confidence level of the returned results. Allowing engineers a way to view the source code of the data analysis library gives them the means to validate their results. The ability to view and edit these functions provides users with the means to verify the integrity of the calculations that the system is making and make corrections as they see fit.

Report generation

The ultimate goal of post-test systems across all disciplines is to create a report that contains the results of the flight test engineer's investigation into the flight data. These reports can include anything from the initial data queries (e.g., what maneuver, flight conditions, and metadata was requested), the data analysis results, and any of the steps in between.

A prominent aspect of these reports that overlaps all flight disciplines is the ability to create plots – detailed graphical visualizations of the data resulting from the engineer's analysis. Being able to plot data across multiple flights allows the engineer to identify trends and patterns in the flight data that wouldn't otherwise be obvious from viewing a single flight alone. To showcase all of this information, a number of different data visualization elements are used. The most common graphical elements include time history plots, frequency plots, and cross plots.

In addition to a wide range of graphical plot types, the ability to shape and customize individual attributes of these plots is also required. Attributes such as line color/thickness, axis granularity, and scale all must be customizable to produce the desired visual output. Any post-test system that aims to satisfy the plotting needs of all flight disciplines must be flexible enough to provide a built-in collection of commonly used graphs and the ability to add/modify the attributes of those graphs.

Approaches to solving the common requirements

In recent years, Curtiss-Wright's IADS team set out to determine what sorts of systems currently exist for the analysis of flight data in a post-test environment to determine what common requirements exist and how these could be satisfied. This task led the team to meet with several engineering groups from different flight disciplines to discuss each group's requirements for their respective post-test systems. The goal of these meetings was to discover what, if any, commonality exists between the systems currently in use and any overlap between the requirements needed across these different disciplines. From these discussions, the following approaches were developed to meet these needs.

Natural language as a data querying tool

Discussion with different flight disciplines revealed that the most common queries of flight data involved the following: maneuvers/test points, flight parameter data, and flight metadata. In the simplest of cases, a data query might be looking for information such as when a given maneuver was flown, when a particular flight condition occurred, when specific metadata criteria were met, or any combination thereof. Requiring users to know a formal structured language can make even the simplest of these queries somewhat cumbersome. For example, in a database query language such as SQL, instances of "simple" queries might be:

- SELECT * FROM Flights WHERE Maneuver = 'Wind Up Turn'
- SELECT * FROM Flights WHERE Mach > 1.0
- SELECT * FROM Flights WHERE Pilot = 'smith'
- SELECT * FROM Flights WHERE Maneuver = 'Wind Up Turn' AND Pilot = 'smith' AND Mach > 1.0

In many cases, minor typos or syntax errors can render a given query useless. For example, a particular database table might be mistyped (e.g., "Flight" or "Flihgts" instead of "Flights"), a maneuver name might be ambiguous (e.g., is the maneuver called "Wind Up Turn", "Windup Turn", or "Wind-up Turn"), or the language syntax may not be followed properly (e.g., the word "SHOW" is used instead of "SELECT"). Such is the nature of formal languages – if the specific format of the language isn't followed precisely, it can prevent results from being returned for reasons that may not be immediately obvious.

What if the end-user was allowed to enter a less formal, more "free form" style of query, akin to a request that one might make from a search engine on the web? Using a natural language processor, as opposed to a formal language, addresses many of the aforementioned issues. Rather than being required to remember the exact syntax and format of a given language, a freeform natural language query is more intuitive to write and easier to remember. Take the formal query from above:

- SELECT * FROM Flights WHERE Mach > 1.0

And compare it to the following natural language alternatives:

- Show me the flights where Mach went greater than 1.0
- Which flights did Mach exceed 1.0
- When did Mach go above 1.0
- Mach > 1.0

Natural language queries not only make query requests simpler, but they're also more easily understood by the user. No knowledge of a given language is required to parse and understand what is being asked because that burden is placed on the natural language processor. Without a formal syntax to adhere to, the engineer is allowed more ways to ask the same query. Additionally, some natural language processing libraries provide text correction capabilities (e.g., TextBlob[i]) to catch and correct common typos/misspellings that would otherwise render a query useless.

User-defined data analysis

The ultimate goal of querying the flight data is to ascertain a collection of time ranges across multiple flights that satisfy a given flight condition, maneuver, or collection of metadata. Once these time ranges have been retrieved, analysis needs to be performed on data within these time ranges that allows engineers to identify trends and patterns that wouldn't be visible otherwise by looking at a single flight. A unified post-test system needs to provide a broad collection of built-in mathematical functions that enable engineers to perform this analysis. Among the categories included in this general-purpose collection are elementary math, linear algebra, statistical functions, as well as time and frequency domain functions.

Additionally, discipline-specific mathematical functions will likely be required to allow engineers to fully process the time slices returned from the query stage. While a system can offer many of these functions in tandem with the more general library outlined above, it's impossible to deliver every mathematical function that every engineer across every flight discipline could conceivably need. This means that a unified system, besides providing a robust library of general-purpose built-in data analysis functions, must also provide a way for the end-user to create their own custom mathematical functions that can be used for data analysis.

Currently, the two most prolific programming languages used to write scientific and numerical analysis functions are MATLAB© and Python©. MATLAB is already heavily established in the flight test community, with many existing projects currently using the MATLAB environment to write their own data analysis code. A number of libraries written in MATLAB exist to perform scientific analysis and generate plots/graphs as well. On the other hand, Python has been enjoying increasing acceptance in the flight test engineering community in recent years[ii].

Similar to MATLAB, Python also offers a wide array of software packages such as SciPy, NumPy, and Matplotlib[iii] that can be used for scientific analysis. Choosing which language to use to allow engineers to develop their own custom scientific analysis routines is a subjective decision, as all languages have their own set of pros and cons concerning features, flexibility, and cost.

Whatever language is chosen, ideally, the post-test system provides an integrated development environment (IDE) for the engineer that allows them to seamlessly create their own functions, interact with the flight data directly, troubleshoot/debug their code, and also publish their custom routines so other engineers can use them (and edit them if necessary). At a minimum, a capable IDE needs to provide features such as syntax highlighting, code auto-completion, breakpoints, the ability to step through code line-by-line, variable evaluation, and stack trace information. While many of these features are satisfied by using an external development environment (most external environments have capabilities to create and debug code as outlined above), there are several benefits to having an IDE built directly into the post-test system.

In many of the existing post-test systems that were evaluated, exchanging data between the system and the external development environment typically involved exporting data to a predetermined file format (e.g., Comma-Separated Value (CSV), MATLAB, Hierarchical Data Format (HDF), etc.), and then having the external environment import and process that data. With an integrated environment built into the post-test system itself, the need to export the data to an external format goes away, and the integrated environment can more efficiently process the raw flight data. Not only does this streamline the analysis process, but it also protects against precision/rounding errors that can be introduced into the data when being translated between file types.

Report generation

In most cases, the final step of the post-test analysis process is for the engineer to produce a report of the investigation that they have created. This report is the sum total of the information gathered over the course of the investigation. This can include a wide range of information such as flight data queries and the results of the data analysis routines that were used to perform any scientific analysis. Ultimately, a graphical representation of these results is needed to illustrate visually how any trends or patterns developed after performing the analysis across multiple maneuvers/flights.

Many different data visualization elements are necessary to convey this information in the final report. These elements can include (but are certainly not limited to) common graphs such as time history plots, cross plots, frequency plots, pie charts, and bar graphs. In addition to a wide range of graphical plot types, the ability to shape and customize individual attributes of these plots is also required. This includes attributes such as data color, marker style, and size, axis ranges and labels, gridlines, legends, and annotations.

A post-test system that wishes to provide the capability to produce these kinds of reports needs to offer what amounts to the same functionality as a modern-day word processor. The ability to insert and format text, images and the aforementioned graphical elements must be provided. The system must also allow the user to arrange this information flexibly in a multipage environment. Headers, footers, automatic page numbering, and spell-checking all need to be provided as well. The finished report must be able to be exported to common, established output file formats (e.g., Portable Document Format (PDF)), as well as be printed directly from the system itself.

Report templates are another important feature that must be included. Much of the work that goes into building the final report is determining how the data should be presented – where plots will be placed on each page, how many plots will be shown per page, how many parameters are shown on each plot, common headers, and footers, etc. Once created, this arrangement of graphical elements will likely be useful to engineers not only during the current investigation but also in future investigations. Therefore, the system must be able to save this layout as a template for reuse in the future.

Lastly, the system must provide a "live" connection to any data that is being visually represented from the search/analysis portions of the application. Since the data shown on these plots is being read from the output of flight data queries and data analysis routines, any changes to those input elements need to be automatically updated and "pushed" to the graphical elements on the report.

These changes can include modifications to the initial query of the flight data (e.g., change the maneuver or event that is being analyzed), as well as any updates to any of the data analysis calculations being performed (e.g., updates to the code making the calculations). Seamless integration of all of these aspects of the system ensures that the data being represented in this final step of the investigation is never stale or outdated. Users are free to make modifications to the earlier steps of their research without worrying about the accuracy of the data being shown in the final report.

Curtiss-Wright's Post-Test Explorer

The following section will illustrate Curtiss-Wright's new Post-Test Explorer solution using a hypothetical post-test use case scenario where an engineer is trying to find a collection of "Yaw Rap" maneuvers in their flight data. The goal is to calculate the frequency and damping of various tail accelerometers on the aircraft during this particular maneuver when Mach exceeded 0.8, and the altitude is 10000 feet.

The first step is to allow the engineer a means to search the flight data for specific maneuvers. Using the natural language techniques described earlier, a sample query of this nature might be like: "show all yaw rap maneuvers." With the flexibility of NLP, this same information could be yielded with other queries such as:

- "show yaw rap maneuvers"
- "yaw rap maneuvers"
- "yaw rap"

In this example, the system has returned a collection of 10 results (two of which are visible in Figure 1) that represent time ranges during which the specified maneuver was flown. To visualize what is happening during this time range, the system has provided a collection of simple plots of parameter data associated with the maneuver (TailAccel1-4). Additionally, some general flight conditions have been provided: Mach, altitude, knots equivalent airspeed (KEAS), and roll rate. Showing maneuver-specific data and general flight conditions gives the user a better sense of situational awareness about the execution of the maneuver.

Suppose the user deems that the output results are too general or sees a specific occurrence in the data that interests them. In that case, they are free to refine the query as necessary and search the system again to limit the results further. This process is repeated until a specific area of interest, or set of conditions has been isolated in the search results. Moving forward with this example, the query has been refined to limit the search results to the aforementioned flight conditions as follows:

- "show all yaw rap maneuvers where Mach > 0.8 and Altitude is 10000"

Once query results have been refined and the specific areas of interest identified, the next step in the process is to perform data analysis. In this particular example, the goal is to try and find the frequency and damping of some of the accelerometers on the tail of the aircraft during a yaw rap maneuver. When the user has sufficiently limited the number of query results to their liking, they can be viewed in a matrix that displays all of the results in a tabular format (see Figure 2).

Figure 2: An example search query, showing two results (out of ten returned).

This matrix displays all of the results returned from the query, with each row representing a single query result (time range). Initially, the start and end times of the query result are displayed, along with some pertinent flight metadata (flight number, test number, tail number, and flight date). The user can add new columns of information that both show parameter data and perform analysis routines on parameter data in these time ranges. A user interface is provided to allow easy browsing between all available flight data parameters, built-in analysis routines, and user-defined analysis routines.

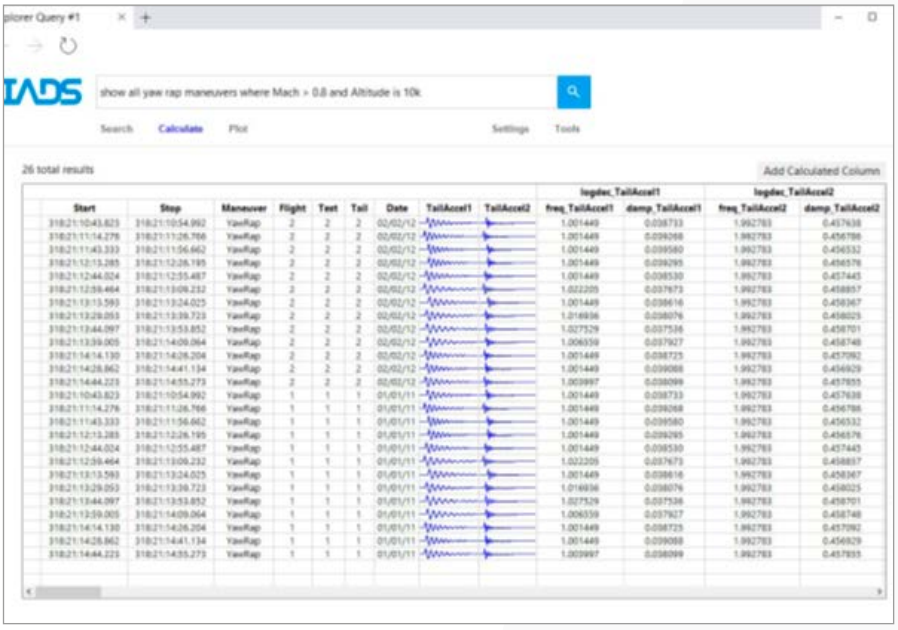


Figure 3: Completed analysis matrix showing the frequency and damping of two accelerometers.

In this particular example, the accelerometer data of interest has been added to the matrix and columns that calculate the frequency and damping of these accelerometers from the system's built-in analysis routines. Once an analysis routine has been selected, the code defined within it is run automatically across all query result time slices. The results are output to columns in the matrix. This scenario is shown in Figure 2, with two accelerometers – TailAccel1 and TailAccel2 – each being added as a column to the matrix. Additionally, each accelerometer's frequency and damping values have been computed via the system's logarithmic decrement function, with each result being added as columns as well.

Once query results have been refined and the specific areas of interest identified, the next step in the process is to perform data analysis. In this particular example, the goal is to try and find the frequency and damping of some of the accelerometers on the tail of the aircraft during a yaw rap maneuver. When the user has sufficiently limited the number of query results to their liking, they can be viewed in a matrix that displays all of the results in a tabular format (see Figure 2).

This matrix displays all of the results returned from the query, with each row representing a single query result (time range). Initially, the start and end times of the query result are displayed, along with some pertinent flight metadata (flight number, test number, tail number, and flight date). The user can add new columns of information that both show parameter data and perform analysis routines on parameter data in these time ranges. A user interface is provided to allow easy browsing between all available flight data parameters, built-in analysis routines, and user-defined analysis routines.

In this particular example, the accelerometer data of interest has been added to the matrix and columns that calculate the frequency and damping of these accelerometers from the system's built-in analysis routines. Once an analysis routine has been selected, the code defined within it is run automatically across all query result time slices. The results are output to columns in the matrix. This scenario is shown in Figure 2, with two accelerometers – TailAccel1 and TailAccel2 – each being added as a column to the matrix. Additionally, each accelerometer's frequency and damping values have been computed via the system's logarithmic decrement function, with each result being added as columns as well.

Once the analysis has been completed, the final step of the process is to build a report that visually displays the calculated frequency and damping values. Figure 3 shows an example of a simple report that shows the frequency and damping results from each accelerometer plotted onto cross plots. Text editing (font, style, alignment, etc.) and plot types are shown at the top of the report building interface, while individual plot properties (symbol color, size, etc.) are on the right side of the page.

As columns are created in the analysis matrix, the results become available to be plotted on the graphical elements provided by the report page. In the report displayed above, each of the frequency and damping columns output from the logarithmic decrement function in the previous step are represented on the cross plots. In the first cross plot, the freq_TailAccel1 and damp_TailAccel1 columns are plotted against each other. Similarly, the second cross plot shows the data from the freq_TailAccel2 and damp_TailAccel2 columns plotted against one another. Any column that is present in the analysis matrix can be placed on any of the available plot types and added to the report. Plots are free to be arranged on the page in any manner the user desires. To round out the contents of the final report, plots can be supported by other elements such as text, images, and annotations. Once a page layout has been set up, that layout can be saved as a template and reused in future investigations. Once finished, the user can print the final report or save it to a standard file format such as PDF.

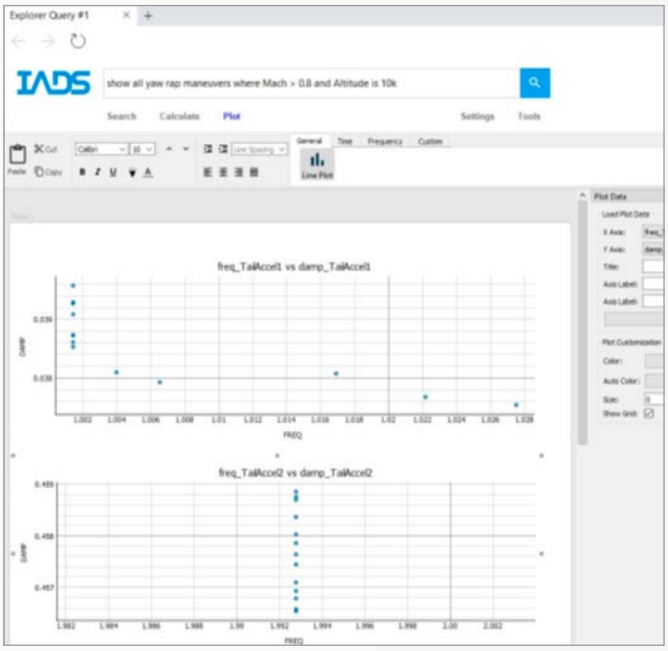


Figure 2: An example search query, showing two results (out of ten returned).

Conclusion

Post-test data exploration and reporting can be difficult and time-consuming, especially today when more and more data is being collected. The typical approach of using custom made tools for every program is time-intensive and often lacks some features that are too resource-intensive to create given the time constraints of each individual program.

A collaborative framework that solves many of these problems would need to seamlessly integrate the ability to search large amounts of data, perform analysis on the results, and generate reports from that analysis is the backbone of a unified post-test system. Providing a core library of standard analysis routines and plotting capabilities while simultaneously giving users a way to create their own custom routines and plots allows the system to be helpful to engineers across multiple disciplines.

Curtiss-Wright has been working to include these features in an easy-to-use, standardized application that aims to streamline the post-test analysis process significantly with Post-Test Explorer. Using a system like this in new flight test projects could potentially reduce the large amount of time, money, and manpower that is typically devoted to designing and building new systems that meet that project's specific needs. This software has the potential to empower flight test engineers and benefit the flight test community as a whole.

References

> i Simplified text Processing: https://textblob.readthedocs.io/en/dev/

> ii Bretz, John: "Progress in the Migration of Flight Test Analysis Routines to Python", The ITEA Journal of Test and Evaluation 2018.

> iii SciPy available from https://www.scipy.org/, NumPy available from https://numpy.org/, and Matplotlib available from https://matplotlib.org/

curtisswrightds.com
ds@curtisswright.com