FUNDAMENTALS OF BUILDING A TEST SYSTEM

Test Executive Software

CONTENTS

Introduction

Most test systems are designed fundamentally around two concepts: efficiency and cost. Whether working in the consumer electronics industry or in semiconductor production, test engineers are concerned about individual test time and total throughput of a test system, and how these affect resources. When applications grow large enough to constitute multiple tests, a variety of instruments, and several units under test (UUTs), they inevitably require the oversight of test executive software to continue to address their cost and efficiency concerns.

Test executives are typically implemented as in-house solutions, or purchased as a commercial off-the-shelf (COTS) products. In the prototypical build versus buy argument, a test architect must determine whether it makes more sense to write a custom test executive or to invest and integrate an existing solution. Before deciding whether to build or buy a test executive, it is necessary to understand the purpose and core functionalities of this kind of software. This guide summarizes key functions of a test executive and explores practical scenarios to apply this knowledge.

Background

A test executive can automate and streamline large test systems. Sitting at the top of the software stack, it consolidates common functions, such as test execution, result collection, and report generation, up from the individual test level. The features of this solution are not unique to a particular UUT, so a variety of applications can use the test executive as a framework. This means that developers writing test code in G in LabVIEW software, C, .NET or other languages can focus on the intricacies of testing a particular device, while common functions across all UUTs are maintained at the top-level test executive. Overall, the test executive defines such common functions in a manner that proves efficient from a development, cost, and maintenance perspective.

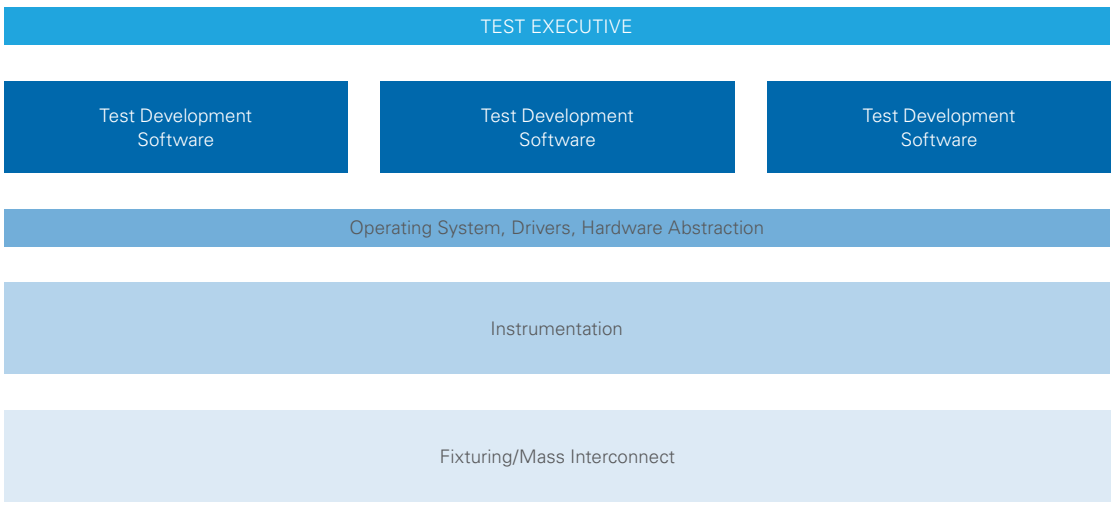

Figure 1. Test executives allow the separation of individual test development from the architectural needs of the entire test system by accomplishing tasks common across all tests at a higher level of abstraction.

Features of a Test Executive

Depending on the size of a company, the scale of a particular tester, and the variety of devices under test, complexity of a test executive can range from simple to advanced. This guide outlines common features that this software might contain. Some features are crucial to all implementations of a test executive, while others represent additional functionality that may not be strictly necessary. Each feature outlines an estimated amount of development time to complete. These estimates are based on experience with hundreds of automated test customers, as cited in Test Executive Software—Build or Buy? A Financial Comparison Using NI TestStand.

Test Sequence Development Environment

A test executive provides a development environment in which to architect test sequences. This feature is both fundamental— providing the development interface for the whole execution— as well as complex. Sequence architecture encompasses the ability to implement branching or looping logic, a means to import test limits, and the specification and organization of individual test code. Interfacing with test code requires flexibility across a variety of built formats, such as DLLs, VIs, and scripts, as well as integration across different development environments. Test executives may also use test code that originates from a source code control provider.

Implementing a test sequence development environment in a custom-built test executive can take around 100 person-days to complete, whereas a commercial solution provides this environment outright. This feature requires the most development time for an in-house solution because of the range of functions that a development environment provides. However, it is fundamental to the sequence architecture experience and cannot be omitted.

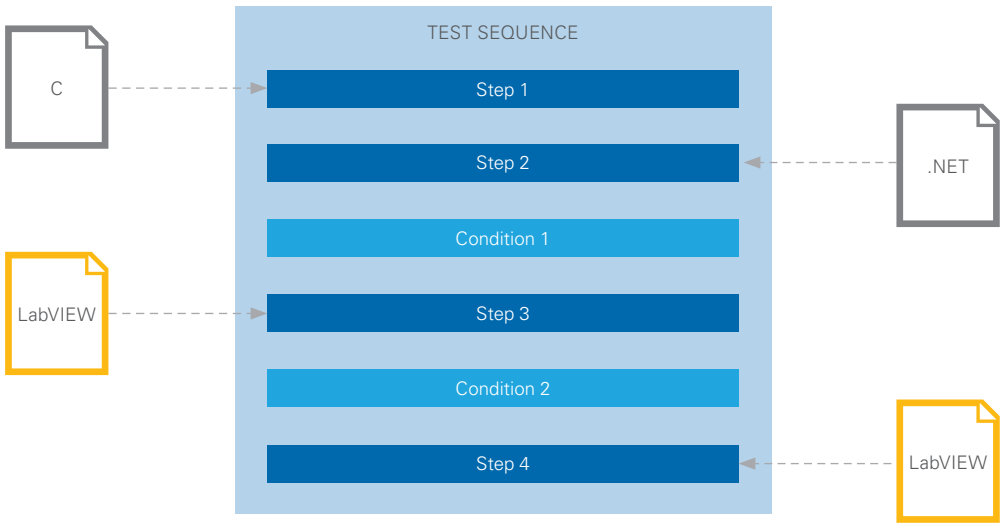

Figure 2. A productive sequence development environment gives test engineers the ability to develop and debug complex sequences that call into existing test code.

Custom Operator Interface

The operator interface is the display through which the operator interacts with the test system. It typically allows for the selection of key input parameters, such as UUT identifier, test sequence to execute, or report path. It also contains a Run or Start button to control execution. Many large test systems today require a professional GUI differentiated by application or company and written in the programming language of developer choice. In addition to customization, this highly functional interface includes the ability to load, display, and run test sequences complete with interactive user prompts, execution progress indicators, visualization of test data, and localization.

Implementing a custom operator interface can take a range of eight to 32 person-days' worth of development time. A COTS solution can reduce this estimate because of existing libraries and UI controls. Developing a custom operator interface can be a nontrivial time investment, regardless of whether the test executive solution is built or bought. Test engineers who do not feel this component is crucial to their system may instruct operators to work through the development environment instead.

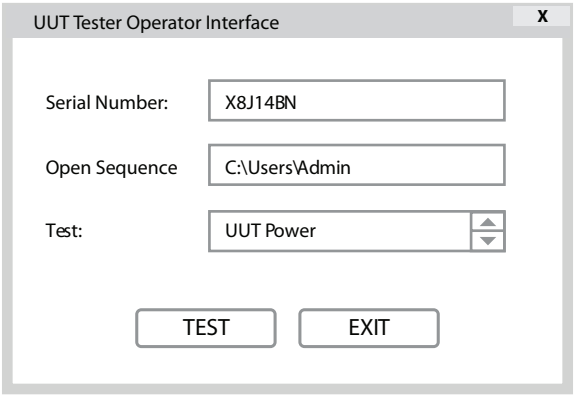

Figure 3. A customer operator interface uniquely identifies the UUT, company, application, test, and role of the operator for a given test sequence.

Sequence Execution Engine

A core provision of the test executive is a sequencing engine. The sequence execution engine is responsible for all the actions required to evaluate a UUT. This includes calling individual test code, creating a flow for execution between tests, and managing data between tests. The sequencing engine is what executes a given test sequence, whether in the development environment, through a custom operator interface, or on a deployed tester.

Implementing a sequence execution engine requires a minimum of 15 person-days to develop in-house. However, it is a must-have feature of all test executives.

Results Reporting

Given the abstracted role of the test executive, this piece of software is responsible for consolidating individual test data, storing temporarily into memory, and publishing comprehensive test results. Reports can come in a variety of formats, including XML, text, HTML, and ATML. Data may also be pushed to a database following execution. The test executive makes this variety in formats possible through extensible reporting options. Results reporting is a necessary component of many test systems.

Developing result collection and a report generator from scratch can take around 15 person-days, depending on the specific report required. Given a built-in report generator in a COTS solution, results reporting can be customized to meet the needs of an application in a person-day or less.

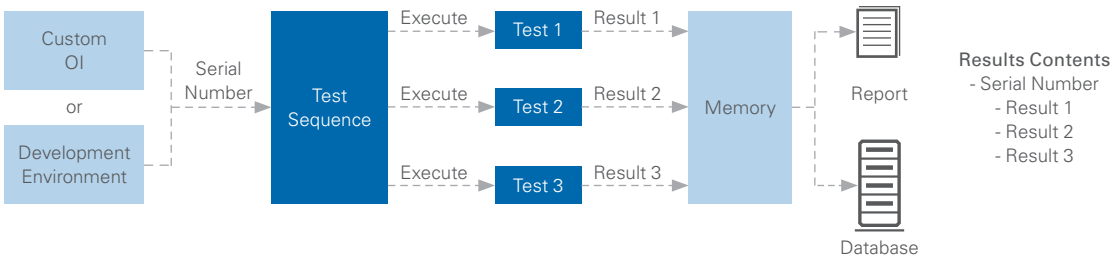

Figure 4. Part of a test executive's role in a test system is to consolidate results across an execution and publish to a report or database.

User Management

It may be necessary to separate roles and responsibilities at the test executive level. User management tools effectively compartmentalize the responsibilities between the overarching test architect, the individual test developer who writes and debugs test code, and the operator or production manager who runs the test. Functions available to a given user may even be password protected to prevent misuse of the test sequence.

Implementing a user management system in a custom test executive takes about five person-days' worth of development time. Although not necessary for the use of a test executive, user management tools do not require a significant amount of developer effort to implement and can simplify the enforcement of test executive responsibilities.

Privilege	Architect	Developer	Operator
Edit	√	—	—
Save	√	—	—
Deploy	√	√	—
Loop	√	√	—
Run	√	√	√
Exit	√	√	√

User	Level
Mark	Operator
Larry	Operator
Julie	Developer
Scott	Developer
Lauren	Architect

Table 1. Similar to Windows file permissions, a user manager separates the roles and responsibilities associated with a test executive.

Parallel Testing Capabilities

Parallel test involves testing multiple devices at the same time, while still maintaining proper code-module performance, result collection, and UUT tracking. Parallel test approaches range from pipelined execution, where test order is maintained, but the test executive can test across multiple sockets concurrently, all the way to dynamically optimized, batch, or other complex execution styles.

Implementing parallel test is typically the most time-intensive for a test executive developer, and can take 100 person-days to develop from scratch. Although parallel test may require a large amount of time to develop, the ability to scale up an execution to mitigate throughput needs in a large test system is often crucial. Many organizations do not consider parallel test when first implementing a test executive, and learn later that it is a function they ultimately need and cannot settle on.

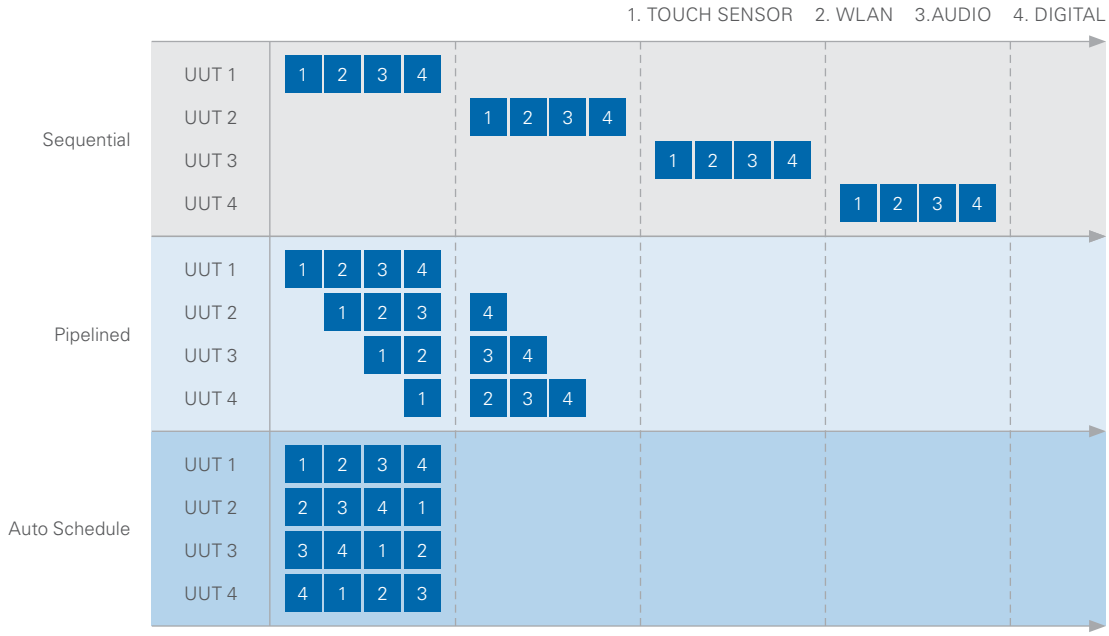

Figure 5. Parallel test capabilities allow for dramatic increases in system throughput without a re-architecture of the test executive.

Unit/Device Tracking and Serial Number Scanning

When testing across multiple UUTs, it can be necessary to uniquely identify and track each device tested. This information can be stored alongside test results for specific analysis at the unit or batch level, or to pinpoint the source of error when things go wrong. Device tracking can range from manual entry by an operator on a keyboard, to a fully automated scanner that loads UUT information after reading a barcode.

Developing this type of functionality can take five person-days from the ground up, or about one person-day to customize when provided by a COTS solution. UUT tracking is not required for every test system. However, it is useful where high-volume, high-throughput testing is needed, such as the semiconductor or consumer electronics industries.

Test Deployment Tool

Most large test systems are not architected in isolation; they represent solutions for multiple test sites or for an entire production floor. A test executive plays a key role in system deployment by providing a mechanism or utility to package the entire software stack into a built, distributable unit. A test system can be distributed in a variety of ways—an architect may be looking to deploy an image of the test system or a fully functional installer containing all necessary dependencies and run times. More information on methods is at Deploying an Automated Test System.

Deployment is a nontrivial task, and it can take a team of developers as many as 20 person-days to implement from scratch. With an out-of-the-box deployment utility from a commercial test executive, it still may take three person-days' worth of time to successfully deploy. Given the applicability of this feature to multiple test sites, it is often necessary to have in a test executive solution.

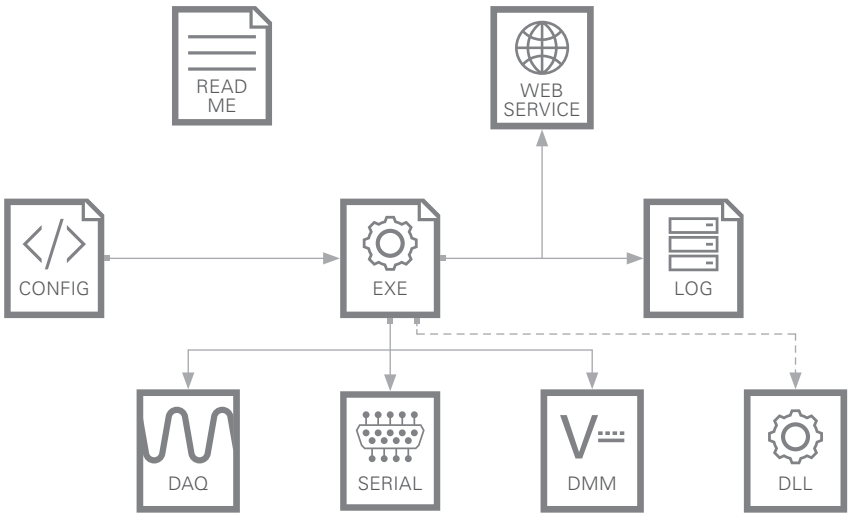

Figure 6. Deployment involves packaging all necessary components of a test system through a deployment tool or build server, before distributing to the wanted test stations.

Maintenance

Just as with any other component in a large test system, test executives must be properly supported to ensure their performance over time. This encompasses expanding to include new tests, maintaining compatibility across software or OS upgrades, and fixing any bugs that are detected. Maintenance of a test executive solution even extends to the realm of documentation. This is a crucial resource that operators, developers, and architects rely on when working with a test executive.

Although it is difficult to predict the needs of a given tester, 15 percent of the initial time spent to develop a custom test executive is spent annually in maintenance. Total development time includes the estimated 20 days required to produce adequate documentation. The granularity of support for a test executive can vary, which changes these cost estimates dramatically. However, it is inadvisable to implement any test executive solution with the minimum in maintenance efforts.

Practical Scenario 1

Jonathan is a test engineer in the design lab at a small company that provides low-cost consumer electronics. A remote controls each device, and Jonathan specializes in writing test code to validate the transmitter-receiver communication between the remote and the prototype, before the device is sent to production. With his company's recent expansion, Jonathan has less time per device to perform the requisite testing. He needs to automate the execution of his existing validation code, so that he can spend more time writing code for new devices. Therefore, he decides to employ a test executive to sequence through his test code.

The table below shows the needs that Jonathan identifies in a test executive.

FEATURE	IMPLEMENTATION
Test Sequence Development Environment	Jonathan needs a development environment in which to architect his sequences. The test code he has been working with is already fairly modular, so he should only have to call and loop over test code within this environment.
Custom Operator Interface	Jonathan wants to be able to execute a set of test code with minimal interaction. He wants to have to specify only a few relevant parameters to identify the device, wanted tests, and report path. However, given that he is the end-use operator, it is not crucial to have an interface separate from the development environment.
Sequence Execution Engine	This is the core need for Jonathan's test executive solution. Each test consists of several individual LabVIEW VIs that must be executed sequentially.
Results Reporting	The existing test code currently prompts the user to generate a new Technical Data Management Streaming (TDMS) file for a given prototype. Each subsequent VI deposits minimal test results into this same file. The test executive needs only to automate the creation of this TDMS file, and then execute the remainder of the test code to generate results according to convention.
User Management	User management is not a priority, because Jonathan plans to architect, develop, and operate this test executive.
Parallel Testing Capabilities	Jonathan executes his tests on only one prototype at a time, so UUT volume is not a concern.
Unit/Device Tracking and Serial Number Scanning	Given that testing is done on design prototypes, there are no assigned serial numbers to track. Instead, Jonathan tracks each UUT by a unique name that the operator enters at run time.
Test Deployment Tool	Jonathan does not intend to deploy this code to additional testers. His test bench is unique to the design lab, and separate from the manufacturing facilities.
Maintenance	This project belongs exclusively to Jonathan. He will implement and maintain whatever test executive is selected. He does not plan to document his work, as he will be the sole person to work on and use this test executive.

Table 2. Jonathan's needs in a test executive solution center on simple automation of existing validation code.

Jonathan decides to build a test executive in-house. He does not have complex sequencing or reporting needs, and does not have plans to deploy this system to other users or test stations. If he purchased a commercial solution, he would not see the return on investment as the majority of features would not be used. Instead, relying solely on his software knowledge and previously purchased application software, he can develop a sequencer to meet his needs in as little as 10 person-days.

Jonathan builds his test executive in LabVIEW software. He architects a solution with a simple interface that gives operators the ability to call a predetermined set of test steps and select the path of the TDMS log. Jonathan can occasionally make small changes to the sequencer as he introduces additional tests for a new prototype. Overall, the design lab sees an increase in productivity thanks to the implementation of this sequencer.

In this particular case, a custom-built test executive proved to be the best option for Jonathan and his criteria. Often, an in-house solution is the first step taken when scaling up to sequencing or full automation, and may be more appropriate overall for a test bench application when compared to the needs of a production setting.

What If…

- After a few months, a new test engineer is hired into the design labs and begins to assist with the testing process. How will this engineer learn how to operate the sequencing tool, or effectively troubleshoot any errors or bugs that appear?
- Jonathan transfers to another department, or leaves the company. How is the knowledge required to update or fix the sequencer maintained?
- It becomes necessary for the sequencer to perform a functional evaluation of an entire prototype. How would it incorporate additional test code that different engineers write in other languages, with different programming paradigms and reporting techniques?
- The test executive is ported to a production setting to ensure consistency in testing. Can these solutions scale up to such needs?

Practical Scenario 2

Dave's company is designing a new functional tester to be implemented at the end of a manufacturing line. Currently, UUT testing is performed by manually executing across a series of existing, disaggregate pieces of code. This process significantly limits throughput of the line, and Dave wants to employ a test executive in automating this process. The company does not standardize on a test executive, and each group typically chooses its own from within a small pool of commercial solutions and innumerable custom-built solutions.

The table below shows the set of requirements that Dave outlines for the tester.

FEATURE	IMPLEMENTATION
Test Sequence Development Environment	A productive development environment that supports key features of a test executive is a must. The environment must enable the sequencing of LabVIEW, .NET, and Python code.
Custom Operator Interface	Dave ultimately wants an operator interface that is customized to the company. He also wants to remove most functionality beyond a Run button.
Sequence Execution Engine	This is an obvious need for this system to address throughput needs.
Results Reporting	Currently, each test individually logs data to an SQL database. There is a need for consolidated result collection by the test executive, with aggregate results communicated to the database and identified by a serial number.
User Management	The majority of interaction with a tester occurs at the production level by the operator. Dave prefers a user management tool or customizable interface that removes development privileges from the operator's view.
Parallel Testing Capabilities	As long as tester throughput matches production throughput, Dave does not need to test multiple UUTs at once.
Unit/Device Tracking and Serial Number Scanning	A serial number identifies each component and assembled UUT. A barcode scanner is used to track such information. The test executive must be able to propagate such information across the different tests it executes.
Test Deployment Tool	Dave needs to deploy the final product to 10 additional testers.
Maintenance	The test engineering department will maintain the test executive, either in full capacity for an in-house solution or where needed for a COTS option.

Table 3. Dave's evaluation of test executive software is driven by underlying throughput requirements on functional testers.

To make his decision, Dave also weighs the financial considerations of the tester. He estimates that a new tester will consist of a large, high-performance PXI chassis and embedded controller pair. Because of the nature of tests required to evaluate the UUT, the chassis will contain several modules that range from DAQ cards and PXI instruments, such as digitizers and arbitrary waveform generators, to RF test equipment. The cost of each tester will sit at around $100,000 USD regardless of the test executive solution.

When evaluating the software stack, Dave notes that purchasing a COTS solution adds to the project cost. A development license of the test executive costs a few thousand dollars, with the added cost of $500 USD per additional tester for a license to deploy.

Dave believes he can save on test executive cost by building a custom solution in Python. The language is open source and the development environment is free—both are benefits he believes will more than offset the additional development time required to build a test executive in-house.

The test engineering team is proficient in Python, which delivers core functionality—a sequential sequencing engine, database connectivity, and code reuse of their existing tests—in the required timeframe. The test executive is successfully deployed to the manufacturing lines. The test engineers are occasionally called in to fix bugs in one or more of the testers.

What if…

- Production demands on the manufacturing lines increase, such that the existing test executive cannot meet throughput needs. It is necessary to scale up to parallel test.
  - How much additional development time would it require to attempt to implement this functionality? How does this affect the cost comparison of a custom versus COTS solution?
  - Assume throughput needs of the tester cannot be met because of known multiprocessing limitations in the Python language. Dave's team is faced with purchasing additional hardware to reuse the current solution, or pursuing another test executive altogether. How does this further affect the cost comparison of a custom versus COTS solution?
- The test engineering team cannot always service or upgrade the test executive because of other priorities.
  - How is production affected when such needs arise and the team cannot help? How does this downtime factor into system maintenance costs?
  - How is the time that the test engineering team spends maintaining the tester quantified? How does this factor into system maintenance costs?

Practical Scenario 3

Karen works at a company that designs and produces small medical devices. Each product has its own fully automated production line. Although each group enlists a test executive for top-level system management, the company has not standardized on a solution. Recently, a new test manager has come aboard and expressed interest in test executive standardization. Karen is tasked with the responsibility of selecting the commercial solution, existing in-house product, or new development effort to act as the de facto test executive.

Karen compiles the following list of requirements across the assorted groups responsible for each product.

FEATURE	IMPLEMENTATION
Test Sequence Development Environment	The test developers require a flexible development environment that, specifically, can interface with their LabVIEW and VB.NET code. Tortoise SVN is used for source code control, and integration with this tool is required.
Custom Operator Interface	The test manager wants to customize operator interfaces according to the product being built or tested. Operators have reported they want a progress indicator to update test status when overseeing a tester.
Sequence Execution Engine	A definite requirement for all testers.
Results Reporting	All production systems must conform to a company-wide, HTML reporting standard.
User Management	The test engineering team consists of a few system architects and a larger number of test developers. The test manager wants to separate responsibilities between these two roles.
Parallel Testing Capabilities	When performing functional testing on an assembled unit, production lines evaluate one UUT at a time. However, board-level testing should be optimized to execute as quickly as possible. To meet the needs of all testers, parallel test is needed.
Unit/Device Tracking and Serial Number Scanning	UUT information is tracked by operator input for each product and board in the company.
Test Deployment Tool	The company has a dedicated team of test engineers that writes test code. This team must be able to deploy from the development environment in their lab to the wanted production setting. Currently, this is accomplished manually.
Maintenance	The test manager requires a formal maintenance plan as part of the standardization effort. Part of this plan needs to accommodate an OS migration that the company is facing later this year when their current selection goes end-of-life.

Table 4. Karen's interest in a test executive solution stems from standardization needs across a variety of testers.

Given this criteria, Karen eliminates all of the existing in-house solutions. Most of them were architected as part of a focused effort to get a single tester off the ground. There is little consistency in architecture that would lend for extensibility into other production lines, specifically in terms of sequencing needs, operator interface customizations, and effective deployment practices. Additionally, it has already proven difficult to track down the test engineer responsible for a given test executive when a problem occurs in the software, or a modification is made to the device.

Instead, Karen proposes a commercial solution to her manager. The test executive is made by a well-known vendor whose other hardware and software tools are already used in the testers. Out-of-the-box features of this test management software can meet the range in sequencing paradigms that testers require, and employ the specific reporting format needed. The test executive includes a set of tools designed to meet some of the other testers needs, including a user management tool and deployment utility. Given that a commercial vendor maintains it, Karen's manager should not have to worry about incompatibility across OS migrations later.

Karen's company is ultimately successful with their decision. Overall, the test executive provides a flexible framework that scales across the different production lines. Standardization across a purchased test executive comes with additional benefits that the company can use. The vendor provides training to facilitate the test engineer's acclimation to the new software. Part of their purchase of the test executive includes a maintenance contract, wherein the vendor agrees to provide routine patches and upgrades. The company also has access to technical support resources that can assist in troubleshooting their test sequences.

The commercial solution remains the standard at Karen's company. When test engineers need to be replaced, because of promotions, retirement, or natural attrition, the test manager can hire an individual with experience in the test executive. The company successfully migrates from an obsolete OS up two complete versions while maintaining their selection in test executive. As new products are developed, the extensible architecture can continuously meet production needs.

Conclusion

Regardless of company size, industry, or individual test criteria, it is necessary to implement a test executive for top-level system management. This implies introducing a degree of abstraction that separates common functions of a system from the specific functionality of test code. A complete evaluation of test executive needs is necessary before architecting the ultimate solution. Many test engineers grapple with the decision to build or buy their test executive. Selection of one path over another involves careful consideration of each solution's benefits from a cost, functionality, and maintenance perspective.

Next Steps

TestStand is industry-standard test management software that helps test and validation engineers build and deploy automated test systems faster. TestStand includes a ready-to-run test sequence engine that supports multiple test code languages, flexible results reporting, and parallel/multithreaded test.

Although TestStand includes many features out of the box, it is designed to be highly extensible. As a result, tens of thousands of users worldwide have chosen TestStand to build and deploy custom automated test systems. NI offers training and certification programs that nurture and validate the skills of over 1,000 TestStand users annually.

Learn more about TestStand