

TECS: A Browser-Based Test Environment For Complex Systems

Kurt A. Richardson
Exploratory Solutions, USA

The Test Environment for Complex Systems (TECS) is a comprehensive browser-based solution designed to support System Engineers, Test Engineers and Program Management through the design, execution and review of the Integration and Test (I&T) programs for complex engineered systems such as satellites. The aim of this paper is to introduce TECS and explore its potential role(s) in the important activity of I&T.

Introduction

The Integration and Testing (I&T) phase of any complex engineered system, such as a satellite, is one of the most challenging phases of system testing. Various components, often built by different manufacturers (using different test standards, processes and documentation), are brought together for the first time and tested as a complete system, rather than a collection of parts. Developing and maintaining the documentation and records to guide the I&T phase is a major undertaking and accounts for a substantial amount of the effort put into I&T. The creation and maintenance of test documentation is primarily the responsibility of Systems Engineers, but these are not the only 'users' of such documentation. For example, Test Engineers and Operators rely on detailed and accurate test documentation to allow them to plan and perform the necessary testing, whereas, Program Managers rely on detailed and accurate records to ensure program requirements have been met and tested thoroughly. Alongside the documentation required to perform a specific test, is the data generated from a particular test, which includes accounts of any anomalies that were observed during testing. Test documentation, test data, and anomaly records (which include both accounts of the anomalies and how they were overcome/solved) form a complex web of detail about the I&T process. As a result of this complexity, it takes considerable time and effort to navigate the 'test space' and understand exactly what the status is of any test program. The Test Environment for Complex Systems (TECS) is a set of tools that have been designed to define, maintain, record, and traverse this 'test space' with ease.

TECS is a comprehensive browser-based solution designed to support Systems Engineers, Test Engineers and Program Management through the design, execution and review of the Integration and Test (I&T) programs for complex engineered systems such as satellites. The aim of this paper is to introduce TECS and

explore its potential role(s) in the important activity of I&T. The test environment provided through TECS can be divided into four major areas (in parentheses are the primary users for each area):

1. Test documentation creation, review and maintenance (System Engineers);
2. Test execution (Test Engineers, Test Conductors, and Test Operators);
3. Test anomaly management (System Engineers), and;
4. Monitoring of the overall testing program (Program Management).

Each of these areas will be discussed in the following sections. But before doing so I would like to comment on the connection between TECS and complexity thinking.

In a sense TECS would seem to be non-complex in that it is really nothing more than a tailored document management system—in many ways the tool itself is very much linear. However, the activity it attempts to support (I&T of complex engineered systems) is undoubtedly complex. In particular it is the changing and revisionary nature of an I&T program that TECS attempts to support. By making the process of creating and reviewing documentation and data as simple and as flexible as possible, it makes it far easier for users embedded in the complex I&T process to change and revise the ongoing record and plan of the process itself, as well to synthesize a current and accurate view of current status. In this sense it is argued that such linear tools can have an important, even central, role to play in managing complex systems if designed and utilized appropriately (see Richardson, 2008, for a discussion of the use of linear tools within a nonlinear modeling culture). The tool itself might be linear, but it is intended for use within a nonlinear system—rocket science needn't be rocket science!

Indeed, there seems to be a general principle involved here: both the understanding and most efficacious utilization of systems imbued with complex dynamics can benefit from a host of very different tools or methods, whether linear or nonlinear, mechanical or dynamical, freeze-framed or processual, functional or algorithmic. Because the specific tool or method may prove helpful, though, doesn't warrant jumping to the conclusion that the properties characterizing that tool or method can be projected wholesale into an adequate explanation of the complex system under study. This is especially pertinent when it comes to linearization techniques used in dealing with nonlinear systems since there is the strong temptation to reduce the system's dynamics to linear functions which, of course, are much easier to handle mathematically.

A related case in point is how fractal generating functions, although they may display graphical and/or statistical properties that may seem remarkable in their capacity for simulating the apparently fractal nature of some natural object, say, the branching of a tree, do not necessarily translate into an adequate explanation of how in fact the natural object itself was generated. Intelligent working

scientists, accordingly, always need to keep in mind that methods and tools do not by themselves explain anything outside of the specific context in which they were found useful, and that these contexts have to be incorporated into theorizing beyond the particular use of the tool.

Furthermore, a good design is a good design whether it emerges naturally in a natural complex system like the mammalian brain or it is intentionally designed into a complex engineered system such as the production of a satellite. That is one reason that complexity science has prompted a burgeoning of research and publications involving applications of naturally-occurring emergent design into intentional design (e.g., see Peng & Gero, 2009) as well as applications of intentional design into the study of organisms (e.g., see Lewens, 2005).

Test Documentation Creation, Review and Maintenance

For every test performed in the I&T process a procedure is needed to direct the test, and for a comprehensive performance test (CPT), for example, these can be very complicated documents. TECS was designed to ease the creation of such documents substantially. Regardless of how much effort goes into preparing a procedure, as soon as testing starts, changes to the procedure will undoubtedly follow as more is learnt about how the spacecraft bus and payload (for example) interact with each other. It is this dynamism that TECS is particular good for capturing and managing.

The TECS Procedure Development Tool (PDT) implements a bottom-up approach to procedure development that helps avoid inconsistencies, errors, and the time required to implement procedural revisions, as well as to develop new procedures or sub-tests. It also provides a clear record of when changes were made, why, and by who, which (among other things) means users can go back to any prior date and reconstruct a procedure exactly how it appeared for a particular past test, i.e., it provides a very robust audit trail (without any additional work). Digital versions of the issue/error documentation that are created when something 'off-procedure' occurs can also be referenced, via the TECS Anomaly Management (AM) tool, making it much easier to trace the development/evolution of a particular test, or an entire procedure. The output of TECS-PDT is HTML-based which means that procedures can be developed and read via conventional browser software (such as Internet Explorer or Firefox) by persons with the appropriate access profiles, and navigated with ease. The browser-based approach means that no software (other than the standard browser that comes with most operating systems) needs to be maintained on users' machines, making TECS distribution a relatively trivial exercise; the whole system can be hosted on one central Windows server (behind a firewall). Although TECS has been written for a Windows-based server, it can be accessed from any OS because of the cross-platform nature of HTML. TECS is written using ASP.NET 4.0, AJAX and LINQ, state of the art programming frameworks with extensive support.

OPERATION MANAGEMENT

In the panes below you can view all available operations for the current program, as well as create new ones, and preview how operations will appear to the test operator.

NOTE: Refreshing the page clears all fields, and so any changes will be lost. There is usually no reason to perform a page refresh or reload (unless it is to deliberately clear all fields).

All Available Operations		ID/Version	Created	Reqs/Data/Img
<input type="button" value="Edit/Copy"/>	<p>Sub Operation Text</p> <p>Command SSR to dump the recorded data. From AstroRT, run script "SSR_Dump_RF.pl" to dump the telemetry listed below, using the following filename convention:</p> <p style="text-align: center;">VCDU_ArchiveX_[TESTNAME]_[MMDDYY]_[HHMM].bin</p> <p>Where X is the VCID. For example,</p> <ul style="list-style-type: none"> • VCDU_Archive8_L-OBS-001_032307_1223.bin, or, • VCDU_Archive8_2OT_041207_0934.bin, or, • VCDU_Archive8_FLAG26_112107_1640.bin <p style="text-align: center;">N.B. SPACES CANNOT BE USED IN ANY FILENAMES</p>	16/1	8/7/2009 3:34:01 PM	
<input type="button" value="Edit/Copy"/>	<p>Move files dumped from the SSR in the previous step to the appropriate folder on the data server (K-drive).</p>	40/1	8/11/2009 11:15:57 AM	
<input type="button" value="Edit/Copy"/>	<p>Confirm with the LAT operator that they can see the latest VCDU_Archive8... file.</p>	41/1	8/11/2009 11:17:18 AM	

Create / Edit an Operation

Preview

Figure 1 A Screenshot of the Operations Management part of TECS-PDT – ‘All Available Operations’.

Operations, Sequences, Tests and Procedures

The basic building blocks of TECS-PDT are operations, or steps. These are individual indivisible steps, such as, “select script XXXXX.pl”. These operations in themselves do not achieve much. These are then grouped into sequences, which do have some functional value. For example, for the Gamma Ray Large Area Space Telescope (GLAST—now Fermi) there were three operations involved in performing a solid state data recorder (SSDR) dump. In the TECS-PDT terminology, these three operations comprise a sequence. Operations and (short) sequences are the most important elements in PDT. From these units, whole tests and entire procedures can be constructed with relative ease. Operations and sequences are subsystem specific, and operations for different subsystems cannot be combined into the same sequence.

Even at the level of sequences there is a lot of commonality. The details of specific sequences tend to vary between different hardware configurations, but these differences are often small. TECS-PDT’s ability to easily create duplicates of operations and sequences and adjust them to specific hardware configurations saves time as well as ensures a common ‘language’ and ‘grammar’ throughout different tests in different hardware configurations. One of the advantages of this bottom-up approach is that changes made to particular operations or sequences will diffuse automatically through other procedural levels. This can also lead to inadvertent changes, and so TECS-PDT contains tools (such as the Operation Mapper) that enable the procedure developer to see how changes made at one level will proliferate through other levels.

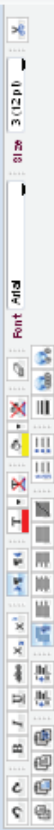
Figure 1 is a screenshot of the Operation Management part of TECS-PDT. When you first open this page you are shown the three tabs: ‘All Available Operations’, ‘Create / Edit an Operation’, and ‘Preview’. The ‘All Available Operations’ tab shows the full list of operations that have already been created for the current Test Program. It should be noted that users can only view this page if they have the appropriate access profile (which is managed via the TECS Security Layer (TECS-SL), and only operations for the current program are shown, which means that no unauthorized user can accidentally see operations created for a different test program. In this example, only three operations have been created, and basic information is shown, such as the operation text, ID/Version, date created, etc. The user can choose to create a new operation from scratch, create a new operation based on an existing operations (which is useful, for example, for when only a script name needs to be changed), or edit an existing operation (editing an existing operation essentially results in the creation of a new operation, i.e., a full version history is maintained). For a program such as GLAST, despite the complexity of the ‘test space’, there were only ~250 distinctly different operations that were used to construct all test documentation. In this case, all the operations can be grouped, filtered, searched against, so that the page is not filled with a list of all operations, and relevant operations can be found efficiently.

OPERATION CODE

Op. ID = 16 : Vers bin = 1

OPERATION TEXT

In the edition below, please enter the operation text that describes all the relevant details to enable the test operator to perform this operation. As data that the operator might need to complete this step is detailed in the next section below.



Command SSR to dump the recorded data. From AstroRT, run script "SSR_Dump_RF.pl" to dump the telemetry listed below, using the following filename convention:

```
VCDU_ArchiveX_[TESTNAME]_[MMDDYY]_[HHMM]_bin
```

Where X is the VCID. For example,

- VCDU_Archive8_L-OBS-001_032307_1223_bin, or;
- VCDU_Archive8_ZOT_041207_0934_bin, or;
- VCDU_Archive8_FLG026_112107_1640_bin

N.B. SPACES CANNOT BE USED IN ANY FILENAMES

**DATA COLLECTION**

This operation requires the test engineer / operator to record data for certain memories for certain memories from the Test. Enter the test name in these can be added here. Enter an mnemonic, a brief description of that memory (e.g., its full name), and then click on the add button.

Once all the relevant memories have been added you can re-order them in the Reorder List below. Memories can also be removed.

Data Mnemonic:

Min Value:

Max Value:

Units:

Units: **UPLOAD IMAGE FILES**

If you would like to include a graphic with this operation, please select the graphics file, and provide a short description below. File types accepted: jpg, bmp, gif, png. The file will be automatically uploaded when you save the create an instance of this operation.

VALIDATION EXPRESSION
 Select a validation template for this operation from the drop-down list below. If a validation template is required, contact your TECS systems administrator.

OPING.PIN ▾

SUB-SYSTEM

Select the associated sub-system for this operation from the drop-down list below. Note that only operations associated with the same sub-system can be combined into a sequence.

Spacecraft Bus ▾

ESTIMATED EXECUTION TIME
 Enter an estimated execution time (in seconds) for this operation. This information will be used to provide a duration times, and can be adjusted as test data is collected.

Execution Time (s): 300

ASSOCIATED DOORS / REQUIREMENTS CODES
 Select the requirements codes, if any, that are associated with this operation is expected to meet (or contribute to meeting).

000-R55532-South State Peconic

COMMENTS
 Add any other comments you have about this operation in the text box below. These comments will not be shown during test time.

COMMENT ARCHIVE
 8/7/2009 3:34:01 PM: Operation first created.

ADDITIONAL COMMENTS

Figure 2 A Screenshot of the Operations Management part of TECS-PDT – ‘Create / Edit an Operation’.

Once the user has decided to either create, duplicate, or modify an operation they move on to the 'Create / Edit an Operation' tab, which is shown in Figure 2. Here is where all the details that define a particular operation are entered. The main 'fields' are:

- *Operation Text* (required): The direct guidance to the test operator. A sophisticated text editor is used to allow the clearest possible guidance to be designed;
- *Data Collection* (optional): A list of the data mnemonic values that need to be recorded as the time the operation is executed;
- *Image* (optional): The filename of the graphic that should appear;
- *Validation Expression* (required): What form of validation is required to allow the test operator to move onto the next operation. In the example shown in Figure 2 the verification template (for which there are a variety of choices), required the test operator to indicate whether the operation was executed successfully (OK), or failed (NG, no go), and then enter his or her unique pin. If NG is selected then the Test Operator will automatically be given the opportunity to create a Test Anomaly Report, or provide a reason as to why, given a failure, the test is to continue;
- *Sub-System* (required): The sub-system for which the operation is designated;
- *Estimated Execution Time* (required): An estimate of how long it takes to execute the operation. This estimate can be revised with real test data. This is very useful when it comes to planning, as it becomes a trivial task to generate time predictions for much longer tests such as a CPT;
- *Requirements Codes* (optional): If the program uses a standard requirements system such as DOORS then requirements codes relevant to the operation can be selected from the requirements database. This enables System Engineers and Program Managers to easily monitor which requirements have been met, which tests contributed to their having been met, etc.;
- *Comments* (optional): Any comments that the user feels are important to record, e.g., if the operation is valid only for a particular hardware configuration. When changes are made to existing operations, there are a number of automated comments that are generated, such as which fields were changed, what the previous operation ID was, when the change was made and by whom. This results in a comprehensive audit trail for each operation. This same trail is created at all levels of the TECS hierarchy.

Once all details have been entered, a preview can be generated (Figure 3) which shows how the operation will appear to the Test Operator at test time. Note that some standard elements are automatically generated, such as navigation buttons, email, anomaly generation, etc. (not all of which are shown in the preview).

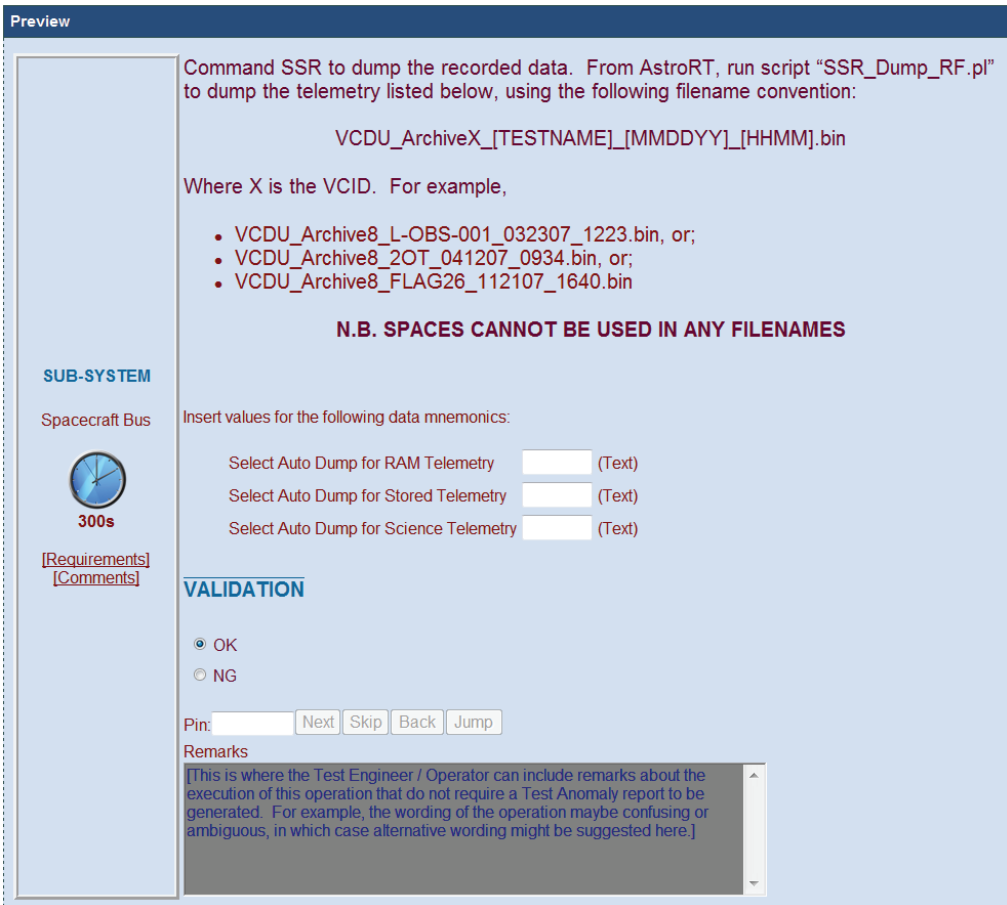


Figure 3 A Screenshot of the Operations Management part of TECS-PDT – 'Preview'.

What cannot be shown in the figures is that each object in the screenshot has hidden 'Intellisense' that pops-up whenever the mouse cursor hovers over that object. This provides the user with built-in instructions on how to use the various tools, further minimizing errors, and encouraging consistent usage.

After a number of fundamental operations have been created and saved, the user can then move onto develop meaningful 'sequences' of operations to achieve a sub-system specific outcome. Figure 4 shows the Sequence Management page from TECS-PDT. Here, existing operations and sequences are combined to produce new sequences. As with the Operation Management page, sequences can be created, duplicated, or modified, and a full auditable record is generated along the way. A sequence may be defined to, for example, detail how the command window is passed from one sub-system team to another, or how a solid state recorder dump is performed. It is likely that such sequences could be reused with a variety of hardware configurations without change. There are no hard and fast rules about how long or short a sequence should be. Through continued use the experienced Systems Engineer will figure out what

the appropriate compartmentalization should be, and this will undoubtedly evolve as the test program is developed. Here we have a case where a fairly linear tool can be employed to develop heuristics of a nonlinear system, perhaps one of the most important advantages of linearization. But again, heuristics are, by definition, adaptive in the sense they are expected to be modified as more is learned about the system.

On the top left of Figure 4 existing operations and sequences (in the example given there are no existing sequences) can be added to the list on the right hand side. These entries can be moved up or down and deleted as needs by. As operations and sequences are added to the new sequence the estimated execution time is updated automatically (by simply adding the execution time of the component operations, plus a 10% overhead). Also, any requirements that are associated with the comprising operations are also listed. New requirements can be added to the new sequence for scenarios when the completion of a sequence of operations is needed to demonstrate that a requirement has been met, rather than just single operations.

As mentioned above, operations from different sub-systems cannot be mixed in the composition of sequences. The sub-system for a particular sequence will be the same as the first operation in the sequence, and the editor will automatically prevent the user from mixing operations and sequences from different sub-systems.

Once a sequence has been defined, the user can preview it. The preview is basically a concatenation of screenshots like the one shown in Figure 3. This enables the user to step through the sequence as it will appear to the Test Operator.

In reality, there will be far fewer sequences than operations, and more often than not the relatively fewer sequences will be used to build combinations higher up the TECS hierarchy, such as tests and procedures. For emphasis, the real test detail is captured at the operation level. Sequences are really no more than a collection of operation and sequence codes, so although the definition of an operation can be quite involved, a sequence might be expressed as simply as, for example, op_16, op_40, op_41, sq_3. What the TECS-PDT various editors do is introduce a bottom-up structure for test procedure development, and a simple scripting language to build more complex test documentation from low level elements.

The next level in the TECS-PDT hierarchy is the test level, where full self-contained tests can be created by bringing together various operations, sequences, and even other tests. An example of a 'test' is the full functional test of one particular hardware configuration, or perhaps a flight software upload test, which would include the upload itself as one test, and the subsequent regression testing as a separate test or set of tests. Fine tuning a test so that it is valid for a range of hardware configurations is a very simple matter. The Test Management editor is

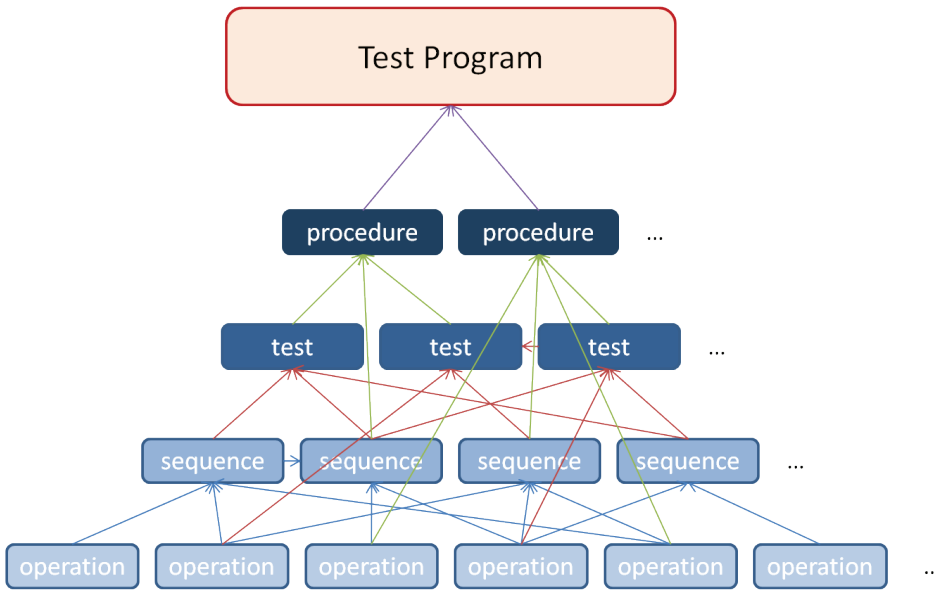


Figure 5 A simple representation of how the hierarchy of operation, sequences, tests and procedures are combined to define a Test Program.

very much similar to the one shown for the Sequence Management page shown in Figure 4, but of course existing tests is included in the list of components that can be added to define a new test.

The highest level is the procedural level which would contain various tests, sequences, and operations, and even other procedures. Figure 5 gives a simple representation of how operation, sequences, tests and procedures are combined to define a Test Program. A CPT is a good example of a full procedure containing a variety of tests to be performed in a variety of hardware configurations. During GLAST the procedure-level was also used to define the testing to be performed during a particular testing session. As such, there were instances where the test procedure for the session was to perform a particular ‘test’ (in the language of TECS) and not a set of tests. In these cases, the procedure definition may be as simple as “ts_4”, which would simply mean perform test 4. The point is that the procedure would still need to be defined even if it contained only one existing test. The reason for this is that it is at the level of procedure that Program Management, Lead Engineers and Quality Control Officers are required to sign-off to allow any sessions’ testing to go ahead.

The hierarchy of the TECS is engineered-in, so to speak, but this same designed hierarchy gives inklings of how nature typically employs “self-organizationally” hierarchical design because of the latter’s many expediciencies.

TECS & Authentication

At each level the test designer can state who has the authority and responsibility to sign-off on the execution of each test phase. An individual with the correct role and designation must be logged in and authenticated before s/he is authorized by TECS to sign-off on the successful execution of an operation, sequence, test, or procedure. If say a Program Manager, a Systems Engineer, and a Quality Engineer are specified as having the authority to sign-off on a particular procedure for example, then that procedure won't be recorded as complete until those individuals log in and digitally 'sign' the document. All authorized personnel are maintained in a TECS program-specific roster. This provides an effective and simple way to monitor I&T progress and identify which tasks are complete and which ones need further investigation.

TECS & Requirements

As has already been discussed briefly above, during the test program design process, requirements (such as DOORS codes) can be associated with the successful completion of individual operations, sequences, tests and procedures, and even entire test programs. With such data made available, combined with as-run test results (also maintained in the TECS central SQL database), it becomes a trivial matter to see which requirements have been met at any stage during the test program. Furthermore, given the database centered approach adopted by TECS, generating output to be imported into other management systems is a simple case of defining the appropriate search query, and the output template.

TECS & Flexibility

The toolset developed in TECS-PDT allows for the creation of robust and consistent procedures via a simple browser-based graphical user interface (GUI). Another valuable attribute becomes apparent during testing itself. CPT dry runs, for example, rarely run smoothly and often procedural changes are needed quickly to allow testing to continue in a smooth and timely manner. Using the database of operations, sequences, etc. that is created as part of the procedure development process, 'new' procedures and tests can be rapidly prototyped, reviewed and run. For example, for the last CPT the Large Area space Telescope (LAT, the primary instrument onboard Fermi) team put together a 300+ step procedure that hadn't been run before. The resulting procedure required only minor changes on review as it was constructed from operations and sequences that had already be reviewed and executed a number of times in previous tests.

An important part of this flexibility is the quality assurance (QA) process. QA is essential if testing is to meet requirements efficiently without damaging hardware, and/or wasting test time. Often the review of test documentation occurs at the procedural level. As full procedures can be long complex documents, this review process can be time consuming, and not always thorough. Who can hon-

estly say that they have reviewed a 500-page test procedure thoroughly from beginning to end? With TECS-PDT the QA process can occur at a lower level, e.g., if a certain configuration independent sequence has been QAed, then it can be reused in other procedures, tests and sequences without revisiting the review process. When larger tests or procedures are created in the dynamic pre-CPT environment, for example, then the review/QA process need only be concerned with how the various sequences and tests interact with each other, rather than revisiting every individual operation. This makes the QA process far more efficient, without losing its effectiveness (the document development process basically becomes self-QAed). Whenever a procedure, test, sequence or operation is opened in the TECS-PDT environment a full history for that element is provided which represents a complete audit trail. Integration and testing generates an enormous amount of paperwork and having a tool to easily and successfully navigate this 'document space' is a genuine asset.

Once a procedure or test is ready for release and execution, TECS-PDT generates an instance of that procedure to be run separate from the current version stored in the TECS database. This way all the data for a particular test can be captured and isolated from other tests, to facilitate further analysis and the ability to easily access the details of tests perhaps performed months or years in the past. A paper-based version can also be easily generated, and various user-defined templates designed and utilized.

Test Execution

During actual execution of a test, usually performed by a Test Conductor accompanied by a number of Test Operators, the operators (who must first log in and be authenticated) are presented with a simple GUI (via test console's browser) showing details of the test about to be performed, and some high level details about the test procedure. Once the required sign-offs are confirmed (this is easily incorporated by simply defining an operation that requires a number of sign-offs and placing this operation at the beginning of the procedure) the test proceeds by displaying one operation at a time. From this display the operator can confirm that the operation has been performed, create a test anomaly case in the event that the execution of the operation fails (via TECS-AM), move onto the next operation, access roster details, send an email directly from the test console to anyone in the program roster, and move to the next step. Whenever the test operator interacts with TECS an internal log is updated. The operator must sign-off on an operation before being permitted to move onto the next operation in the procedure.

The test team can also submit change requests for any part of the procedure. These are often not associated with a test anomaly, but are simpler changes that might reduce ambiguity or improve the flow of the procedure. Such effective communication between Systems Engineers and Test Engineers is another essential dimension in the effective execution of an I&T program.

Where TECS Fits In

In the case of test documentation itself, TECS is designed to replace the paper-based documentation systems of the many engineering firms that comprise the aerospace industry, and provide a standard way to generate such documentation (which alone would make for a smoother more efficient and effective I&T process, especially where teams coming from different companies are working together). In the simplest terms, TECS is a flexible document management system specifically designed for the I&T phase of manufacturing of complex engineering products, such as satellite systems.

A typical test set-up for a multiple instrument satellite system, for example, would be for separate test consoles (comprising separate computer systems and monitors) for each of the instrument teams. A Test Conductor would wield overall control of how the test proceeds, but execution of instrument-specific aspects of the test is delegated to the respective instrument teams. A bound paper version of the procedure to be performed is passed from team to team as their turn is taken, or copies are available to each team. During actual testing, this paper-based procedure is replaced with a browser-based system that can simply be realized as another window on the instrument team's console. TECS does not communicate directly with the hardware (which is done through a Test Executive), but simply provides a digital representation of the procedure, along with digital recording mechanisms for collecting certain test data. Although anyone can log in and view the status of a particular test at any point, only the individual or role associated with each operation is able to make changes. This is achieved via the two viewing modes:

- **Passive view:** This mode allows any authenticated user to step through any procedure, but without the ability to make any changes. Once a procedure has been run, only passive view is available;
- **Active view:** This mode allows an authenticated user who is also listed as an authorized sign-off agent for the current operation to update and sign-off on the current operation. The view only becomes active if the individual is signed-on, and all previous operations have been completed and signed-off.

TECS Anomaly Management

In the real world testing does not always go to plan, and integrated systems do not always behave as simply the sum of their respective parts. Issues emerge at the integrated level that were not necessarily foreseen at the sub-system level. This fact, in combination with the fact that documentation can often contain errors, and Test Operators sometimes make mistakes, means that problems that might occur during testing, generally do occur during testing, especially early in the I&T process as Engineers learn more about how the new integrated system behaves. As Test Operators execute particular operations, TECS provides

tools for them to report anomalous behavior via a Test Anomaly Report (TAR). If an operation fails for whatever reason, TECS automatically requires that a TAR is completed. Similarly to how operations are designed and maintained, TECS provides a TAR Management System (TMS) that keeps a full record of each TAR (including the test session and operation that was being undertaken at the time of the observed anomalous, or 'off-procedure', event), allows Engineers to add further documentation, evidence, comments, resolution, etc. to the original record, and, importantly, provides a status of each TAR as 'open' or 'closed'. A list of required sign-offs can be associated with each TAR, which determines who are the responsible agents for evaluating the closure resolution. At any time, a full account of any TAR can be viewed, as well as the status of all TARs for a particular procedure or program. Although an as-run procedure may be considered 'completed', even when there exist TARs associated with that procedure, a Program cannot even begin to be considered completed before all TARs are satisfactorily resolved. Whenever the resolution of a TAR results in a change to the test documentation, the ID of the relevant TAR is captured in the change history of the operation, sequence, test or procedure that was amended. This aspect of TECS again contributes to the comprehensive audit trail that using TECS for I&T results in; a trail that can be readily summarized, searched, and critiqued at any point during the I&T program.

I&T Program Monitoring

The benefits of the digitization process that TECS brings to test procedure development, maintenance and execution is that a very detailed audit trail is created, so essential to test robustness and quality. As such Program Management can remotely log into the system and easily generate status reports that detail how a particular test is proceeding, how long any part of the test program takes to execute (which can also be used to generate predictions regarding test execution durations), the progress of the entire test program, details of which design requirements the testing has already verified, overviews of the number and nature of outstanding test/design anomalies, etc. There exist many opportunities for generating pertinent reports to facilitate Program Management planning and budgeting as a direct result of digitizing a major proportion of the integration and test process. Although TECS provides a number of standard reporting tools, it is a very simple manner to add further reporting capabilities derived from the captured test data, and even extend the database fields when gaps in the collected data are identified.

Summary

TECS has been designed to facilitate the robust development, maintenance, review, and execution of testing procedures for complex engineered systems. A cut-down development version (desktop focused rather than browser based) was used with considerable effect by the LAT team during ob-

servatory I&T for NASA's recently launched Fermi telescope. For LAT, as an example, although there were hundreds of pages of test procedure, there were only around 250 distinctly different test operations, many of these being variations on only a few themes. TECS, then, was able to reuse many operations, sequences, and tests (with perhaps only minor adjustments) during all phases of the test program. It is asserted that such a bottom-up structured approach to I&T can lead to significant time and, therefore, cost savings for any complex I&T program. As a result engineers can spend more time focusing on engineering issues, rather than document management, which would undoubtedly result in higher quality final products and, in the case of satellite design and construction, reduced on-orbit failures and errors.

Acknowledgement

I'd like to thank Jeffrey A. Goldstein for insightful additions to my original manuscript in relation to clarifying the ongoing relevance of linear systems in understanding and guiding nonlinear systems.

References

- Lewens, T. (2005). *Organisms and Artifacts: Design in Nature and Elsewhere*, ISBN [9780262621991](#).
- Peng, W. and Gero, J. (2009). *A Design Interaction Tool that Adapts*, ISBN [9783639135893](#).
- Richardson, K.A. (2008). "On the limits of bottom-up computer simulation: Towards a nonlinear modeling culture," in L. Dennard, K.A. Richardson and G. Morçöl (eds.), *Complexity and Policy Analysis: Tools and Methods for Designing Robust Policies in a Complex World*, ISBN [9780981703220](#), pp. 37-53.

Kurt A. Richardson, PhD is currently the owner of Emergent Publications—a small publishing house specializing in the field of complex systems thinking; the owner and chief technical offer for Exploratory Solutions—a startup company focusing on developing hardware and software for a range of high tech companies (including NASA and General Dynamics); the lead software developer for Inergy Systems—a new startup company developing software and hardware for household and commercial smart grid solutions; and is also the lead FPGA (field programmable gate arrays) engineer for Orbital Network Engineering, who builds satellite telemetry systems for the likes of Raytheon and Lockheed Martin. His research focus is on the development of fundamental theory for complex systems, and their epistemological implications. This work has lead to 15 books, and more than 40 articles on complexity. On the commercial front his interests lie in applying principles of complexity thinking to both software and hardware design in a wide range of fields from publishing technology to smart grid products.