
Software Reuse in Wind Tunnel Control Systems

by Charles E. Niles

Software is an important element of wind tunnel operations at NASA Langley Research Center (LaRC). Reuse of wind tunnel automation software, while limited, has produced benefits on a local scale. Two forms of reuse have been utilized—from project to project, and from system to system within a project.

LaRC possesses a broad range of wind tunnels, test facilities and laboratories which support our nation's aeronautics and space endeavors. The wind tunnels enable researchers to study aerodynamics, fluid dynamics, acoustics, heat transfer and other similar interests in order to evaluate and improve the performance of aircraft, missiles, jet engines, spacecraft and various components thereof.

Although wind tunnels vary in purpose, size, shape and operating range, similar software has been used across facilities for data collection systems, data reduction systems and control systems.

To simplify discussions, only closed-circuit wind tunnels will be addressed. Essentially, a wind tunnel is a continuous, large-diameter cylinder arranged in an elongated, oval circuit. Air or some other medium is moved at speeds up to Mach 1.3 around the circuit by large fan blades. Two-thirds of the distance around the circuit from the fan blades is a test section in which a model is manipulated at various pitch and roll angles to gather data on aerodynamic effects. The model may be fitted with scale-size jet engines which are driven through independent high pressure air systems. Pressures inside the tunnel range from one to several atmospheres. Temperatures range from near absolute zero to over 150 degrees Fahrenheit.

Wind tunnel control systems actuate subsystems to affect fan speed, pressure, temperature, pitch, roll and other tunnel, model or test article processes. The dynamic interaction among tunnel processes causes

control system software to be moderately complex and uniquely adapted and tuned for a facility. Personnel safety and facility/model protection justifies very reliable software which further adds to complexity.

Automation System Projects

Between early 1985 and late 1989, the 16-Foot Transonic received a major overhaul in which almost every operational system was affected. The staff assembled for this effort encompassed every imaginable engineering discipline—civil, electrical, structural, mechanical, controls, and software, to name a few.

During the overhaul, the control room was modernized and state-of-the-art microcontroller equipment was installed to provide automated controls for tunnel (TNL), model attitude (MDL), and high pressure air (HPA) systems which were interconnected with a standby (STB) system via a local ethernet network. The STB system served as a ready standby for any one of the other three microcontroller systems. The STB shared its chassis with a gateway (COM) which communicated with a process monitor and control (PMC) minicomputer via a custom parallel link. The PMC also communicated with the facility data acquisition system via a separate network.

The effort involved the development of over 150,000 lines of code. The TNL, MDL, HPA, and STB microcontroller code was written in PLM, FORTRAN, and assembly language for Intel 8086 CPUs running the RMX-86 operating system on the Multibus I architecture. The PMC code was written entirely in FORTRAN-77 for a Modcomp running the MAX IV operating system.

Approximately 75% of the PLM source code, known as the environment code, is identical on the TNL, MDL, and HPA systems. The FORTRAN code,

which represents the control algorithms, is necessarily unique. Overall, PLM (85%) and FORTRAN (15%) make up the bulk of the code. Assembly language (less than 1%) was used only when necessary. The STB system is a composite of the environment code and the specific portions of the TNL, MDL, and HPA code.

The TNL, MDL, HPA, and STB systems consist of about 30,000 source lines each, while the COM and PMC combine for about 30,000 lines. During the project, the environment code was developed using the TNL system as the basis. In effect, the MDL, HPA, and STB systems were instances of reuse within the project. In addition, that portion of the COM software which supports the local network among the systems is identical. The COM software which supports the parallel link and the PMC have not been reused.

A year after the 16-Foot project, two control system upgrade projects were performed. The first project, at the Jet Exit Test Facility, involved cloning the HPA system, making some facility-specific changes, and enhancing the environment code. This was a small project, requiring one software developer. In an extremely unusual scenario, the computer hardware arrived, the control system software was installed and checked out statically. Several months later, the rest of the project caught up while the software developer was at graduate school. The author, a veteran of the 16-Foot project, was pressed into service to support checkout of the integrated system. As a testament to the stability of the software, checkout went smoothly.

The second project at the National Transonic Facility (NTF) was more complex. This project included newer hardware (80486 Multibus II vs. 8086 Multibus I), a newer operating system (RMX-III vs RMX-86), a newer language (PLM-386 vs PLM), and integration of the existing control algorithms. Each element offered a different challenge. The first three were straightforward—new hardware required new drivers, a new operating system required new or modified system calls, and a new language was almost transparent. But the existing control algo-

rithms had to be repackaged to conform to the environment-algorithm interface.

In the transition from 16-Foot to NTF, some parts of the environment code were optimized. However, adding generalized code which was formerly handled uniquely by the microcontrollers caused the overall size of the environment code to increase slightly. At the end of the NTF project, 90% of the original 16-Foot microcontroller software had been reused with little or no modification.

In all, these three projects account for eight systems which consist of the same environment code with incremental improvements and optimization over time.

Issues

The obvious benefits of reusing the environment code include shorter product delivery time, minimal time invested in documentation after the initial facility, and easier maintenance. Beyond the obvious benefits, the environment code has provided a firm foundation to which new control algorithms have been and are being added.

It is a pity that the code has not been reused more. Unfortunately, the large initial investment in the 16-Foot project took its toll. When the project was completed two years late, management took a dim view of performing a software intensive project using in-house personnel. Thus, only selected projects were subsequently tackled. Of course, there were other reasons—more projects than in-house staff could perform, urgency in obligating funding (a form of sheer madness), the emergence of commercial applications, and a rapidly changing hardware climate.

There have been several significant impediments to reuse. Perhaps the biggest has been individuality. Most of NASA's major accomplishments are attributable to individuals. NASA is full of free-thinking scientists and engineers...and software developers who are probably the most individualistic of all. Of course, management traditionally has fostered an environment of creativity and designer-preference.

So, there has been little discipline to reuse anything, much less software, except on an individual basis.

Another impediment is organizational structure. LaRC lags far behind the NASA space centers where software has been a critical element of most everything ever launched. Although there are pockets of individual software expertise scattered across LaRC, there is no formal organization. Software reuse will flourish more in a software engineering organization than not. In the author's engineering organization, mechanical, structural, and civil disciplines are prevalent. There are engineers who develop software, but no software engineers. Such an environment is simply not conducive to software engineering. Without software engineering, good software design occurs by accident. Usually, inferior design results in inferior source code which should not be reused.

Next Generation

Three events since mid-1994 have changed the long-term vision of automation projects within the author's facility automation software development staff. First, the staff underwent a capability self-assessment which was facilitated by a cross-center team of experienced software personnel and the Navy's software engineering group at Damn Neck, Virginia. The results brought management attention to issues which affected the broader organization. In short, the staff should focus more on facility automation and less on software development.

Second, the author, long an advocate of a standard approach to automation systems at LaRC wind tunnels, accepted an offer for a team from the IV&V Center in West Virginia to conduct a domain engineering effort of wind tunnel control systems. The team, known as the Software Optimization and Reuse Team (SORT), methodically analyzed several wind tunnel control systems and developed an essen-

tial set of requirements. More recently, SORT has developed a set of derived requirements during a domain design. The SORT effort also influenced the third event.

Third, the author and a systems engineer who is also a standard product advocate have embraced a new approach with the full support of management who want to reduce development costs and time in the face of budget cuts and loss of personnel.

The rapidly changing hardware climate has also been a factor. Since Intel decided to drop support for its Multibus II architecture (along with the RMX-III operating system), the need for a different hardware platform became evident. The VXI bus architecture has been chosen in order to accommodate the instrumentation needs of both control and data acquisition systems. The Lynx Operating System (LynxOS) has been chosen to replace RMX-III. Together, they represent the standard hardware/operating system platform of the next generation.

The next generation also involves a widely used software package known as EPICS (Experimental Physics and Industrial Control System). EPICS, which originated within the Department of Energy, was developed by computer scientists and physicists for application to electron beam accelerator facilities. Over the years, EPICS has been adapted to other applications including physics labs, astronomy labs, and jet engine test facilities. Although EPICS is making its first appearance at LaRC, it is already installed at over 75 sites worldwide. Oddly enough, EPICS is used by NASA at the Canberra tracking station.

The author's organization believes EPICS will transcend software reuse. Combined with a standard hardware platform, EPICS means that **systems** can be reused.