

REPRINT

Title: Commercial-Off-The-Shelf (COTS) Software Systems for Data Acquisition and Analysis ... Love Affairs and Land Mines

Author: Strether Smith, Lockheed Martin Advanced Technology Center, Palo Alto, CA

Source: *Sound and Vibration*

Date: April, 1996

About three years ago, our organization faced a critical fork in the technical road. We had been building large, expensive data acquisition and analysis systems primarily for structural dynamic testing (Ref: *Sound & Vibration*, Nov 87 and Jan 89). To our dismay, there were two essentially-simultaneous catastrophic events: the "large system" market dried up and our choice of host computer, that we had put all of our faith in, became non-viable.

We had tens of thousands of lines of custom code, optimized for our host of choice, and a collection of customers wondering how we were going to support them. We needed tools for continued development of machines that would support a wide variety of applications and data speed/volume requirements.

We selected the "PC-clone" as our new host of choice. We felt that, despite serious shortcomings, it was the logical choice for low- and medium-sized systems. They could also be used for high-end applications if they were used as front ends for data acquisition systems based on VXI or other higher-performance architectures.

On the software side, there were two choices: port our old "legacy" code to the new host with modernizations or climb on the Commercial Off The Shelf (COTS) software bandwagon. The arguments for porting the old code were familiarity, both on our and our customer's part, and rational use of our huge investment. Arguments for using COTS programs are legion: Lower cost, better maintainability, more flexibility, the facility to blame someone else for problems,...and the fact that the concept is very much in style. However, we could not find anyone who had used these tools to build large, complex, user-friendly (error-protected) systems.

Before we go on, we need to clarify the subject: We are not talking about turnkey software programs that perform specific tasks such as the shaker-control systems that you might get from LMS, Spectral Dynamics, or M+P. The software under discussion falls between the turnkey systems and the "conventional" software-development systems like FORTRAN and C. They are specialized for specific tasks, such as data acquisition and time-series analysis, and are meant for the development of custom applications.

After a few abortive attempts at porting, the COTS option was selected. The following is a description of the trials and tribulations associated with that choice, "compiled" after two years of system development (that has produced fourteen custom systems of various sizes in the process).

The first thing that we discovered was that there are well over 200 data acquisition/analysis programs available and, to make it more confusing, many of them are very good. Marking the "bingo" cards in our favorite magazines quickly buried us in glossy literature and demo disks. We were presented with beautiful plots, pretty push buttons, and the promise that we would be able to solve any data acquisition or analysis problem. But the literature and demo disks gave us very little feel for how the program worked (look and feel) or how powerful (or perverse) it was. The demo disks were often severely crippled and all came with such poor instructions that it was impossible to evaluate the package properly. It appeared that the vendor's attitude was that their system was so straightforward that we should not need any guidance. Well, we did! More important, we found that most of the demonstration software was only adequate for a very cursory examination.

We set up the following criteria for the selection of the candidates (listed in order of importance):

- The program had to run on a variety of host computers and software needed to be easily transportable. (We were not going to get caught in the bind we were just leaving).
- The program had to include a large signal-analysis toolbox.
- Mathematical operations, such as FFT and IIR/FIR filtering had to be fast and easily implemented.
- Object-oriented/reusable code was required.
- The system had to be oriented/adaptable to small and large systems.
- The program had to have a modern "look-and-feel."
- Special features (ease of operation, interface naturalness, operator taste.....) were included as a "fuzzy," but critical, criteria.

Smith, "COTS Systems for Data Acquisition and Analysis," *Sound and Vibration*, April, 1996, Page 2 of 4.

- The program had to be well-supported:
 - We required a support/development/support staff of at least 20 "techies".
 - A dedicated "Technical Support" facility with unlimited (paid-subscription) phone/E-mail support was required.

With this list in mind, we culled through the forest of options by reviewing the vendor's literature. Eight candidates were selected for demo-disk evaluation. We then spent a few hours evaluating these as well as possible from the demo disks and reduced the candidates to four.

We then called the software vendors to "borrow" a full copy of the software with documentation. When it arrived, we played with the provided examples (they normally, but not always, worked). When we were fairly familiar with the program, we built several applications that were typical of the kinds of operations that we expected to do. One of the packages that we evaluated performed the demo examples flawlessly but invariably crashed when we got very far from the "beaten" path. We did timing and accuracy studies on Fourier transforms up to 128K points (both "power of 2" and not) and other mathematical functions that we expected to use heavily.

We called the vendor's technical support line when we got in trouble. The speed and effectiveness of their response was evaluated and thrown into the hopper.

We found that two, inherently-different, types of data acquisition/analysis processors are available.

The first type, which I will refer to as "command-line-driven" programs, are developments/extensions/enhancements of processors that have been available for years. MATLAB, DADiSP, and Mathematica are prime examples of this genre. These are powerful programs that have been developed over the years in response to the demands of the mathematics and signal-analysis communities. They are controlled by interactive entry of command phrases or by "scripts" made up of command phrases and program-flow commands. The programs have recently undergone major facelifts to give them a modern look and feel. They are primarily suited to post-test data reduction, although several of them are "growing" a real-time acquisition/display capability.

The second type, which I will refer to as "graphical-programming" systems, include programs such as LabView, VEE and Visual Designer that were developed by the data-acquisition community. They all share the "modern" concept of object-oriented, drag-and-drop/wire-connect programming. These are all directed toward real-time-display

operations and promise to provide excellent test-operation and display capabilities. Because they are used for real-time operations, compute-speed is critical and relatively-large, fast, machines are required, even for minor tasks. Since the approach requires a program/compile cycle to make any analysis modifications, trial-and-error analysis-procedure development is tedious. Despite the fact that their on-screen plotting capability is spectacular, they have relatively-poor hard-copy and report-generation facilities and some other program (such as Excel or one of the "command-line-driven" programs) is often required to produce report-ready plots.

As the programs evolve, the capabilities of the two families are looking more and more alike. But, for the moment, the two program types offer significantly different capabilities.

So your first task is to decide which program type (or both) is best for your application. We concluded that a program of each type was required to satisfy all of our data acquisition and analysis requirements.

A review of the capabilities and shortcomings of the selected programs is far beyond the scope of this editorial. However, there are several general observations that can be made after two years of using the programs and their associated strategies.

The first surprise, which in retrospect should have been expected, was how long it took to become efficient and knowledgeable programmers in each of the two languages.

A C/FORTRAN programmer who is experienced in data-acquisition and signal-analysis processing was assigned to "learn" and implement systems using the graphical-programming language. His experience produced the following conclusions:

- It took him about 4 person months to become proficient.
 - Once he learned the necessary skills, he could develop application programs about three times as fast as he could with C or Fortran.
 - Once the "guru" was developed, additional programmers were bought along very quickly.
- The graphical-programming environment is only different, not inherently better.
 - The paradigm has real advantages for simple tasks but as the program gets more complex the advantages diminish.
 - Since the program has no interactive capability, "cut-and-try" algorithm development is tedious and inefficient.

Smith, "COTS Systems for Data Acquisition and Analysis," *Sound and Vibration*, April, 1996, Page 3 of 4.

- The environment does not remove the need for constant consideration/handling of memory management, global variables, or handling of many of the other complexities of conventional programming.
- The "reusable-code" aspects of the concept did not work well. Writing and maintaining reusable modules proved to be very difficult (more trouble than it was worth!). However, this is largely offset by the ease of constructing special modules custom-built for the task.
- Once the programmer/user is familiar with the programming paradigm, the code is easily "read" and understood. However, rational documentation of large application programs is very difficult. The concept requires a multi-dimensional road map that is very difficult to express on a flat piece of paper.
- An excellent collection of demonstration examples was provided. These could often be modified to suit a required (simple) task.
- The program is an enormous memory hog and we soon stopped attempting to cram any but the most trivial programs into machines with less than 32 megabytes.
- The printing and graphic-output facilities were very primitive. Excel, or the "command-line" program is normally used to produce report-quality graphs.

Our experiences with the "command-line" system have been similar but, of course, different.

- Learning the program was easier because its command-line/script paradigm was relatively familiar.
- The command-line interface is ideal for interactive, cut-and-try, algorithm development.
- The "script" method of programming is straightforward and amenable to documentation with in-line comments.
- Most of the "computer parameters" are well hidden and transparent to the operator.
 - It is meant to be a "scientific" program, isolating the user from the computer details.
- The program that we chose has very serious robustness problems. These surfaced regularly in the program development tools. Often the toolbox functions did not function as documented (or at all). The developer is conscientiously pursuing these errors but, because the system is largely built of legacy modules that must be made compatible with the "new face," it is an enormous task.

- Once the programs (scripts) were developed and debugged, they did operate consistently and correctly.
- The program has relatively-good hard-copy and report-generation capability.

Using either program can be an exciting experience. Reaction can change from "Wow, that is really neat," to "That is unbelievably perverse," to "I have to develop a work-around," to "I will have to use something else," in a 30-minute period.

Both programs had serious shortcomings:

The most critical is the "*land-mine*" problem. Once the developer strays far from the well-beaten track, the chances of stepping on an injurious, or even fatal, "feature" increase significantly. Also, the chance of tripping on a "memory leak" increases dramatically with the size of the application program. So far, we have been able to work around (kluge) all of the (many) explosive devices in the "command-line" system. Although fewer work-arounds have been needed, the "graphical program" had one fatal flaw that caused us to abandon it for one application.

In both systems, the more advanced (and less used) elements of the toolbox are complex and poorly documented. Significant trial and error is often needed.

In principle, this problem should be solvable by good technical support. We found that the two vendors had significantly different capabilities and responsiveness. One vendor, a relatively-large company, provided excellent responsiveness to easy ("beginner's") questions but it was often difficult to track down expertise on the hard ones. In some cases, the information took more than a week to materialize and considerable "mining" on our part was required. On the other hand, the smaller vendor assigned a single, knowledgeable, person to respond to our problems. In several instances, that representative wrote custom code to provide work-arounds for problems encountered. Their responses to the hard questions were much more timely.

In both programs, handling of logarithmic plot scaling was primitive. Shortcomings in the "command-line" system were partially overcome with a special command script (not provided with the system). There were no tools in the "graphic program" to handle this plot format efficiently.

One of the most-significant reputed advantages of COTS software is transportability. However, we found that moving developed programs from one machine to another produces formatting problems if the screen resolutions are different. This requires the development of "custom" versions for each machine. Our few attempts at porting software to different hosts were less than perfect. Also, op-

Smith, "COTS Systems for Data Acquisition and Analysis," *Sound and Vibration*, April, 1996, Page 4 of 4.

erational difficulties were experienced with both programs on some hardware (PC) platforms. Data acquisition operations often did not transport well, even to nominally more-capable machines, reflecting the fact that all PCs are not created equal (a subject that requires some real attention).

A primary consideration is the amount of code that must be generated to perform a particular task. We have implemented the Lockheed Martin SHOCKANA shock-response-spectrum (SRS) program, originally written in FORTRAN, with both of the COTS packages. The statistics are:

FORTRAN

2918 lines of code. (Not including an extensive toolbox of text and graphic functions that is roughly the equivalent of the COTS systems' tool boxes).

"Command-line"

480 lines of code (about 20% is work-arounds)

"Graphical-programming"

307 functions/icons connected by 511 wires

It is obvious that either of the COTS approaches has significant advantages in "bulk of code" over the conventional-programming approach. Code development is, in fact, much faster using these tools.

The features and shortcomings of the systems produced a significant change in programming style. In both systems, major effort must be put into the development of work-arounds that circumvent land mines and missing capabilities. As the applications get larger these "features" are encountered more often and kluges often become a major part of the code.

The bottom line is that there are huge advantages of using the COTS approach but that entry into the new world should not be taken lightly. The advantages are not as clear as advertised. However, as COTS software improves, the tradeoff will certainly move in that direction. One thing that we can do as users is to put pressure on the COTS software system developers to concentrate on robustness concerns before they add scientific enhancements. Any other approach seems shortsighted.

For the moment, the water's warm, but it is deep in spots, and there are snapping turtles lurking in the holes.