Modern Technologies Are Biting Off More Than Our Test Systems Can Chew

Neil Baliga
Verifide Technologies, Inc.
6050 Hellyer Ave, #150
San Jose, CA 95138
neil.baliga@verifide.com

***Abstract*— Despite billions spent on software in testing aerospace and high technology products, companies are still struggling to keep up with Moore's law and shorter schedules. This results in excessive software development, hard to analyze data, manual labor that slows the disposition cycle time, lower production rates, and costly errors in high-value DUT's.**

Modern technology has delivered a one-two punch for aerospace manufacturing organizations. The first blow is the increased complexity of devices; existing test systems are stretched beyond their capabilities and require major rework and iteration. The second fatal blow is the pressure on test organizations to release on extremely short schedules and budgets due to rising competition in the industry.

This paper presents two key concepts to mitigate the effects of the new technology revolution on your test systems - Modularity and Scalability. These are not new concepts; this paper will, however, discuss what these concepts mean for test systems, and also cover how to implement and combine the them to reduce recurring costs and technical risk in your test organizations.

I. IMPACTS OF MODERN TECHNOLOGIES

In the rise of the technology age, aerospace and high-tech companies were pioneers in the fields of testing. They had to wade into unchartered waters and deliver things never done before; they had to make sure things worked. Missions like Apollo had a lot more than money at stake.

In those days, the emphasis was on blazing new trails, not on improving yield or throughput. The technology was not easy, but it was less complex. Testing was done either manually or semi-automated and were often staffed with large teams of engineers with white shirts and black ties. As we progressed into the 80's and early 90's markets were developing and competition had started to develop in these areas. Companies needed to switch into production mode to accommodate the growth in the commercial and military sectors. Labor rates and technology iterations were relatively low which reduced the pressures for achieving higher levels of automation so lot of companies got by.

In the late 90's and early 2000's in the midst of the dot-com boom, device complexity and labor rates had ticked up significantly. Companies improved production throughput with solutions using both commercial off the shelf (COTS) and home-grown hardware and software. Most test systems were point-solutions; they did what they needed to do for testing a particular product.

In the last 8 years, technology has taken a significant leap in both consumer, commercial, and military applications. Device complexity has increased and so has competition. Older test systems that were developed and getting by are starting to crumble under new sets of requirements, schedule expectations, and modern cost structures. Often times the sales and procurement side of the business does not understand the impacts of the new business to manufacturing and the project is under-bid, putting more pressure on the test organizations.

Complicating this is the fact that cycle time for new technology generations has also accelerated, creating more frequent hardware and software iterations for test systems in production. Some test system iterations can take up to a year for a "tech refresh" and then needs to be re-evaluated in a couple of years for the next generation of products. The perpetual non-recurring costs are eating into the thinner margins manufacturers are already facing.

For manufacturing and test organizations, the agony of Moore's law is that there is no expected cost increase for the increase in complexity of the technology. Everything is expected to cost the same or much less, so economies of scale need to be squeezed from somewhere else. Test organizations are also typically further down the value chain and suffer the highest schedule pressures with depleted budgets.

Generally, there is agreement that things need to be simplified to reduce costs using Commercial Off The Shelf (COTS) and modular solutions. The rise in labor costs and the

often cyclical nature of test system iterations makes COTS a more attractive proposition for these companies.

On the hardware side of systems, there have been positive trends with modularity in the concepts of common backplanes like PXI and Software Defined Measurements (SDM) which abstract hardware from measurement firmware. This model allows system designers to fine tune and scale the system to a certain degree.

On the software side of test, scalable and modular software solutions are woefully lacking; developers build some degree of home-grown modularity and scalability in their system in an attempt to solve the problem. Though well intended, these solutions often fail at implementing the concepts properly. A main reason for this is that organizational and budget pressures steer test projects towards one-time point solutions, not towards a holistic solution that is more enduring. Some of this is caused by "the color of money" problem - Aerospace and high-technology companies are very stratified; they have overhead budgets, program funded projects, R&D allowances, and capital dollars that each have their tunnel view of the problem to solve. This also makes it hard to solve these problems internally so any and all code-reuse is highly welcomed.

When modularity is attempted but not implemented well, it only results in wasted costs because the overhead of designing and building system infrastructure is not cheap. In many cases, the project may be better off without having spent efforts on modularization at all.

This paper addresses a solution to these ongoing constraints on test organizations and developers through the adoption of two concepts - Modularity and Scalability. When implemented properly AND together, they provide the basis for achieving the economies of scale needed in handling trends in the technologies of today and tomorrow.

II. MODULARITY & SCALABILITY

A. Basic premise

Before we discuss how modularity and scalability can help, it is important to first understand what modularity and scalability means. Merriam-Webster defines the word modular as: **having parts that can be connected or combined in different ways**. It also defines scalable as: **capable of being easily expanded or upgraded on demand**

The key words from these definitions are:

- *Parts* – modules are units that have boundaries that make them distinguishable.
- *Connected* – purpose of the module is to interact with something else
- *Combined* – module may be autonomous and/or a piece of a larger system through combination.
- *Expanded or Upgraded* – Modules must be able to scale both vertically (expanded) and horizontally (upgraded)
- *On Demand* – It has to be relatively easy to swap or upgrade modules else it is not considered scalable.

So what's the big deal? The basic premise is that a system that is built with adaptable and adjustable pieces is far easier and cheaper to maintain, iterate, and manage over time. For example, when a car breaks down, a faulty part like an alternator can be isolated and replaced without a significant rework of the car. Chances are the alternator can also be purchased from other multiple vendors with better specifications or reliability.

So the basic idea is that a modular system allows a system to be created with modules that can be added and removed from systems with no impact and said modules can handle a wide variety of use cases (scalable). When systems are built with both these concepts, they are more able to adapt to the changing requirements and DUT variations.

B. Applying Modularity in Test Systems

Test systems are always modular to an extent. They have instruments, drivers, tests, data etc. The issue is not whether there are modules in the system, rather it's *where* the boundaries are around modules and more importantly *how* they are integrated with each other.

The process of partitioning a test system can be reduced to 3 tiers as follows. Each tier of modularity builds upon the previous and provides better separation of concerns.

Tier 1 - Identified

Tier 2 - Neutralized

Tier 3 - Isolated

A typical test system will have many tests that an operator runs either through a sequencer or from a front-end user interface. These tests may have some re-usable code, but for the most part the bulk of the measurement algorithm, instrument commands, error handling, communicating with the DUT, storing data etc. are all intertwined inside of the test.

There are many problems with tests implemented this way.

Firstly, there is a lack of consistency across tests (initialization, algorithms, etc.). This impacts how the tests are executed and can impact troubleshooting. For example, Test A performs a power setting in a different way than Test B. This may seem innocuous, but if the power setting routine has safety concerns, or if there is a more efficient or accurate way of doing it, then the lack of consistency across tests starts to have a real impact on the quality of the test system.

Secondly, any change to an instrument, DUT interface, test conditions, or data storage impacts the test code and as a result, the entire test code is now "dirty". With small bench testing, this is not an issue. For production testing, and especially testing in high-reliability industries like aerospace, a "dirty" test has a lot of costs associated with it. It must be re-verified which can be difficult to do once the hardware of the test system has been sold off to known standards etc. Another impact is that the "dirty" test which was once stable is not so anymore and starts to incur more and more troubleshooting and downtime on the test system and hits the production schedule.

Figure 1- Changes to one part creates a domino effect

Thirdly, if the test needs to scale due to rising complexity or test volumes, then it becomes a major refactoring effort. For example, if a system that was running tests now needs to support parallel testing, then the tests are basically thrown away and need to be re-written. If the hardware architecture of the racks is changed and resources are distributed across various computers in a test station, then tests would also be subject to significant changes.

So the trick is to implementing modularity in a test system is to ensure that:

1. [Tier1]: Modules (source or binary) can be re-used across tests and other modules in the system.
2. [Tier2]: Modules (binary) can be replaced without impacting other areas of the test system software.
3. [Tier3]: Modules are fully scalable such that you can add modules to scale a system vertically or distribute modules (across computers etc.) to scale horizontally.

C. Tier 1: Identifying Modules

In software systems, modularity takes various meanings. It could mean reusable blocks of source code or it could mean DLL's that can be shared. For LabVIEW, it could mean a .vi that constitutes a module.

This variation in the definition and understanding of this term in software is a significant contributor to the lack of effective modularization in test systems. Often times, systems will be developed with lots and lots of modular pieces – but they are all highly intertwined into the proverbial spaghetti code.

There are no hard rules for segmenting a system that would work across all test scenarios, but at a minimum, there should be separation between Test Code, Driver Implementation and DUT interfaces. A sample modular system is shown below:

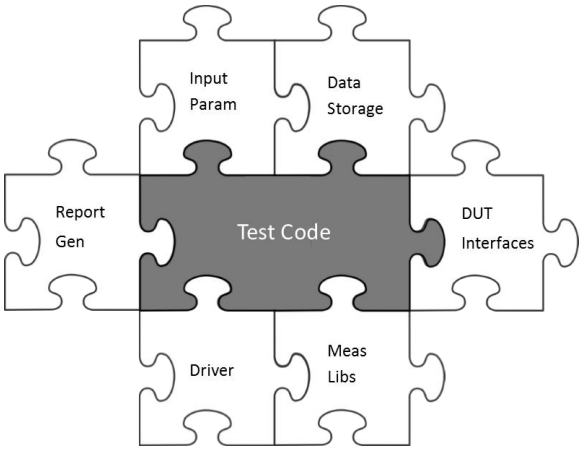

Figure 2- Modules must be separated by function/type

In the first tier, identifying module types gets a system to the first level of benefit which is that code concerns are separated and that the architecture is defined. The effort in this part will be primarily oriented around identifying the parts of the test system that may be re-used or frequently modified.

This task is never as simple as it may sound. This is because it requires the skillset of someone with full domain knowledge. The architect of the system must understand the tests and their common functions, the equipment and its use cases etc.

The benefits from a Tier1 modular system is cleaner and easier to manage source code. Re-using code across tests will also provide an immediate benefit with consistent measurement steps etc. and will accelerate test system development time. Modular code is also more supportable in general, so will result in shorter bug disposition times in the early days of the system.

D. Tier 2: Neutralizing Modules

So if modularity makes so much sense, then there has to be a reason why it's benefits have eluded so many test system projects.

Since modules are pieces of the puzzle, their functionality comes from the interfaces that they provide to be connected or combined with other modules and systems. In an ideal world, the modules all have standard and compatible interfaces and can interact with each other in a pre-defined way. This way, the module can be replaced with another module that has the same interface and everything should work.

In the real world, such modularity rarely exists. There are industries like semiconductor that have achieved some standardization, but other industries like aerospace are severely lacking this type of common ground. The technology and products are vastly different. Combined with the fact that a significant amount of test system development is home-grown, the logistics of getting the right people together to find solutions is even more elusive. The end result is that driver changes or substitutions continue to require significant test code rework.

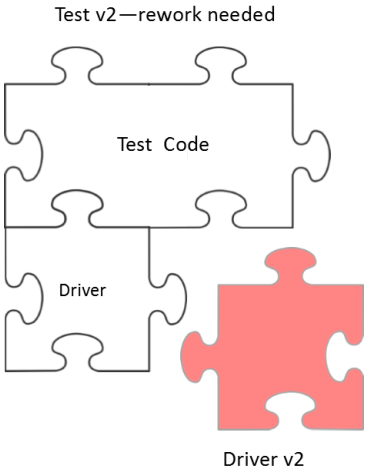

Figure 3- New drivers don't just fit into existing code

1) Instrument Drivers

Instrument vendors attempt to fill the void and find standard interfaces that work across the wide swath of use cases. This eventually leads to a least-common-denominator approach. The outcome is driver interfaces that may be common, but incomplete, inadequate, and require additional patchwork for integrating into a test system's software anyways.

Many attempts have been made to provide standard interfaces to instrument types. The IVI foundation has worked tirelessly to achieve the ideal case of modularity. This would benefit both vendors and customers with a "free-market" approach to building systems.

As noble as the effort is, it is unlikely to succeed in the near future. This is because there is too much fluidity in the equipment industry alone. Consider, for example, how the traditional role of signal generator and analyzer is now being wiped away with modern network analyzers with full measurements embedded in the instrument. Also consider recent trends where Software Defined Measurements are abstracting the Data Acquisition (DAQ) from the measurement processing software. All this fluidity in the instrument side of things complicates the task of standardizing interfaces. Instrument vendors also typically lag the industry as a whole because the device technology develops first before the test technology adapts to provide efficiency.

For any production worthy system, it is therefore **unavoidable** to need to create *customized* boxes around vendor instrument drivers with *tailored* interfaces that make sense *specifically* for the DUT and its value chain. Boxing is simply the process where the vendor's instrument driver is wrapped with an external interface definition that is more stable and meaningful. By keeping the "box" the same, the internal driver can be swapped out with no change to test code.

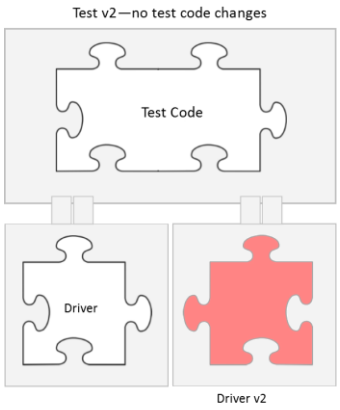

Figure 4- Boxing neutralizes the external interface

The task of boxing is not simple. It requires design and understanding of the driver to ensure that the intended use cases are accommodated. The benefit of modularity is at its peak during future changes and iterations where you get a very high level of re-use. There are however immediate benefits to be had from boxing such as faster troubleshooting and system sell-off, safety and error handling, and cleaner configuration management.

Consider, for example, a company develops a sequence of settings to mitigate noise floor during analyzer measurements – they can integrate that into their boxed driver and re-use that across tests in the same system, or across product lines or test phases that share the measurement technique.

2) Test Code

The concept of modularity with software drivers is easy to understand since they are associated with the idea of benchtop instruments. We see instruments as modules with interfaces, protocols, and data transactions.

Tests are not traditionally viewed in this way. From a software perspective however, there is no fundamental distinction between a test and a driver; they are both identical in that they take inputs, perform measurements and algorithms, and yield an output. In many systems, tests are viewed as the highest tier of the system.

Take for example a test for an amplifier that needs to perform a power in vs power out curve from -15dBm to -5dBm at a particular measurement frequency with 100 measured points. This may be a test that is developed for a product line and works fine. However, when things change like number of points to measure, power levels, limits, frequencies etc. Then the test is susceptible to changes.

The more ideal way to handle the test would be to write a test that is modular – It takes input parameters of power levels, point counts, etc. and provides a known measurement outcome. This is structurally no different than issuing a power sweep command to a benchtop instrument.

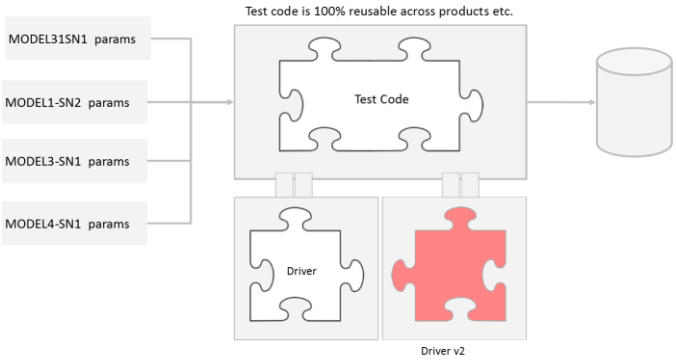

Figure 5- Tests must also be modular and have interfaces

Following the guidance for instrument drivers, tests also need to boxed well with design and upfront thinking on anticipated use cases. When tests wing it with some using parameters and some hard-coded values, then the benefit is erased for the most part because the rework required to these tests later on will be comparable to the non-parameterized variant of the test.

Benefits of neutralized and re-usable modules are significant because the pain points with costs are in the iterations of these systems.

E. Tier 3: Isolating Modules

Systems that have reached Tier 2 will have modules well defined and with clean interfaces. This goes a long way in getting re-use and reducing iteration costs in a system.

There is, however, one aspect that still inhibits portability of modules – **link dependencies**. When software is segmented into modules they are often built as shared objects such as .DLL files. This allows them to be re-used. In software, this is done through linking dependencies during compilation between the test code and driver, for example.

The reason that this is bad is that despite having neutral interfaces, the modules are still intertwined into each other with software level linkage. If a system needed to integrate multiple versions of the same module on a single system, it would need test code to be re-versioned as well.

If a driver was replaced, the interface neutrality from Tier 2 would not require test code changes, but it *would require* that the test code be re-compiled and released with a new version. This fracturing only gets worse as more changes are piled onto them and all traces of modularity can disappear.

For a system to be scalable it needs modules to have the ability to move freely across systems and processes, retain multiple versions of modules on the same system, and be able to cross computer or network domain boundaries.

Consider the following scaling needs in high-technology and aerospace test systems:

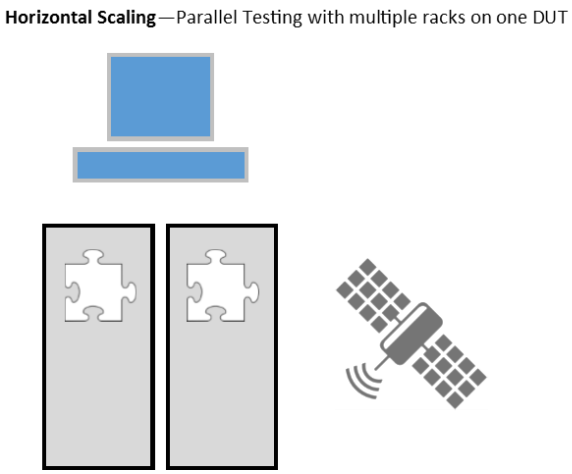

Figure 6- Example of horiz scaling with duplication

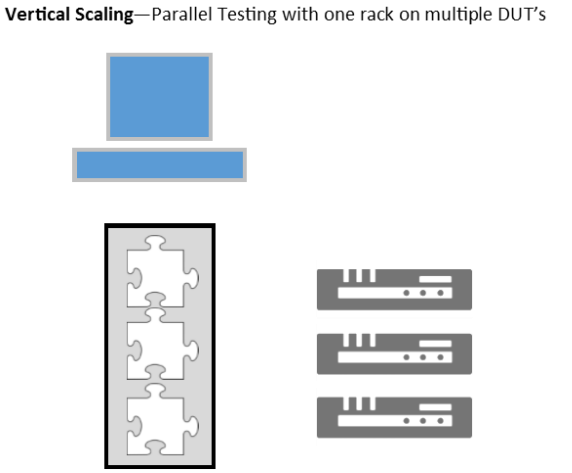

Figure 7- Example of vertical scaling through addition

This level of scalability can only be achieved with a layer of isolation between modules. On computers, the boundary of isolation is the process so it makes sense to be able to host modules in separate processes. The modules, however, need to communicate with each other.

In Tier 2 code where modules are linked in the same executable process, exchanging data is easy because code just calls other code and data moves freely in the process' memory space. With Tier 3 modules crossing process boundaries, a data bus needs to be implemented that supports data exchange – much like how PXI has a backplane for data interchange and modules just plug into the chassis.

For computer architectures, TCP/IP is the ideal data bus for data exchange.

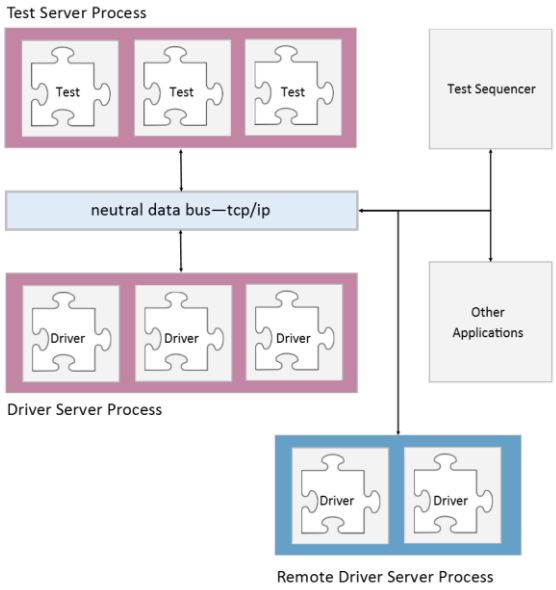

Figure 8- Modules must be highly accessible

With this architecture, a server process is implemented that can load any number of components dynamically at runtime. This allows the server process executable to be implemented without dependencies on what is loaded into it. This dynamic loading capability is supported through technologies like shared objects and Reflection in Microsoft .NET.

Any number of servers can be spun off on any number of computers and hosts that allow full scalability of components and services to build the system that is needed with maximum re-use and minimum technical risk.

This design not only makes equipment substitution easier but allows other use cases such as simulation and emulation of modules. Take for example a test system that is being developed before the DUT is ready. A simulated DUT interface can be inserted into the system for development and then replaced out later.

When systems have lots of modules that are being iterated, the domino effects of changes are higher in systems with more dependencies. This poses an adjacent problem with configuration management because trying to use the same code base across products with minor changes still results in major code branching which are hard to maintain and support over time. With isolated interfaces and the server architecture, you can release software with multiple versions of the same component that are loaded at runtime. This minimizes the number of CM branches and makes code more supportable.

III. ENTERPRISE BUSINESS CASES

A. *Benefits*

In the previous sections we primarily discussed the technical merits for modularity based on real-world testing scenarios. The case for modularity, however, extends beyond technical concepts by presenting a compelling business case for larger organizations that perform lots of testing across the value chain.

When a new product model or iteration/generation of product is taken on for manufacturing, there are fixed costs associated with getting a production system and software ready. As we mentioned before, the changes in technology and devices require changes to test code etc. There are also variable costs such as operating and executing tests.

The impact of technology changes hits the fixed costs directly. Assuming variable costs are the same across devices, the key cost reduction for systems would be to reduce fixed costs that are incurred. In non-modular systems, these costs escalate over time and eat into the already thin margins.

In a modular and scalable system, the high levels of re-use and scalability allow significant mitigation of these fixed costs. This allows a manufacturer to take on newer business with lower costs and less technical and schedule risk.

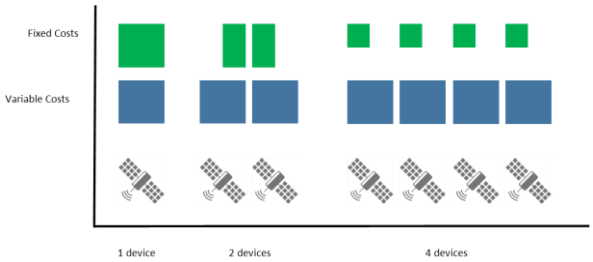

Figure 9- High re-use results in splitting fixed costs

Larger companies typically perform manufacturing through various stages of the value chain. Being able to re-use modules across the value chain both vertically and horizontally also gives these companies significant economies of scale.

The value chain itself will be different based on the type of manufacturing. For example, a low-mix, high volume manufacturing may have a value chain from prototype, design verification, low volume, and high volume manufacturing. A high-mix and low volume scenario that is typical in aerospace systems may have a value chain like that shown in the diagram below.

The diagram below shows a hypothetical case where a large company has invested in modularity and is able to re-use modules across the value chain (vertically) and across products (horizontally).

Figure 10- Sample of vertical and horizontal integration

B. Costs

The costs of employing modularity always appear to be higher than the cost of "just getting it done". This is because it requires upfront engineering, the right skillsets, and infrastructure. This is scary to some managers who would feel that the upfront engineering will blow their budgets away for a future benefit that is not in their scope of delivery.

If a company chooses to build the infrastructure to support such modularity, then, yes - it is more expensive in the first phase of the project. Such projects should not be embarked on with the expectations for high ROI on the first device tested.

The ROI and break-even that is achieved will depend upon how much testing is performed and the level of fixed costs that are typically incurred each time there is a material change to the products being tested. More complex and higher volume testing will yield ROI much faster than simple bench test scenarios due to the higher fixed cost structures that are mitigated.

One solution to accelerating the ROI would be to leverage a COTS platform that provides the infrastructure needed to build a scalable system for your production. This way, the upfront hit of building infrastructure with all the life-cycle and technical risk overhead is mitigated and your test system projects can get immediate benefits from the scalability.

CONCLUSION

Test systems are being stressed by the higher complexity of new devices. Manufacturers and test organizations are feeling the increase in costs of building and iterating their test systems due to rising labor costs and the frequent iterations in devices.

Modularity and scalability are two concepts that allow test systems to achieve high levels of re-use and portability. These concepts are familiar but hardly implemented well in systems. Improper implementation does not yield the level of re-use for companies to attain the ROI promised.

This paper presented 3 tiers of modularity that allow incremental benefits with Tier 3 being the most portable solution of them all allowing vertical/horizontal scaling of the test system for use cases such as parallel testing.

Besides the technical reasons to modularize, there is a real compelling business case to adopt such modularity. We illustrated a sample use case of a company with an integrated value chain that can benefit from high levels of modularity and re-use.

In the end, modularity is hardly sold on those who have not yet been bitten over and over again by the costs of iterating test systems. Those that have lived through will appreciate the need for such modularity, and they may have strived for it in vain. It is my hope that this paper has shed some light on a tenable solution for modularity that will work for your test operations.

ABOUT VERIFIDE

Verifide is a software company with a platform for test automation. The distributed Tier 3 modularity presented in this paper is included in our software, giving you the infrastructure needed to build a scalable system out of the box. We also offer a comprehensive data management platform for scalable test data storage and analytical tools.