# Application of Software Technology to Automatic Test Data Analysis



**J.R. Stagner**

Jet Propulsion Laboratory
California Institute of Technology

**Abstract.** The verification process for a major software subsystem was partially automated as part of a feasibility demonstration. The methods employed are generally useful and applicable to other types of subsystems. The effort resulted in substantial savings in test engineer analysis time and offers a method for inclusion of automatic verification as part of regression testing.

## INTRODUCTION

One area of interest for the application of new software technologies is the automation of labor intensive functions performed as part of the overall testing process. A specific problem area is analysis of the results from test runs to verify correctness of software under test. Currently, most analysis of data resulting from test runs is performed manually. Data from one test run can take up to 60 hours to analyze. Hence, this is a candidate problem area for machine-aided or automatic analysis. The work, described below, was performed as a proof of concept experiment to demonstrate the feasibility and usefulness of rapid prototyping and production system programming techniques in the solution of this recurrent class of test problem. The expected benefits are to conserve test resources and improve the quality and thoroughness of test analysis.

The software selected for this proof of concept experiment was the Data Monitor and Display (DMD) subsystem (Figure-1.), which is part of the new Space Flight Operations Center (SFOC) under development at JPL. The SFOC is a multi-mission ground data system. The DMD is an end-point in the Ground Data System telemetry data flow and provides visibility into the data processing. It's major input is channelized telemetry data (TLM). It's major outputs are displays of the data in human readable form, including a hard-copy dump called the Latest Available Data (LAD). The DMD makes use of several tables, contained within the Channel Parameter Table (CPT), coefficient table (COEF), and

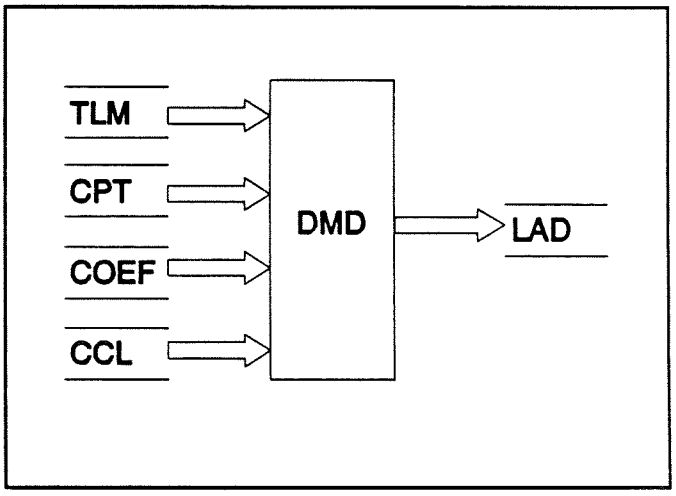


**Figure 1.** DMD Test Run Data Flow

Channel Conversion Language (CCL) files.

The raw telemetry data (TLM) are in Data Number (DN) units. Data numbers are converted to Engineering Units (EU) by means of a table lookup or polynomial computation. Conversion tables and polynomial coefficients are contained in the COEF file.

The CPT contains, for each channel, the data type (integer, unsigned integer, floating point, status, ASCII, digital data, etc.), the subsystem where it originated (usually a spacecraft subsystem) and any limit tests that are to be performed.

CCL is a language that specifies processing for each channel. Some examples would include channels derived by (1) averaging three other channels, or (2) taking the arc-cosine of a channel and adding a constant factor or (3) multiplying a channel by a constant to convert from radians to degrees.

The LAD contains a dump of raw telemetry (input) DN, DMD computed EU values and alarm states based on DN and information contained in the CPT, COEF and CCL files. This information is provided over a time window established by the test engineer in the data collection process.

To verify that the DMD correctly computed an EU value for a given channel, the CCL listing must be examined to determine if any processing has been applied. Relevant information from the COEF listing is used to manually compute the EU value so it can be compared with the DMD computed value contained in the LAD listing. Similarly, the alarm state indicators in the LAD data must be verified by manually applying the alarm limit criteria found in the CPT listing. The problem of verifying the approximately 3400 data channels, for just one mission (Magellan), becomes apparent.

Manual analysis of test results is usually a straight-forward but laborious task. We would like to have a computer program that does what the test engineer (TE) does. That is, stand in the place of the TE and do the analysis. We would expect it to perform, in minutes, what the TE would do in days and the results would be more accurate and of higher quality. It would be economically feasible to systematically analyze all the data channels, for every software delivery throughout the software lifecycle, instead of using a sampling technique.

A program like that needs the TE's knowledge about all of the various elements that he checks when he analyzes data channels. In the DMD, for example, the analysis program must be able to read the electronic versions of the various listings, determine how to convert digital numbers (DN) to engineering units (EU) and, given an EU, how to determine the alarm state. With this knowledge and supporting facts about alarm limit types and values and coefficients or tables to be used in the DN to EU computations, the program should be able to examine output from DMD, channel by channel, determine what DMD thinks the EU and alarm states are, decide if the TE would agree with the answer produced by DMD and signal the results of this analysis somehow.

## PROGRAMMING TOOLS

The programming tools used in this work were AWK, CLIPS and UNIX. AWK (Aho, et al. 1988) is a programming language that is well suited for exploratory programming. AWK syntax is close to C and provides simplified program control, I/O, character string functions, and regular expressions, which make AWK very useful for low-level data filter operations and report generation.

The "C" Language Production System (CLIPS-4.3) (Giarantino 1989) was developed by Johnson Space Center (JSC) and is the expert system shell used for this task. It provides a forward-chaining inference engine and interactive development environment in support of the prototyping, execution and debugging of knowledge bases. CLIPS syntax is similar to other production systems (e.g. OPS5, OPS83, ART). Interactive production system programming techniques supported by CLIPS encourage the acquisition of knowledge about the analyses and sub-problems in manageable units which are encoded in the form of rules. Each rule is easily modifiable, immediately executable and its effects on the analysis can be quickly evaluated. The turn-around time from concept to evaluation is quite rapid. Since rules execute opportunistically, control is hidden and knowledge is explicit. Therefore, rules are generally easier to understand than an equivalent "C" program where program control must also be made explicit within the code.

A significant benefit of both AWK and CLIPS is portability between PC's and UNIX work-stations. Much development was performed on a PC/AT (MSDOS 3.3). The AWK scripts and CLIPS knowledge bases were subsequently ported to the host computer. For some general guidelines on the types of problems suitable for solution using a production system, see (Brownston, et al. 1986) pp 19-29.

The UNIX Cshell (Sobell 1985) was used to integrate all of the AWK data filter and CLIPS analysis functions to define the end-to-end processing task (Figure-3). Cshell provides an impressively simple, high-level and flexible mechanism for integrating CLIPS into a traditional programming environment to perform an analysis task. It also provides many other useful services, in a simplified fashion compared to an equivalent C program. For example, tests were performed to verify that the input files physically exist before the analysis is started. Timing information is displayed for performance measurements. The DN-EU and alarm-limits analyses are forked as separate sub-processes to run in parallel and take advantage of input/output overlap. The entire analysis process can be run in the background, thus clearing the terminal for other work.

## APPROACH

### Problem Definition

Initially, the problem was ill-defined. The number of analyses was fixed at two, but the number of sub-problems was unknown. This characteristic points to an evolutionary programming approach. Knowledge acquisition was performed by means of an interview, prototype and evaluate cycle. Incremental improvements in the analysis capabilities were made at each cycle. This process quickly produced an end-to-end analysis capability, for a large subset of data types.

The DMD TE was interviewed to gain an understanding of the contents and format of the data to be analyzed and the steps he uses in verifying that the results are correct. A prototype was developed using the AWK and CLIPS programming languages (described below) to create a set of functions that perform the analyses so that the TE could see some results. This initial prototype was built around the CLIPS expert system shell. Thus the initial prototype clarified functional issues and was highly interactive and programmer-oriented.

From this initial prototype, it was learned that the TE really wanted a UNIX command-line function that could be used to analyze multiple sets of input files. Therefore, the final prototype addressed encapsulation issues and became user-oriented, taking into account how the TE wanted to perform actual work. That is, operational details were hidden in order to simplify the user interface.

## Merged Data Base

In order to help the test engineer articulate the steps he performs during manual analysis of data, a tool was written (Figure-2) which merged all of the information for each of the given channels from the diverse inputs. The goal of this tool was to provide a displayable database with information clustering and minimal filtering.

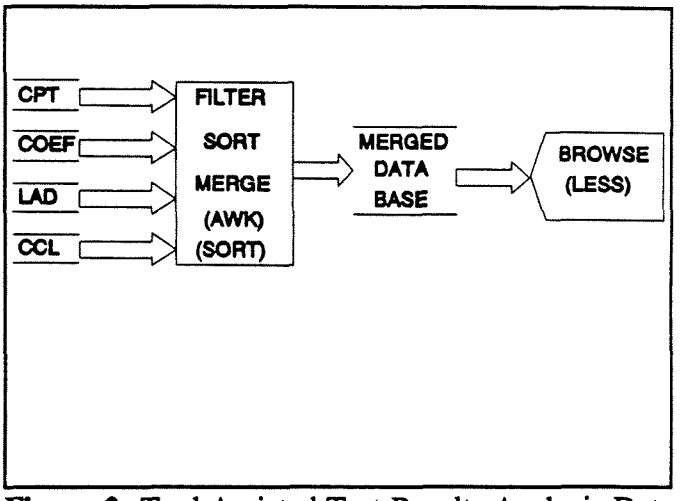


**Figure 2.** Tool Assisted Test Results Analysis Data Flow

The merged database product is probably the best way to begin the knowledge acquisition process for this kind of problem. Bringing all of the relevant information together at a single point is requisite for any kind of analysis program. It also helped the test engineer organize his thinking about how he analyzes channels and avoids references to four separate stacks of printout, a time consuming process. All of the relevant information is collected together in a single display. This minimally filtered product facilitated the discovery process of ways to partition the overall analysis problem into sub-problems and their associated analysis technique. We were able to map these analysis sub-problems into software modules implemented either by AWK programs or CLIPS rule bases.

The merged database product proved not only to be useful for gaining insight into the analysis process, but it also provided a dramatic improvement in manual analysis productivity. It is estimated that the manual analysis of 1000 channels previously took 20 man-hours. Using this reformatted data set, manual analysis could be performed in about 4-hours. Improvement in productivity was achieved by consolidation of information between the four different listings and the clustering of information distributed within the listings.

## Analysis Data Flow

Figure-3 shows a Data Flow Diagram for the final analysis prototype. Although the DMD data listings are human readable, the formatting did not allow for easy detection of problems by humans. It is easy to miss problems buried in many pages of printout. Embedded character strings add clutter which make it even more difficult to notice anomalies in large volume printouts.

The filter program was designed to automatically sense the type of input and format the relevant data from the CPT and COEF files as CLIPS-world data structures (fact-lists). The relevant LAD data are output as ordinary lines of data to be read by CLIPS, one at a time, and analyzed.

The number of CLIPS-world records are counted for comparison with a count of input COEF, CPT and LAD records. This provides a check to verify that no data were lost in the filtering process. The CPT fact-list and alarm knowledge base are input to CLIPS as part of the initialization procedure for an alarm analysis. Similarly for the COEF fact-list and DN-EU knowledge base.

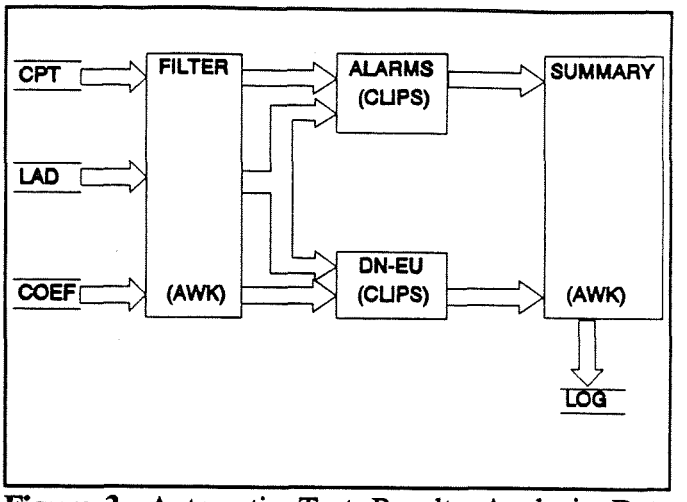


**Figure 3.** Automatic Test Results Analysis Data Flow

## Filtering

Data filtering is a necessary part of the overall problem solution. Too little filtering allows too much bad or irrelevant material to pass and requires more analysis (human or otherwise) to validate the results. Too much filtering can obscure some valid results. The task of filter programs is similar to their electronic counter-parts, that is, to eliminate irrelevant information such as page headings, blank lines, unwanted keyword-like character strings, new-page characters, etc. They also convert multi-line records to a single line format in order to further simplify subsequent processing.

The format structures for CPT, COEFF and LAD data, were similar since they were computer generated. That is, single or multiple-line records with interspersed page headings, footers, and short, blank or null lines. The fields within records sometimes consisted of character string data identifiers, like "RED" for red-alarms and "HI" for the upper alarm limit etc., followed by an equal sign and the data-item itself. Some data items are given as a list, where the list items are separated by a comma or white space delimiters. It should be noted that the test engineer has considerable flexibility in formatting these outputs. For example, he can choose the data-item mnemonic, spelling and case, or none at all.

The filters described below were "tuned" for the particular printout formatting employed by the DMD test engineer. Regular expressions were adequate for filtering these data. A better solution might have been to employ the inferencing capabilities of CLIPS to provide more flexible data-item specification and/or location and extraction.

The CCL specifies all data channel definitions and processing in a FORTRAN-like programming language. The optimum way to extract relevant information from this file is to use a parser. The author decided to avoid building a parser, as part of this initial effort, by focusing on non-derived data channels for which regular expressions could be used to extract the relevant information for analysis purposes. But parsing is a well understood type of knowledge and should be implementable in CLIPS. It is not clear (to the author) that a parser

could have been developed faster with CLIPS than using the UNIX utilities lex and yacc. A parser implemented with CLIPS would probably be easier to maintain and would avoid having to use yet another language dialect to implement analysis related processing.

**Knowledge Bases**

The CLIPS knowledge bases contain the TE's knowledge about the various ways to compute EU from a knowledge of DN and how to determine the different types of alarm states. With this knowledge and supporting facts about actual alarm limit values and tables to be used in the DN to EU computations, the CLIPS analysis functions examine output from DMD, channel by channel, to determine what DMD thinks the EU and alarm states are and if the TE would agree that DMD produced the right answer.

This latter decision was implemented in a straight-forward mathematical way. CLIPS computed values are assumed to be correct. DMD and CLIPS computed values must agree to a certain level of precision in order to be acceptable. This rule takes into account differences in precision between printouts and the CLIPS internal representation of numbers. The great majority of channels tested were determined to be correct, but this precision criteria was good enough to catch the anomalies described below. If such an anomaly is found, a highly visible flag is set in the white space of the printout to catch the TE's attention should he decide to browse the output.

A post-process AWK function extracts those cases that have been flagged as described above and presents not only the analyzed output but also the corresponding information from the original CPT, LAD, and COEF input files. Although simple and unorthodox in implementation, this feature is a kind of explanation facility common to most expert systems. The TE has, in one convenient place, without having to flip through many pages of output, all of the supporting information as to why the test analysis tool thinks there is a problem. Now the TE can be the final authority, as to whether the anomaly is real and decide what action might be required.

**Anomalies Detected**

In exercising the above tools, some anomalies (unexpected behaviors) were detected. Two channels had both upper and lower alarm limit thresholds set to zero and their DN values were also zero. DMD and the analysis tool gave different alarm state answers for these channels. This reflects differences in the way the CLIPS analysis tool and the DMD handle pathological cases. Alarm limits were not yet established for some channels in the CPT database.

Another example is that a certain digital bit channel had an incorrect timetag and a wrong bit value. Using the merged database tool, it was discovered that this channel was not defined in the CCL file, but a value from some old data was present and displayed. At present it is not known if this behavior is a bug since the problem can be corrected by defining the channel in the CCL database. Thus, the DMD analysis tool caught certain problems which might have gone un-noticed.

## CONCLUSIONS

We believe the methods described above can be extended to a wide range of processors. If this proves to be the case for all, or even a strategic subset of processors within a system, then we can make automatic test data analysis part of regression testing.

A simple tool to reformat the subsystem inputs and output into a merged display provided an unexpected level of productivity enhancement, compared to strictly manual verification. In retrospect, the reason is clear. Merging and clustering of information saved the test engineer's time in cross-referencing four separate and voluminous printouts. Hence, significant test results analysis quality and productivity can be achieved without full automation. In addition, this tool proved to be useful during the problem definition and knowledge acquisition phase of program development.

In creating the analysis tool, we did not attempt to explicitly duplicate functionality of the software under test. Instead, we tried to emulate operations the test engineer performs when he verifies the software. That is, information collection and verification that the various computations were done correctly.

The current analysis tool could have been implemented without an expert system shell. This was not known at the outset. However, analysis capability for derived channels has not yet been achieved, due to the complexity of language parsing. Parsing is a form of expertise and a CLIPS-based inferencing approach may prove to be a simpler solution to this problem than, for example, a "C" language lex/yacc solution.

Mechanisms exist for inclusion of expertise, such as parsing, but the complexity of incorporating knowledge of this type is unknown. The scope of the present effort did not allow for a reasonable evaluation of this. Encoding test analysis knowledge in the form of rules was easier to review and maintain compared to an equivalent "C" program.

We have tried to strike a balance between production system programming and conventional programming to achieve a useful tool for the automation of test results analysis for a particular subsystem. We used CLIPS to encapsulate the analysis particulars, AWK and sort for data clustering, sorting and merging and UNIX Cshell for integration. The boundary regarding which processing tasks are best done within CLIPS and those best performed by an external function is changing, because CLIPS is changing and becoming more powerful with each release.

This work demonstrated the general usefulness of CLIPS and the ease with which expert systems or other types of production system applications can be integrated into a traditional computing environment. CLIPS applications can be combined with UNIX utilities to perform processing tasks.

## ACKNOWLEDGEMENTS

The author gratefully acknowledges the following individuals for their valuable contributions to this effort: D. Klemp, R. Wells, T. Kratz, P. Harmon, H. Avant and D. Hermsen.

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

## REFERENCES

Aho, A., Kernighan, B. and Weinberger, P. (1988). *The AWK Programming Language*, Addison-Wesley, Reading.

Brownston, L., Farrell, R., Kant, E. and Martin, N. (1986). *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*, Addison-Wesley, 1986.

Giarratano, J. (1989). *CLIPS Users Guide, Version 4.3 of CLIPS*, Artificial Intelligence Section, Lyndon B. Johnson Space Center.

Sobell, M. (1985). *A Practical Guide to UNIX System V*, The Benjamin-Cummings Publishing Co.