

# Reducing the Cost of Test Through Reuse

Douglas D. Lonngren  
Vice President of Engineering  
Serendipity Systems, Inc.

P.O. Box 10477, Sedona, AZ 86339

Phone: (928) 282-6831 Fax: (928) 282-4383 E-Mail: dlonngren@serendipsys.com

**Abstract:** *As the length of time between design and production decreases, the demand for timely and cost effective testing increases. One way to reduce the cost of test development is through the reuse of applications and components. This approach includes designing and implementing reusable components as well as leveraging off of existing tools and utilities.*

*This paper describes a set of test applications that were designed for reuse and that, in turn, facilitate the creation of reusable components. These applications provide a common set of runtime services to a test system. In addition, their open architecture supports a variety of test development environments and integration with third party or custom utilities. Descriptions of the applications are augmented by design discussions and implementation details.*

## I. INTRODUCTION

Reuse is the mantra of test and software development. Write once, use many times. Don't reinvent the wheel! Think twice, write once. Work smarter, not harder. Use objects...

Oftentimes these edicts are accompanied by lots of chanting and little direction or true commitment. The predictable outcome is that the resulting designs are about as reusable as staples (you can do it, but it feels like an unnatural act).

At the other extreme, reuse is embraced by the enlightened and they retire to their ivory tower to contemplate the perfection of a unified solution. They while away the hours and weeks debating the finer details of their theoretical approach. Then, if they ever finish, the result is complete, complex and impractical. In addition, the people interested in the results have long since moved on to something else or implemented their own, "temporary", solution.

If Henry Ford had taken this latter approach, his committee would still be debating whether a sports utility vehicle is a subclass of truck or station wagon (and we would all be knee-deep in horse manure).

This paper describes a more practical method for creating reusable applications and components. It details the design and implementation of a reusable test application that, in turn, promotes the creation of reusable components. Examples from the development process, and resulting application, illustrate various concepts.

## II. BACKGROUND

In 1994 we were approached by a large defense contractor to develop a runtime system to coordinate their internal test processes. They were interested in creating a standard set of runtime services that would meet the various needs of groups within their organization. Their goal was to eliminate the duplicate development efforts that each project incurred when creating custom runtime systems. It was also planned that a common approach would facilitate data collection, standardize result reporting, increase productivity and decrease training and support costs. The design was to also promote the creation of reusable test components.

We were selected because we had already developed several similar systems. Thus we had valuable experience and a unique perspective to apply to the problem. In addition, an outside entity was less likely to become mired down with internal politics or biases. This gave us the distinct advantage of focusing entirely on developing the reusable applications and runtime services. This also allowed us to consider the test requirements of multiple groups rather than just one.

Many companies make the mistake of expecting robust, reusable applications to appear magically as a byproduct of normal test set development. They are then surprised when massive changes are required in order for the next project to use the same application. Unfortunately, when the focus is on an end result, there is rarely time to consider elements that do not apply to the current situation.

### III. PROJECT STRUCTURE

The way that the project was structured had much to do with its success in creating a widely deployed, reusable system. Our client produced a list of requirements compiled from interviews with numerous groups. From this we created a schedule of phased releases with specific capabilities outlined for each. By building on a core set of requirements, each release was functional enough for actual use. This resulted in a rich source of feedback about the system's operation. We were thus able to correct problems and add capability throughout the development process.

Our client designated an individual as the focal point for our questions and as a filter for their internal responses. This ensured that they maintained a consistent internal viewpoint and that someone was empowered to make design decisions and tradeoffs. Groups within the client's organization were encouraged, but not required, to use the system that we were developing. Therefore, to maximize deployment, we needed to win the support and confidence of each of these groups. It was far better for them to accept the system based on its own merits, rather than forcing them to use it.

During development we found that the periodic feedback was essential for adjusting the application's operation. If terminology or a behavior caused confusion, we could correct it at the next release. We also discovered requirements and wishes that were

missing from the original specifications. For the most part, these could be incorporated into the capabilities of a future release. It was often the small requests that generated the most positive feedback when we implemented them. Some of them were as simple as an extra menu item, shortcut key or modified icon.

Because of the multiple releases, it was very important to maintain an accurate list of what each version did relative to previous versions. This was tracked in a Read-Me file that was delivered with each release. The list included bug fixes, new capabilities and any changes in behavior that the user might experience. Not only did this keep the end users updated, they could also see the enhancements that resulted from their suggestions or problems. This closed the feedback loop and was essential for maintaining a healthy developer-user relationship.

### IV. ARCHITECTURE

The architecture of the specified runtime system is shown in Figure 1. The test executive retrieves a test definition from a file and coordinates test execution and data logging. Testing is performed by executable test objects and the results are passed to the test executive via the texDLL interface.

The environmental test manager (ETM) coordinates unit testing with the operation of an environmental chamber. It executes an environmental profile, logs results to a database and dispatches pages based on system conditions. The ETM controls the test

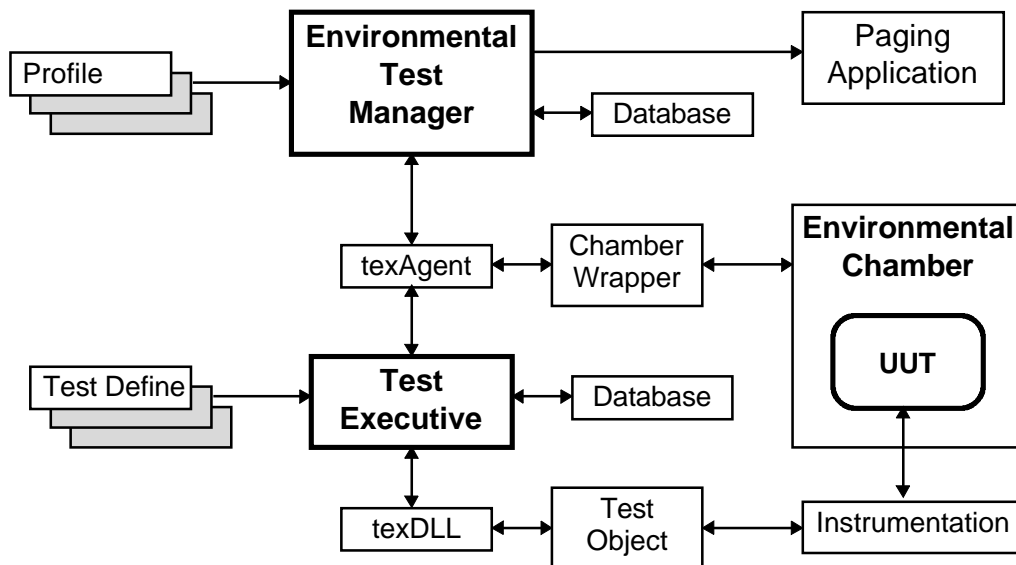


Figure 1: System Architecture

executive and chamber wrapper through the texAgent interface. The chamber wrapper directs the operation of the environmental chamber.

Our initial agreement was to develop the test executive portion of this system. A remote interface was included in the design as a potential link to an environmental test system. Eventually we were approached to develop an ETM that could manage multiple environmental chambers and UUTs. The following sections describe the design and implementation of the various system elements. Special attention is paid to the choices made for maximizing reuse.

### V. USER INTERFACE

Initial reaction to a new application is quickly formed based on its user interface. Consequently, it was important to incorporate the desired capability in a pleasing and functional arrangement. We needed to allow the users to explore the interface without negative consequences. In addition, they should be able to achieve a desired result without mastering the whole application. Incremental success allows a user to slowly expand her knowledge of, and confidence in, a new software package.

When designing for reuse, do not underestimate the value of small niceties such as a file history, toolbars, tooltips, status bars, resizable windows, help files and restoring window positions. These all add to the usability of an application and to its feel of completeness. They also add to the development cost and tend to be left off if an application has a perceived onetime use. The extra effort pays back over multiple uses or with a large installed user base.

For the test executive, an initial positive experience was achieved through a test hierarchy display that includes status icons and color, visual feedback and easy operation (see figure 2). As simple as it may seem, the color selection and icon design went through several iterations until they were acceptable to all users. A status readout and several push buttons complete the minimum set of controls necessary to run a test.

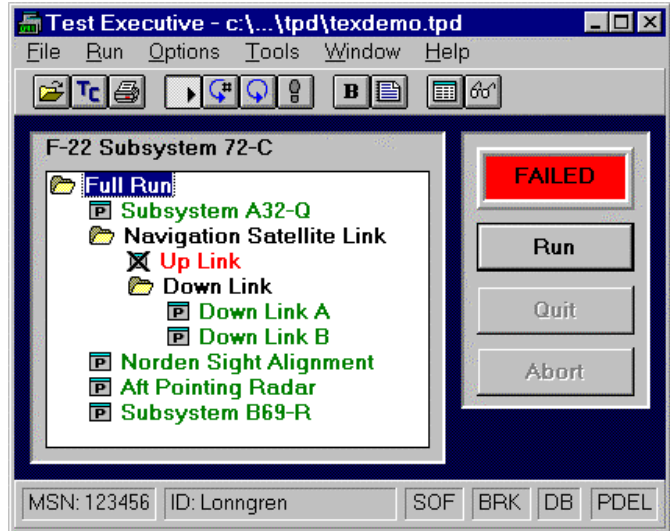


Figure 2: Test Executive

Once the basic operation is mastered, additional information and control are available through separate dialogs, windows and external applications. These are easily accessed from menus, function keys and the toolbar. These capabilities include viewing test results, controlling data logging, entering test data and accessing help. Test history is viewed and printed with an external report generator application. For advanced users, a configurable tool menu provides access to custom or third party utilities.

A variety of debugging and run options complete the capabilities of the test executive user interface. The run modes include looping, single stepping and stop-on-fail. Breakpoints provide visibility and pacing to a test's execution. Multiple breakpoint levels adapt operation to different debug scenarios (see figure 3).

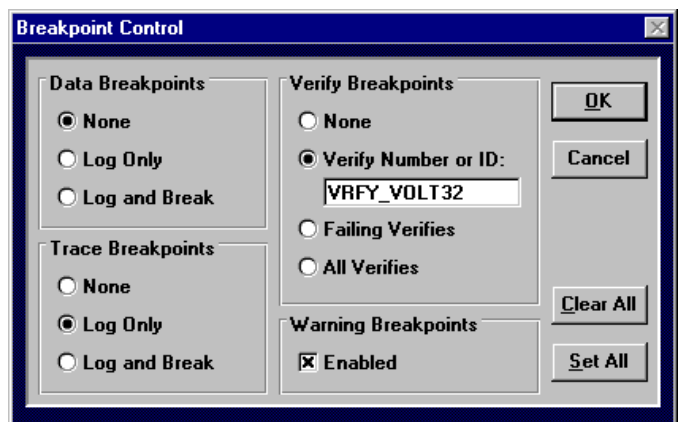


Figure 3: Breakpoint Control

## VI. SYSTEM CONFIGURATION

For the greatest reuse, a complex application must be configurable. This allows it to adapt to a large range of users and requirements. It also solves the common problem of conflicting requirements where one group "has" to have a particular option and another group will not accept a system with that option. Configuration is necessary for managing enhancements and continuing development. When a significant change in operation is implemented, there should also be a way to disable it. Thus, as carefully selected enhancement requests are honored, there is a fallback position for existing users and established processes. Of course, discretion is necessary when adding configurable options in order to avoid Floor-Wax-And-Dessert-Topping Syndrome.

How configuration is handled is always an important consideration. A simple approach is best, but it also depends on the skill level of the targeted users. Since our primary users are test developers, we implemented a test definition file. This has a simple ASCII syntax that is easily edited with a variety of text editors. Syntax checking is performed when the file is loaded by the test executive. A typical test definition file is about two pages long, so it is not an undue burden on a test developer. A file of this size and format is easily added to documentation packages and tracked with a version control system.

The further advantage of this simple approach is that syntax and file processing is very easy to change and extend. This has allowed us to provide incremental enhancements in a timely manner. An interactive test definition editor was considered, but its upgrade, maintenance and documentation would have slowed the test executive enhancements that we were able to deliver.

The test definition file is the key to the behavior of the test executive. It can be very simple or very complex, depending on the needs of the test set. The test definition file controls the test hierarchy, data logging, data entry syntax, interface options and access to external utilities. It also optionally supports version control on the components, files and programs required for a complete test set. This has proven to be a versatile solution for meeting the different project requirements of various groups.

## VII. DATA LOGGING

Generally, every application has links to other processes, whether they are manual or automatic. Most often these links are used to exchange data between applications. The more links an application

supports, the greater is the opportunity for it to consume or generate reusable data.

To best support post-processing test data, two types of data logging are implemented in the test executive. One type is a Microsoft Access database and the other is an ASCII text file. Microsoft Access is a common commercial database with numerous tools for reading, processing and reporting. One consumer of the Microsoft Access database is a report generator application that displays and prints test results.

The ASCII text file is written in a Parametric Data Exchange Language (PDEL) syntax. This matches our client's internal process that automatically gathers test data into a large reporting system. The ASCII files are also available for other post-processing applications for printing, storing or analysis.

## VIII. TEST DEVELOPMENT

One of the major factors for acceptance of a reusable approach in test is the attitude of the test developers. If they perceive that this "new" idea creates more work for them, they may not readily accept it. There may also be resistance if it deviates radically from their existing viewpoint, or if they think that they should develop it themselves. Consequently, it is important to implement concepts that the end users are familiar with and to provide compelling improvements to their current tasks.

The test executive achieved this by modeling the testing process on existing practices. This involved designing a set of verify functions for performing measurement comparisons. Developer written test objects access these functions from the texDLL. To add value beyond a verify operation, a rich set of diagnostic capability was added to the function set. This includes breakpoints, trace points, automatic data logging and help file links. Now a single verify function returns a result, logs the comparison to a database and PDEL file, optionally causes a breakpoint and links to a context-sensitive help file. That capability, packed into one function call, is powerfully persuasive. A further benefit is that it becomes standard behavior for all test objects.

The runtime function set is accessible from a variety of programming languages and development environments. This ensures that the test developers can continue to use tools with which they are comfortable. The 18 functions in the set do not require complicated pointers, structures or object knowledge. Also included in the function set are simple user interface operations, including message boxes, error boxes and data entry dialogs.

Test objects are executable programs that perform the actual testing on a Unit Under Test (UUT). They are also reusable components because they can be designed with all of their measurement limits stored in an external file. This allows them to be reused, with different limits, for manufacturing, acceptance and environmental testing.

Other reusable components supported by the test executive are action objects or tools. These are custom or third party utilities that are added to the test executive's tool menu. They can be instrument soft panels, diagnostic applications or fault isolation tools. Once created or acquired, they can be incorporated into a variety of test sets.

### IX. ENVIRONMENTAL TEST

The environmental test manager is the most recently completed element of the runtime system. It manages multiple environmental chambers while using the test executive to handle test execution and reporting. Many of the concepts applied during the test executive development were reused when creating this application.

The ETM user interface provides visual feedback about chamber, UUT and profile execution status (see figure 4). A reserved use of color highlights the active

chambers and profile phase. A breakpoint mechanism pauses profile execution at operator defined locations.

The ETM is configured by an ASCII text file. This permits it to adapt to the needs of a particular test station. It can control up to eight environmental chambers and each chamber can manage up to 16 UUTs. Control over different environmental chambers is provided by custom chamber wrappers.

Environmental profiles are defined by a simple syntax in an ASCII file. Additional information for testing is located in an associated test definition file. This allows the ETM and test executive to share data format and test information from a single source.

Chamber status and UUT summary test results are stored in a Microsoft Access database. Detailed test results are logged by the test executive through its normal operation.

As with the test executive, several compelling attributes of the ETM make it attractive for adoption by end users. Its close link to the test executive maximizes on existing test sets, knowledge and skills. Automated phase selection algorithms adjust profile execution without requiring operator intervention. In addition, flexible control over system monitoring includes conditional pauses and operator notification via pager.

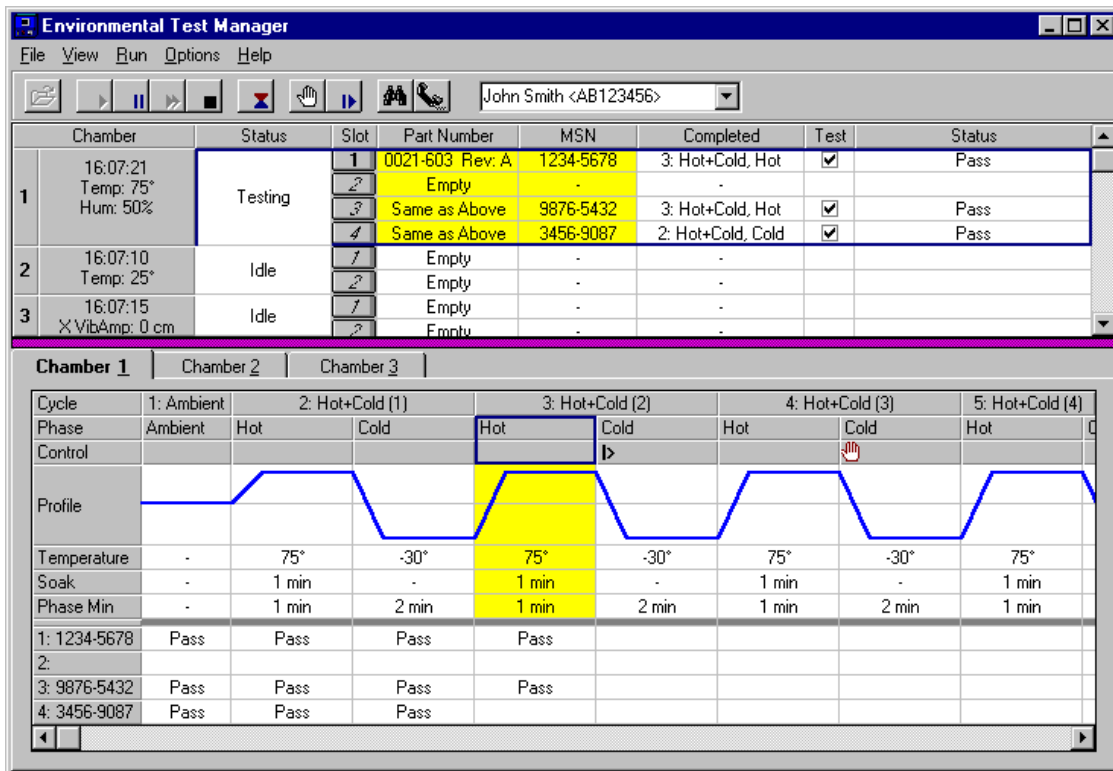


Figure 4: Environmental Test Manager

## X. CONCLUSION

Reuse of applications and components reduce the cost of test. The savings occur from reductions in development, integration, documentation, maintenance and training. Reuse occurs with internally developed applications that are reused on other projects and commercially available applications that are internally deployed for maximum benefit.

When developing for reuse, instigate phased releases to motivated users, incorporate feedback in the development process, design for expandability and keep users happy. Acceptance is as important to successful reuse as is the application's capability. If it seems to be hard to use, it won't be (except by force). Design a rich set of functionality with a minimal learning curve. Make it easy to incorporate into existing processes and procedures. Provide interfaces that encourage its integration into even more powerful systems.

Designing and developing for reuse is possible. At last count there were over 15 groups or projects using the test executive. What is required is a vision of what is necessary and a structured process for achieving it. Strong internal support for the approach is necessary, along with a compelling reason for adoption.