8/21/2014

A Table-Driven Approach to Automated Test Systems

Neil Baliga

Verifide Technologies, Inc.

Executive Summary

At its very basic function, a Test Executive runs tests. However, HOW those tests run and WHICH tests to run involve much more than just software - It is an iterative process involving people in various roles and organizations.

For systems engineers, test developers, test operators, and production managers, the process of test definition, execution and selloff is not a fluent one. There exists a chasm between these core test roles that results in a sub-optimal process because it involves translation from one domain to another using different formats.

Bridging that chasm is a manual effort. Systems engineers need to communicate requirements via email to test organizations that then have to implement them in other constructs like test sequences. Furthermore, any changes to the requirements or specifications results in a repetitive process to move this information down to the appropriate personnel at the test organization, which will then have to iterate its implementation of the new modifications.

So what is the real issue here?

Definitions for requirements, specifications, and input parameters are most often table driven. This is natural and intuitive to define things in tables. Engineers define requirements matrices in Microsoft Excel and then do book-keeping and selloff in tables for compliance.

However, despite the process being a naturally table driven one, prominent test executives on the market today only allow definition of sequences, parameters, and specifications in hard structures that are not consumable in a table form.

This is where our table-driven concept comes in.

In this whitepaper we will showcase our table-driven approach to testing that is used in our Enterprise Test Executive - PASS. This approach embraces this naturally table driven process natively in our architecture so that tables are used to define test sequences, parameters, and specifications. The end result goes beyond technical improvements like modularity and scalability, but more importantly yields significant gains in cost metrics due to the efficiency of the test process.

Trends in Test Complexity

Moore's law has been in effect for the past few decades [1]. Devices are getting more complex and demanding higher test complexity and volume. Older concepts in test systems are constantly being stressed beyond their intended design. Test organizations are also finding themselves under increasing pressure to find efficiencies to accommodate the increase in scope of these new devices.

"The array of uncertainties and failure modes itself grows with the system's complexity, and the mechanisms for addressing these potential failure modes add to it with the resultant effect of making overall system complexity grow exponentially. The system's cost follows suit. This is what we term the cost-complexity death spiral" [33]

"A focus on achieving capabilities embodied in requirements while minimizing cost has, under the influence of technical and programmatic uncertainty, led to ever more complex spacecraft with higher and higher cost, a cost-complexity death spiral. Decision makers respond to increased marginal cost by increasing the scale of [devices] to maximize the overall capability/cost quotient, and increasing lifetime to minimize amortized annual costs. Both trends increase capability, which drives further increases in scale and lifetime" [31]

Because of the cost-complexity death spiral, a budget constrained test process has absolutely no room for error or iteration. When a small or medium level modification is inserted into the cycle, an inefficient test process can result in significant increments to cost. For example, if systems engineering modifies how a test should run, this can result in significant rework, especially when testing complex devices that run tens of thousands of tests.

So how is all this test complexity relevant?

In older test systems, automation was implemented purely in the form of code. If the test specified that it needed to run a measurement with 3 steps, then that was directly implemented in code. It the requirements were to run the device in 5 configurations, then that was also in the code.

Incremental progress was made in the industry and tests were written to be modular. This meant that the test was written to take input parameters that defined how to run the test, and the code was re-usable. This took the efficiency of the test process up one order because it mitigated the software modification lifecycle when changes were required.

In-house and commercial test executives were developed to embrace this modularity by allowing users to define parameters and tests in files that then executed those modular tests.

Great! Now everyone had some mechanism to create defined sequences that executed modular tests. The world is a better place.

Now let's examine the main problems with this new normal:

Problem 1: Data Interchange

The process of creating test requirements, input parameter, and specifications is typically performed by Systems Engineering organizations. The job of creating sequences, modular tests, and running tests is mostly performed by Test organizations.

This means requirements on what to run and how to run tests need to be communicated across organizations and role players. In the computer domain, this is usually Excel files translated into a test sequence using a manual process - in other words, an engineer needs to receive the requirements, review it, and then implement sequences. Needless to say, this is labor intensive and can also be error prone.

Additionally, when test requirements change, the same manual process is repeated. Except this time, it is repeated with LESS efficiency not more. The engineer has to process the changes in the requirements, identify the items on his or her end that need to be changed and then make the changes. Compared to the initial sequence setups, this is always more error prone.

Problem 2: Test Volume

As complexity increases, so does the number of tests - and not necessarily linearly. So, in the process where thousands or tens of thousands of tests are ran on devices, the task of setting up sequences becomes extremely labor intensive. The engineer in the test organization has to create steps for each and every required test configuration and combination.

For example, a test system may have gotten by on previous devices with a thousand tests, but new devices now require five thousand tests. This has a direct impact on the test organizations' bottom line because the effort to create and iterate these sequences is inflated accordingly.

Problem 3: Transparency

When tests are ran through sequences, they are typically setup in steps. These steps are typically setup in a sequence file or each step may be backed by a configuration file.

However, this model does not allow the sequence developer or test operator to have visibility into what is being ran. They simply know that a step named "Test4", for example, is running and returned a status. In more sophisticated devices, it is necessary to know for example, which configuration the device is in when running.

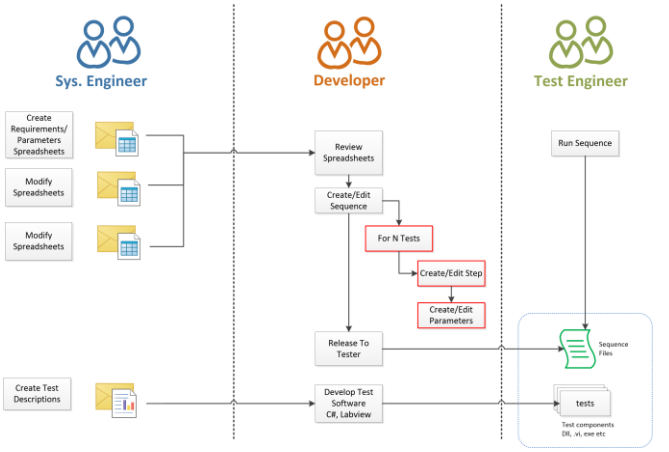
The limit on transparency is not only a disruptor during sequence development, but it also limits the ability for bookkeeping purposes. For example, how does that sequence translate into a summary report that gives the Systems engineer the confidence that the required tests were ran?

Solution

The solution to eliminate these inefficiencies in the production processes by allowing the front-end organizations such as Systems Engineering to independently define, manage, and iterate test requirements and parameters.

This is not easily achieved with existing solutions because Systems Engineers tend to work in spreadsheet and 2 dimensional workspaces whereas test executives tend to deal in multi-dimensional and complex object configurations.

The PASS software takes a Table-Driven approach to the test executive. It uses two dimensional tables to define requirements and parameters, thereby allowing Test Organizations to relinquish that responsibility to the front-end organizations.

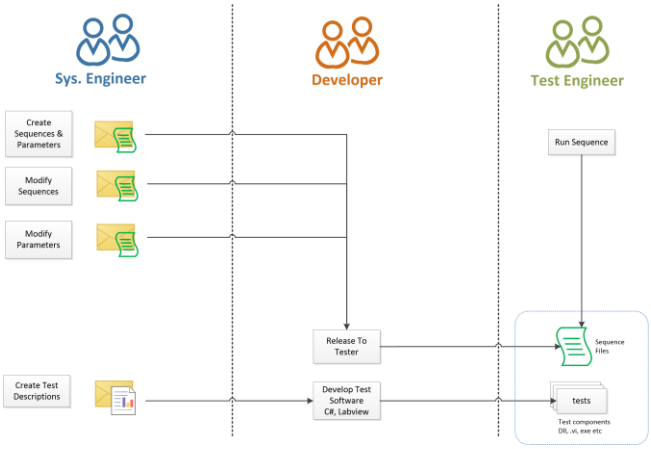


Typical Test Process

Requires translation and implementation of requirements

Costly when process is iterative

Error prone



Test Process using PASS

Requirements and parameters defined by Sys. Engineering

Less error prone

Cost savings are more prominent with higher complexity and higher volume

The end result is:

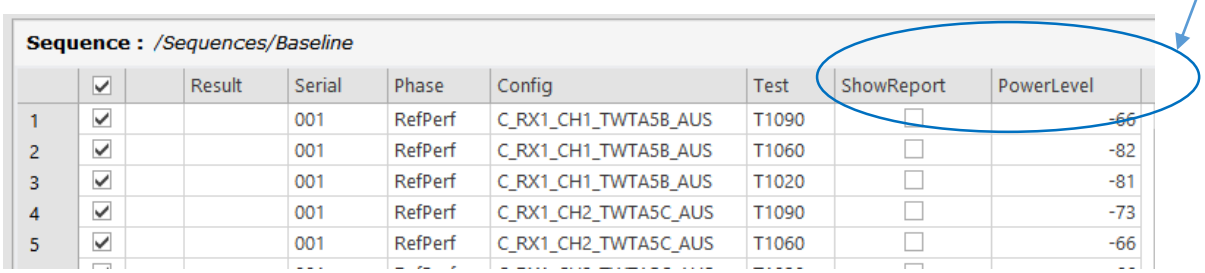
1. Faster workflow
2. Reduced errors in requirements translation
3. Exponential returns with increasing device complexity and test volume

There are TWO points of interest with this solution:

1. **Sequences**

Sequences in PASS are developed in a table structure, which allows the sequence developer to create, edit, and copy/paste requirements in rows and columns. This includes the ability to create, edit, and copy/paste with Microsoft Excel ™

The sequence structure is not completely arbitrary. It requires certain fields to be defined so that a test to run can be identified and executed. It does however, allow the developer to create their own columns in the sequence table that then AUTOMATICALLY become input parameters to the test.



Sequence : */Sequences/Baseline*

			Result	Serial	Phase	Config	Test	ShowReport	PowerLevel
1	☑			001	RefPerf	C_RX1_CH1_TWTA5B_AUS	T1090	☐	-66
2	☑			001	RefPerf	C_RX1_CH1_TWTA5B_AUS	T1060	☐	-82
3	☑			001	RefPerf	C_RX1_CH1_TWTA5B_AUS	T1020	☐	-81
4	☑			001	RefPerf	C_RX1_CH2_TWTA5C_AUS	T1090	☐	-73
5	☑			001	RefPerf	C_RX1_CH2_TWTA5C_AUS	T1060	☐	-66

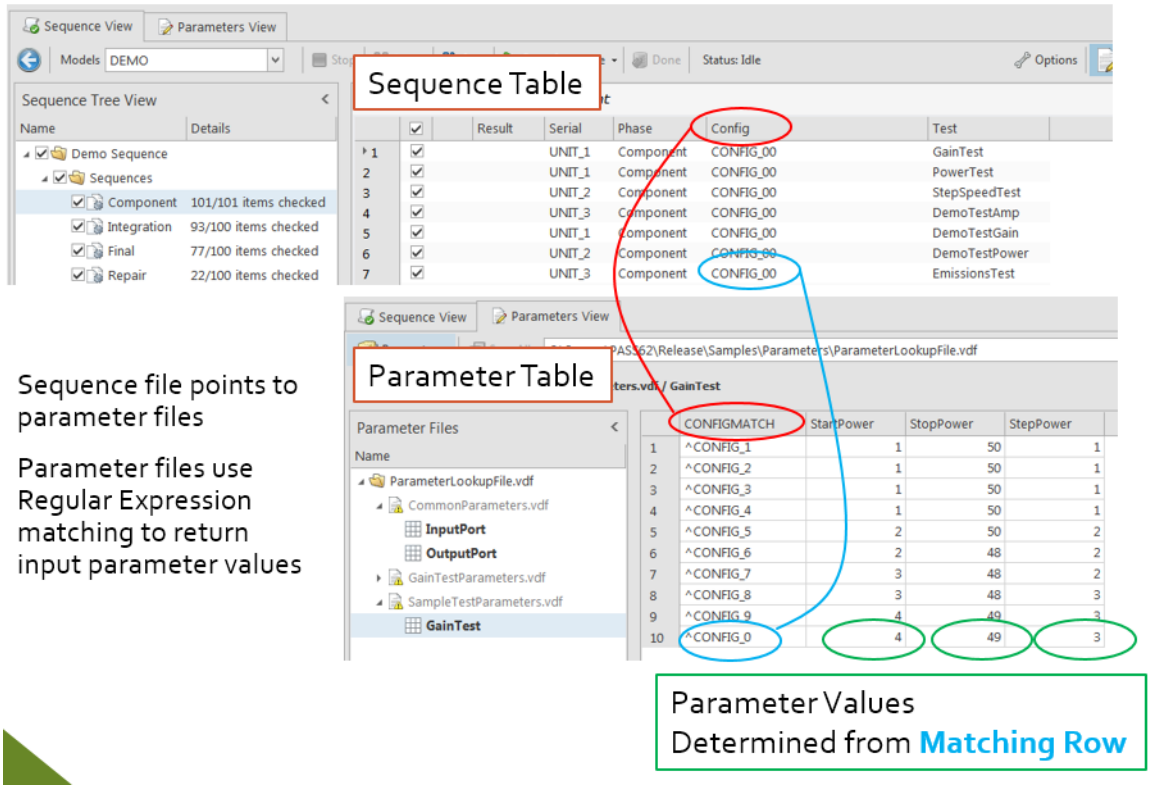
1. **Input Parameters**

The more difficult problem with sequences is the configuration of input parameters. These are the variables that are used to execute modular tests. In this concept, the test is written to accept input variables that are then used to control the flow of the test. Examples of input variables include things like power and voltage settings, test limits etc.

The PASS table-driven concept extends beyond just the sequence table and includes a "parameter-resolution" framework that allows input parameters to ALSO be determined from tables.

This design is a "cascading" design, in that the values from the sequence table determine WHICH parameters are sent as input parameters to the test. This uses an text matching logic which allows the developer to develop structures that make sense to run their modular tests.

For example, you could design the Parameter Table to match on the CONFIG field of the sequence table. You can also add/remove other fields to match on such as PHASE, SERIAL etc that make the most sense to determine the parameters for the test.



The PASS parameter resolution allows multiple parameter tables and files to be defined and configured, which allows the developer to design these structures in the best way possible. This, for example, allows files and tables to be common across tests, or tables to be focused on particular parameter types (eg. Power values, Frequency Values, Limits Values etc)

Conclusion

The primary need for new approaches in test is that devices are getting more complex and test requirements are scaling accordingly. However, older methods of testing in linear steps make it extremely inefficient to handle the new growth in scope of test.

The end result is not only increased cost, but this cost also grows exponentially with device test complexity and test volume.

The typical test organization will make efforts to automate some things to account for the inefficiencies, but that only adds to the amount of software to manage, maintain and configure. Furthermore, it is usually highly bound to a particular test system and often not scalable across product lines or testers.

The PASS Table-Driven approach solves a key pain point of test organizations - How to eliminate steps in the process that are not only time consuming, but also error prone.

This is done by embracing the natural process of requirements and parameters that are table driven in the front-end organizations. By using this approach, systems engineering organizations can better interact with test organizations by providing direct requirements that are executable on the testers. The benefit to systems engineering is that they have the freedom to add/remove requirements without dependencies on other organizations to translate and implement those in sequence steps.

The second benefit of this process is that test organizations eliminate the responsibility to translate requirements and parameters into test steps. This allows them to focus on writing test routines and algorithms and improving overall yield and accuracy.

This software has been in use in major aerospace systems since 2006 and this concept of testing has been shown to improve test modularity, process efficiency, and bottom line cost to test organizations.

(1) http://www.nytimes.com/2014/01/10/science/designing-the-next-wave-of-computer-chips.html
[31] Brown, Eremenko and Collopy (2009) "Value-Centric Design Methodologies for Fractionated Spacecraft: Progress Summary from Phase 1 of the DARPA System F6 Program" American Institute of Aeronautics and Astronautics

[33] Beland, Jonathan (2012) "Satellite Manufacturing Report, January 2012" Futron Corporation, http://futron.com/upload/wysiwyg/Resources/FoF/2012/FutronSM2012-01.pdf