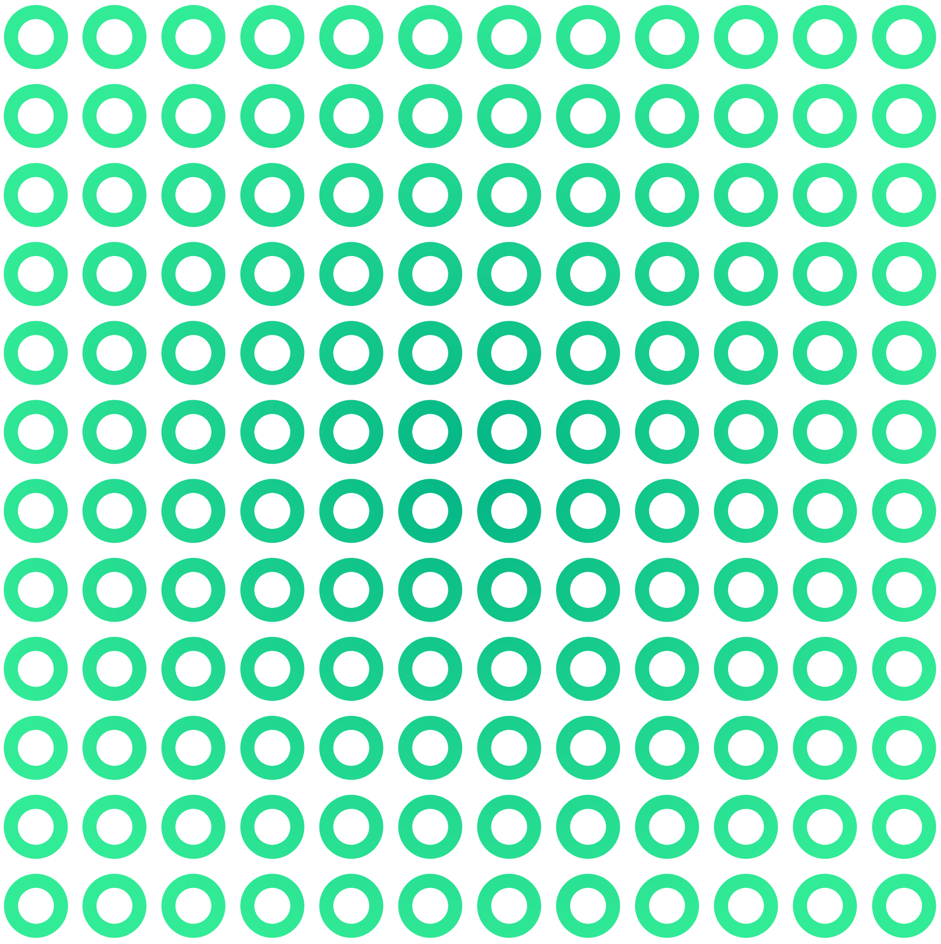
White paper

# How Using a Test Executive Prevents Reactive Development



# Table of Contents

As devices become more complex, test teams are under increased time pressure to get products out the door, making an optimized test strategy more important than ever. Test engineers have the difficult task of building the tester for the subsequent product while maintaining previous testers. This paper explores the significant challenges test engineers face and explains how a test framework addresses the problem by increasing productivity, reducing rework, and decreasing maintenance. Finally, it introduces NI TestStand as an off-the-shelf solution for building and maintaining a test framework, highlighting the added benefits of risk reduction, flexible development, and feature implementation.

## Stuck in a Vicious Test Design Cycle

A test engineer's priority is ensuring a working product is delivered to the end customer. But as products become more complex, so does the testing. Let's take a light bulb, for example. Smart light bulbs have built-in Wi-Fi and Bluetooth functionality—and sometimes even speakers. That means that over the last few years, a test team has added testing for power electronics, wireless connectivity, and acoustic test.

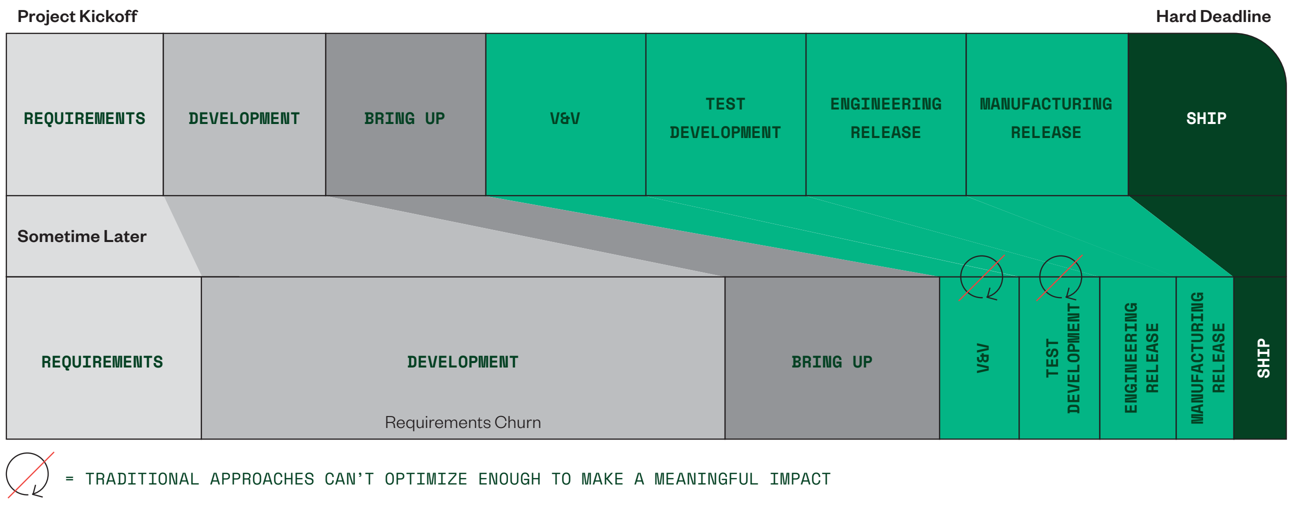


FIGURE 1

Advanced product capabilities have extended development times, shortening the time available for test development.

Figure 1 illustrates the difference between the planned time spent on each phase and the reality. Because the deadlines stay the same, any delay in the earlier phases results in test development time getting cut short—test engineers are asked to test more functionality in less time. When we take time away from the test development phase, we force test engineers to be more reactive rather than proactive. They simply don't have time to consider how to approach this efficiently.

### The Cost of Reactive Test Development

This trend of testing more in less time has been around for years but is often overlooked because the cost of this type of reactive development isn't immediately evident. One significant side effect of reactive development is it leads to a lack of flexibility, which can lead to a loss of productivity for test engineers as they waste time finding ways to add features to a bloated, rigid tester. At a certain point, it becomes inefficient to continue building on the same tester, and engineers are forced to start over and waste time rebuilding the same features as before.

Not only does this lead to a large amount of rework, but it also increases the number of testers that need to be maintained. The maintenance phase comes with its own set of challenges, especially if an engineer who built the tester leaves the company without adequately documenting it. Their departure can also lead to a rebuild of the entire system.

## Testing with a Scalable Framework

One way to escape this cycle of reactivity is to build a test framework. A test framework is a high-level software application used for developing, executing, deploying, and reporting on automated test sequences.

| 75% | 67% |
|---|---|
| Time saved during development | Reduction in time spent on maintenance |

FIGURE 2

Percentage of time saved by NI customers who integrated TestStand into their organization. Check out the **build or buy guide** for more information.

When organizations leverage this type of framework, they save up to 75 percent on development time and reduce time spent on maintenance by 67 percent.

### What makes a good test framework?

When you look at a set of testers, you will see common and unique elements. These common elements, when identified, are the foundation of the reuse model for your test organization. For example, most testers require a user interface; however, aspects of the interface must cater to the preference and competencies of the user.

| COMMON | UNIQUE |
|---|---|
| TEST MANAGEMENT SOFTWARE | TEST DEVELOPMENT SOFTWARE |
| Operator interface | Instrument control |
| User management | Stimulus |
| Test flow control | Measurements |
| Limit checking | DUT control |
| Parallel thread management | DSP |
| Results processing | |

TABLE 3

Example Breakdown of Common and Unique Elements of a Tester

To stop rebuilding test systems, you must identify all the common elements and put them into a framework you can reuse across all your testers. Not only is the framework essential to support the common elements, but it must also be flexible enough to support the unique elements, now and in the future.

## Benefits of using a Test Framework

### Increased Code Reusability

After all your test teams have **standardized on a single test framework**, they can more easily share code between them. Sharing works for the common elements as well as the unique elements.

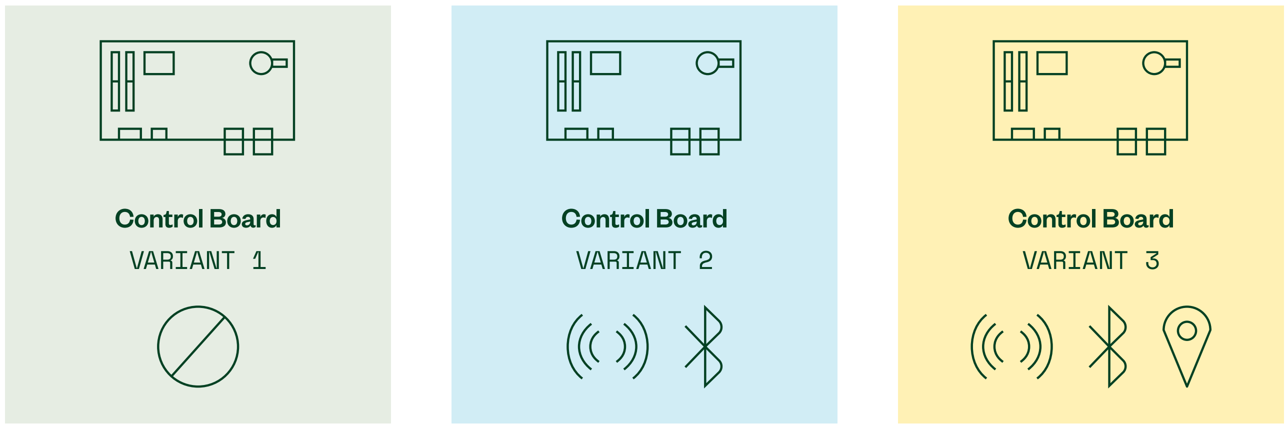


FIGURE 4
Three different variants of a board, with the significant differences being the addition of Bluetooth functionality in the second board and Wi-Fi functionality in the third.

For example, in Figure 4, there are three board variants. While the added functionality in variants two and three might be added years later, the fundamentals of how the board is tested remain the same.

You might develop the same fundamental test system three times if you have a reactive test organization. This approach increases the cost of the upfront tester, and it also affects maintenance costs in the long run.

### Increased Productivity

Productivity is highest when engineers spend time doing what they are most qualified to do. But often, engineers are forced to spend a significant amount of time on other tasks like building user interfaces or generating reports. Developing new testers becomes much faster if you create a common framework for all your testers. All the standard components are already built, freeing the test engineers to focus on building the functionality needed to test the device—which can decrease the time it takes to get new products to market.

### Reduce Maintenance Efforts

The maintenance phase is the longest span of a tester's life. Maintenance includes adding new tests, maintaining compatibility across software or OS upgrades, or fixing detected bugs. Maintenance of a test executive solution even extends to the realm of documentation. Decisions made in the development phase of a tester determine how time-consuming or frustrating the maintenance phase will be.

## Benefits of NI TestStand

One obstacle to building test frameworks is finding someone with the time and qualifications necessary to do so. That's why NI developed **TestStand**, an off-the-shelf and customizable test executive for all your test operations. TestStand provides a common framework for all your testers, streamlining the development of future testers. With all the standard components built, test engineers can focus on creating unique test functionality, reducing churn, and getting products to market faster.

### Programming Language Flexibility

Test engineers are proficient in different languages, and it's essential that they feel empowered to focus on building the steps that get the test done in the language they prefer. TestStand is language agnostic, meaning test organizations can program in various languages, including Python, C/C++, and .NET, and have it all plug into the same framework and work together.

### Risk Reduction

Developing your test management solution is only the beginning of the task. Software is not a one-time cost but an ongoing effort. Software must be maintained to meet the following goals:

- Add functionality as needed
- Fix bugs or defects that are found
- Maintain compatibility with new software
- Keep up and take advantage of new technology advancements

This maintenance effort can often become significant and consume months of development time. On the other hand, ready-to-run test executive packages such as NI TestStand are regularly updated by NI to stay competitive and viable. This policy enables you to take advantage of the latest technologies (and get fixes to bugs and new functionality) at a fraction of the cost. Read our **Test Executive Software–Build or Buy?** guide to learn more.

## Advanced Feature Implementation

The most obvious cost associated with building a custom test executive is the upfront development cost of implementing all the features of the test executive. The following table outlines some of the features associated with test frameworks. The table shows an estimate of the time required to implement each feature from scratch and compares it to the time it would take to implement the same features using NI TestStand.

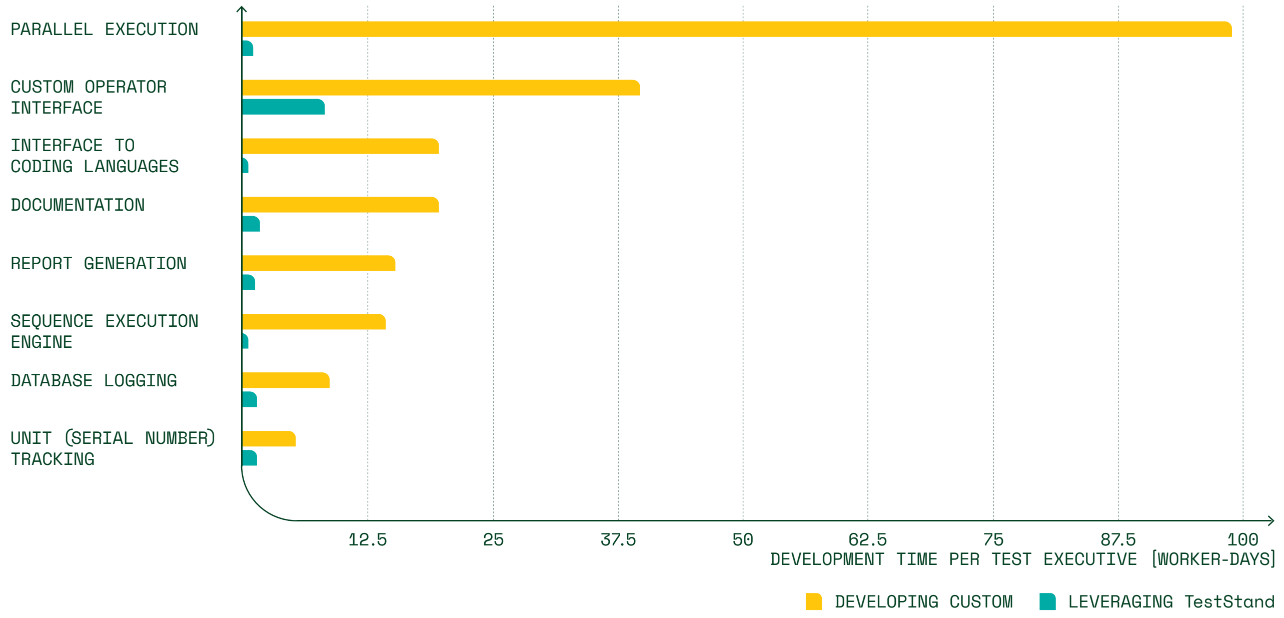


FIGURE 5

NI worked with test professionals to ensure standard features were easy to implement with TestStand. Refer to the build or buy guide for more information.

Not all are relevant to every organization at any given time, but that doesn't mean they won't be relevant in the future.

In this section, we'll discuss some valuable features you can add easily in TestStand—such as parallel execution, report generation, database connectivity, user interfaces, and debug/deployment tools.

### Parallel Execution

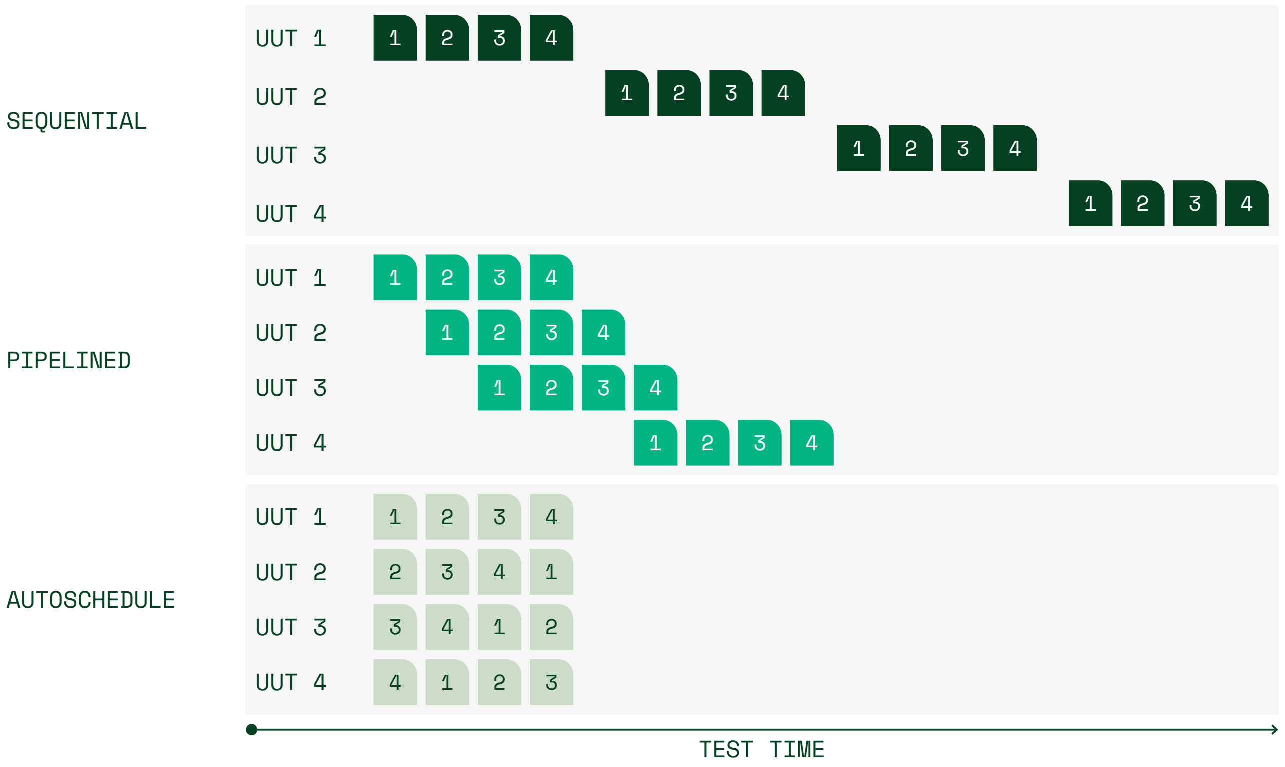


FIGURE 6
This diagram demonstrates TestStand's ability to pipeline sequences and autoschedule resources to enable the testing of multiple UUTs (units under test) at once.

One way to increase test throughput is simultaneously executing tests across different devices. When we look at how much time it takes to develop software that can perform **parallel execution**, it is clear why test organizations avoid creating this feature unless they must. The problem is that parallel execution requires considerable development effort, so when an organization realizes it's needed, it is often too late to implement it. With TestStand, developing parallel systems is far more accessible, eliminating most of this effort and making parallel systems very cost-effective.

### Report Generation and Database Connectivity

When building a test system, it is essential to consider how the test results will be handled. That's why TestStand has built-in features that simplify **database connectivity and report generation**. Customizing reports in TestStand is a common task, and TestStand provides many features to customize the report content, functionality, and style.

Although a test report does a great job of providing a snapshot of the passes and failures of a test run, it isn't convenient when you want to poll historical test results from one or many test stations. Instead, NI TestStand can quickly log test results to almost any open database connectivity (ODBC) system, such as Oracle, SQL Server, or MySQL, out of the box.

### Debug and Deployment

Reduce the time you spend debugging your test code using TestStand's built-in functionality. Take advantage of tools that allow you to monitor test values throughout a sequence and breakpoints to pause execution to further investigate a section of your test code.

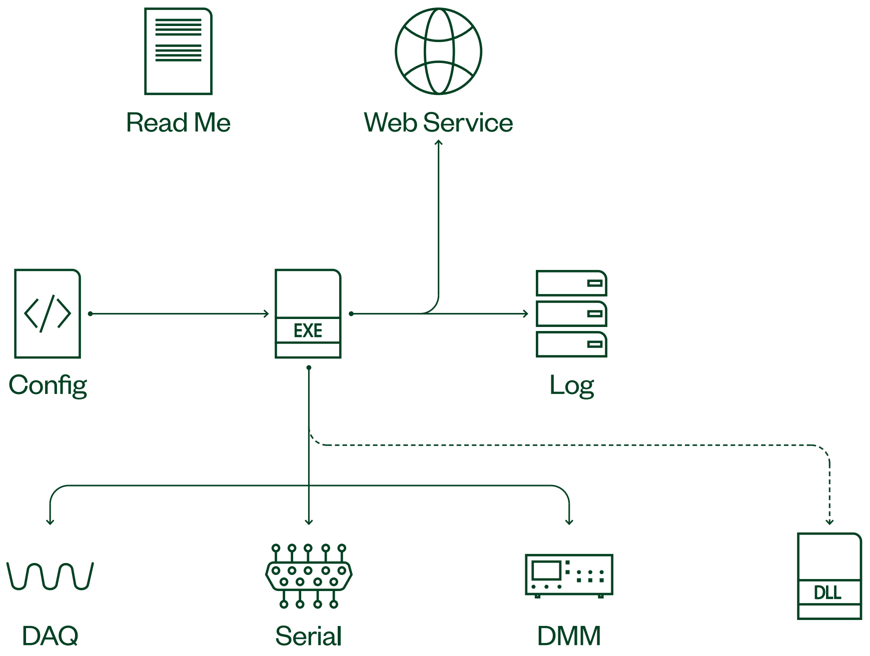


FIGURE 7

Deployment involves packaging all necessary test system components using a deployment tool or server before distributing to the desired test stations.

You can also simplify how you deploy your test code to production machines with utilities that automate building installers and distributions.

## User Interfaces

The operator interface is the display through which the operator interacts with the test system. Developing a custom operator interface can be a nontrivial time investment, especially when usability and consistency are considered. TestStand provides simple operator interfaces to configure and execute tests on deployed systems. You can also develop custom user interfaces in multiple programming environments for greater control over how operators interact with test systems.