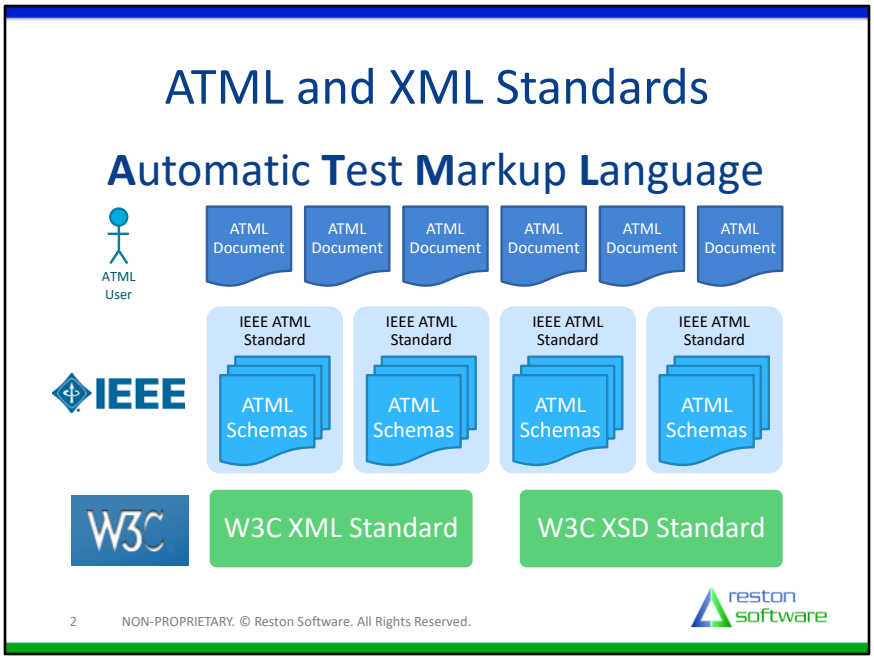
ATML Pad is a development environment for test descriptions, using the ATML Test Description standard format

This presentation discusses the use of standards and software tools in model-based development of test programs for Automatic Test Equipment (ATE)



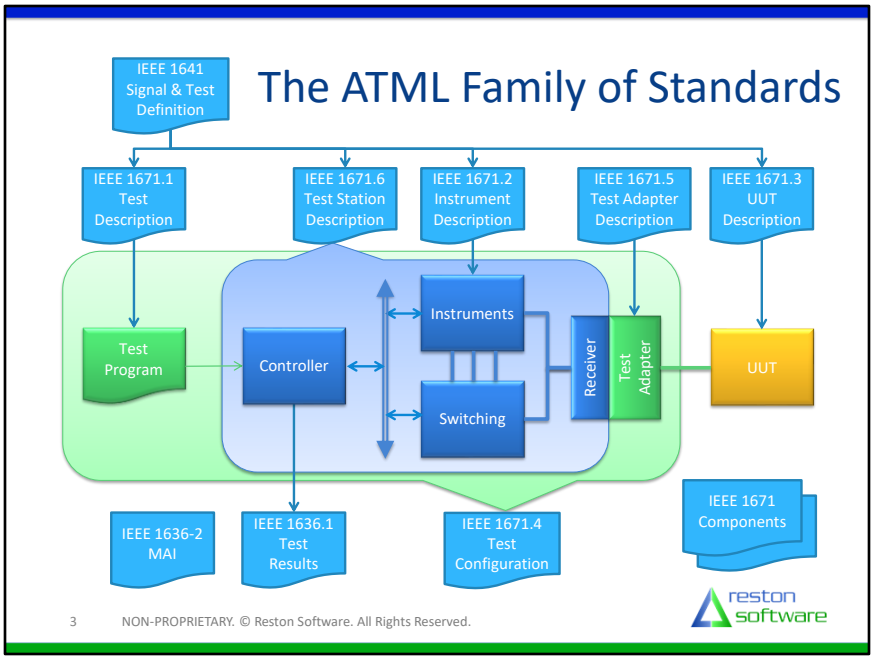
ATML is a family of data exchange formats based on **XML technology**

ATML is **standardized by the IEEE** as IEEE 1671. The "ATML Family of standards" also includes IEEE 1641 and IEEE 1636

- The **XML Schemas** identify the allowed elements, attributes, and their relationships
- The **Standards** define the elements and attributes, and identify additional relationships

User applications exchange **ATML documents** (XML instance document that conforms to the ATML schemas)

ATML documents can be **validated** automatically for conformance with the ATML standards and their ATML schemas.



The slide shows the mapping of schemas from various ATML component standards and related standards to Automatic Test System (ATS) hardware or software items.

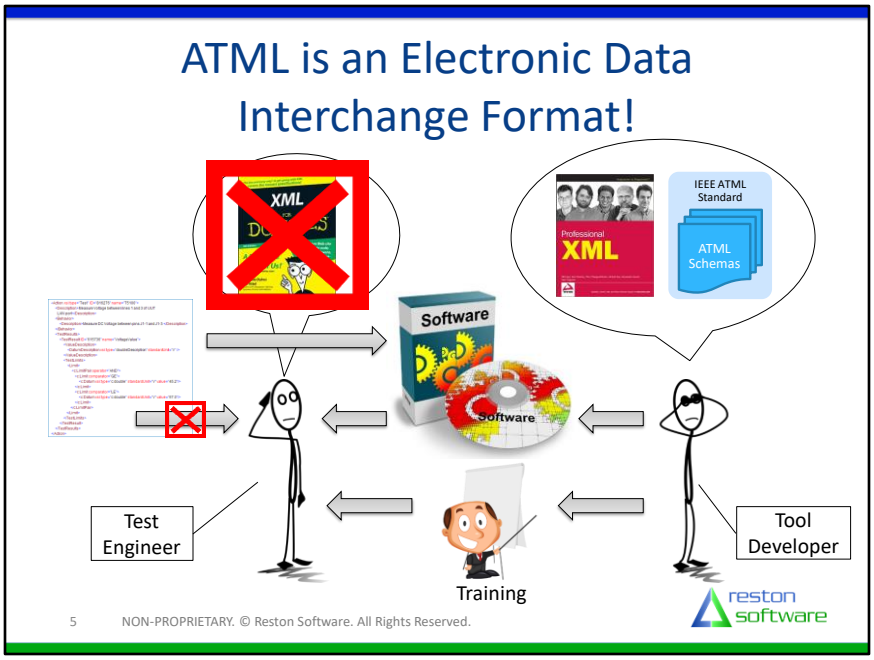
ATML Test Description Example

```
<td:Test ID="test3" name="V_O AC Voltage Test">
      <td:Behavior>
            <td:Description>With power on and input signal present,
            measure AC voltage at the output.</td:Description>
      </td:Behavior>
      <td:Outcomes>
            <td:Outcome ID="t3o1" value="Passed"/>
            <td:Outcome ID="t3o2" value="Failed" qualifier="Low"/>
      </td:Outcomes>
      <td:TestResults>
            <td:TestResult ID="t3tr1" name="V_O AC Voltage">
                  <td:ValueDescription>
                        <td:DatumDescription xsi:type="td:doubleDescription"
                              standardUnit="mV" unitQualifier="pk_pk">
                              <td:NominalValue value="44.7"/>
                        </td:DatumDescription>
                  </td:ValueDescription>
                  <td:TestLimits>
                        <td:Limit>
                              <c:SingleLimit comparator="GE">
                                    <c:Datum xsi:type="c:double"
                                    standardUnit="mV" unitQualifier="pk_pk"
                                    value="40.0"/>
                              </c:SingleLimit>
                        </td:Limit>
                  </td:TestLimits>
            </td:TestResult>
      </td:TestResults>
</td:Test>
```


reston software

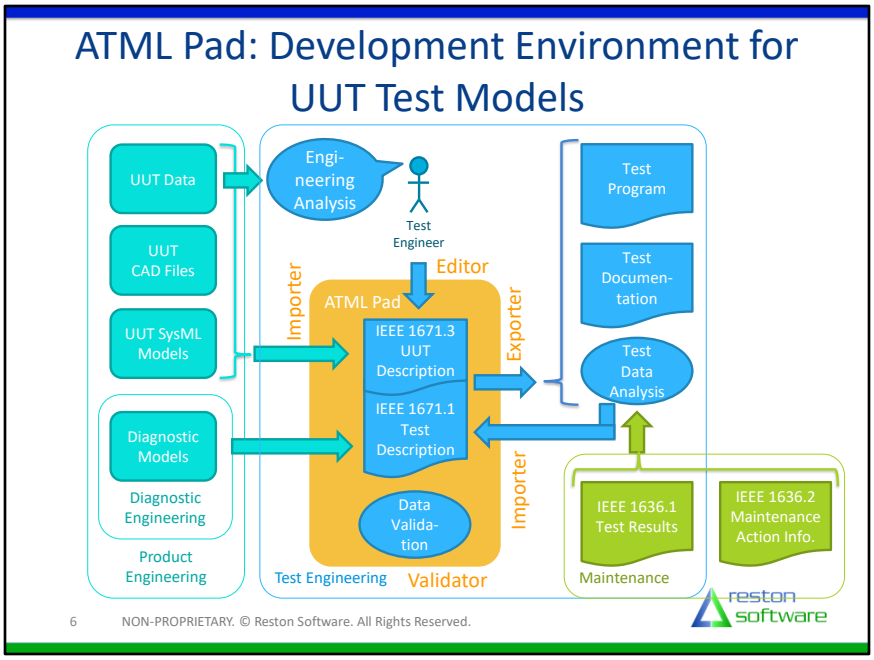
ATML documents are typically text files that contain XML text.

XML consists of elements organized hierarchically and attributes assigned to elements. The elements and attributes have values with diverse data types (string, integer, floating-point, …).



ATML documents are not designed to be handled manually. They are **produced and consumed by computer programs**. IEEE Std 1671 defines an **ATML software tool** as "a program or application used to create, debug, or maintain ATML conformant XML instance documents".

- The end users of ATML-compliant systems (product engineers, test engineers) can focus on describing the targeted ATS entity, without being concerned with XML concepts such as elements, attributes, type inheritance, or constraints
- The developers and integrators of ATML-compliant computer programs can look at XML data directly, for integration and troubleshooting.

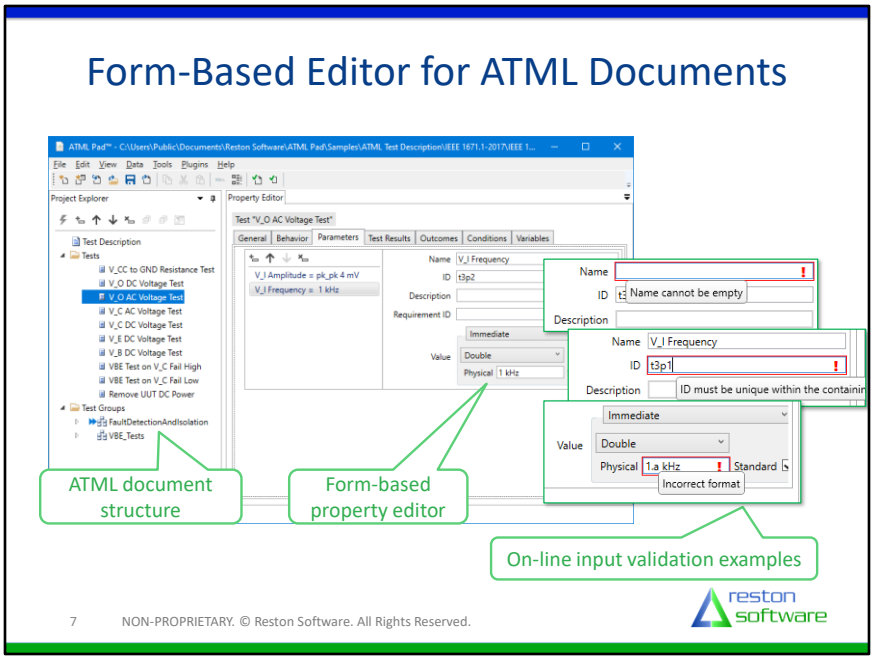


Use cases

- Create UUT and Test Description through test engineering analysis, from UUT data and
- Import UUT & Test data (limited)
- Validate UUT and Test Description
- Generate test program (automatic code generation, resource allocation, switch path calculation, …)
- Generate test program documentation
- Input to test data analysis (ex. for test & diagnostic improvements)

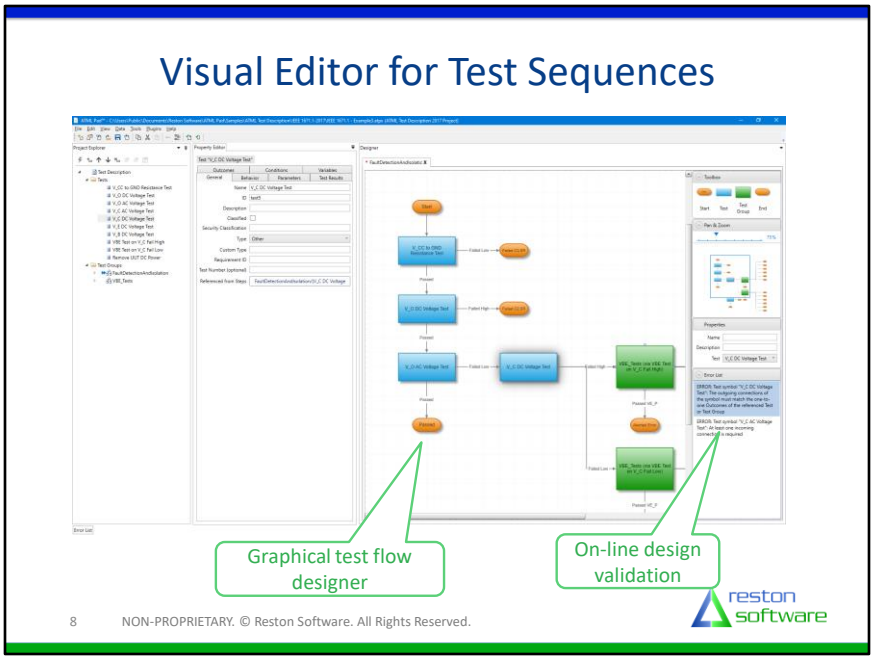
ATML Pad is:

- Editor
- Validator
- Data Converter (Import & Export)



Standard window layout similar to Windows Explorer:
1. The top-level elements of the ATML document (Tests, Test Groups) are displayed and can be edited in tree format, in the Project Explorer (left).
2. The properties of the selected element are displayed and can be edited in the Property Editor (right).

Incorrect data input is identified through a red border (see examples). The error is displayed in the mouse tooltip.

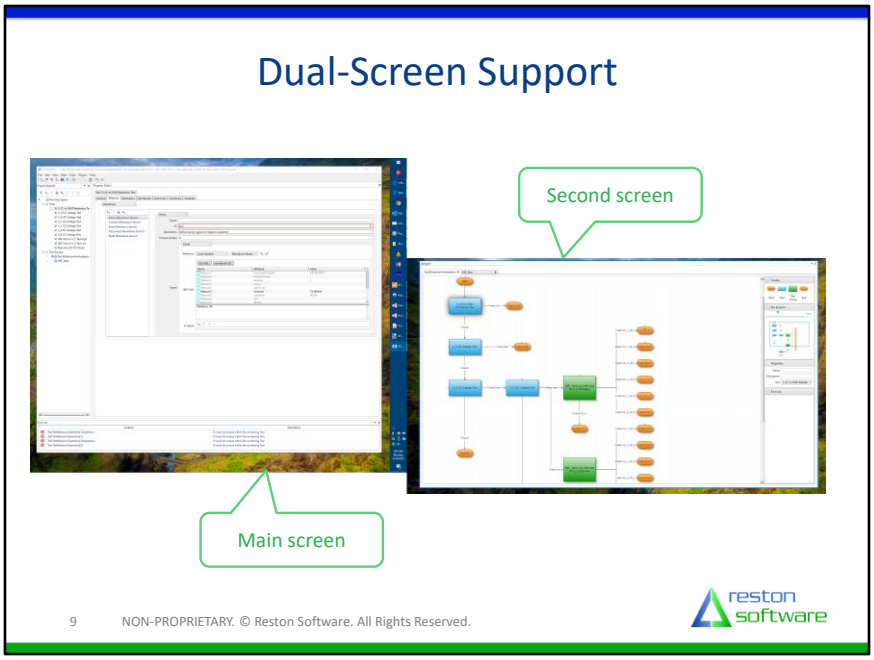


Test sequencing within a test group can be edited graphically, in flowchart form (optional feature).

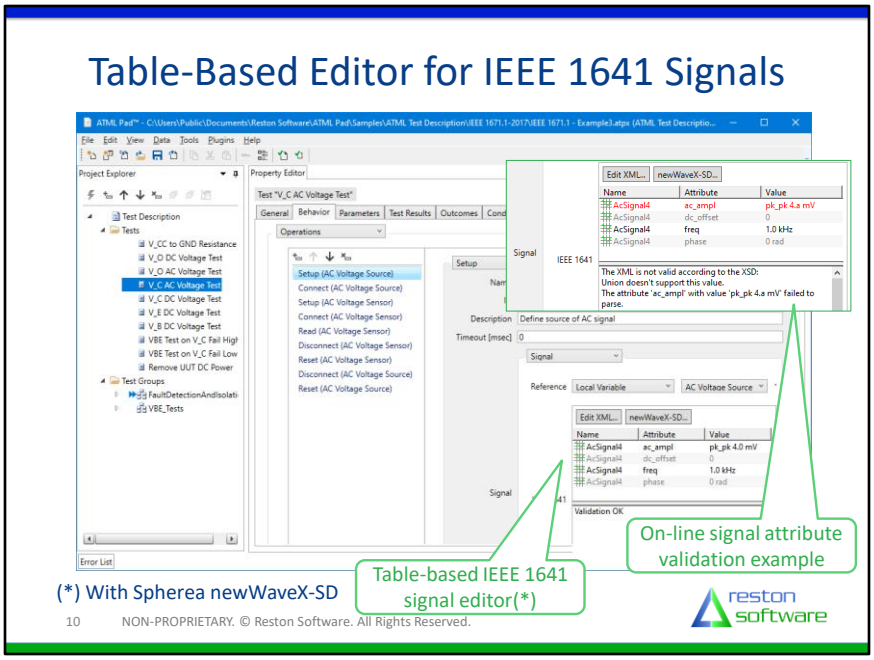
Standard operation, similar to commonly-used drawing tools:

- Drag & drop color-coded symbols from toolbox
- Pan & zoom controls

Missing items, missing connections, and other design problems are displayed dynamically in the error list. This feature guides beginner users through the design process, helping them understand and apply the design rules.



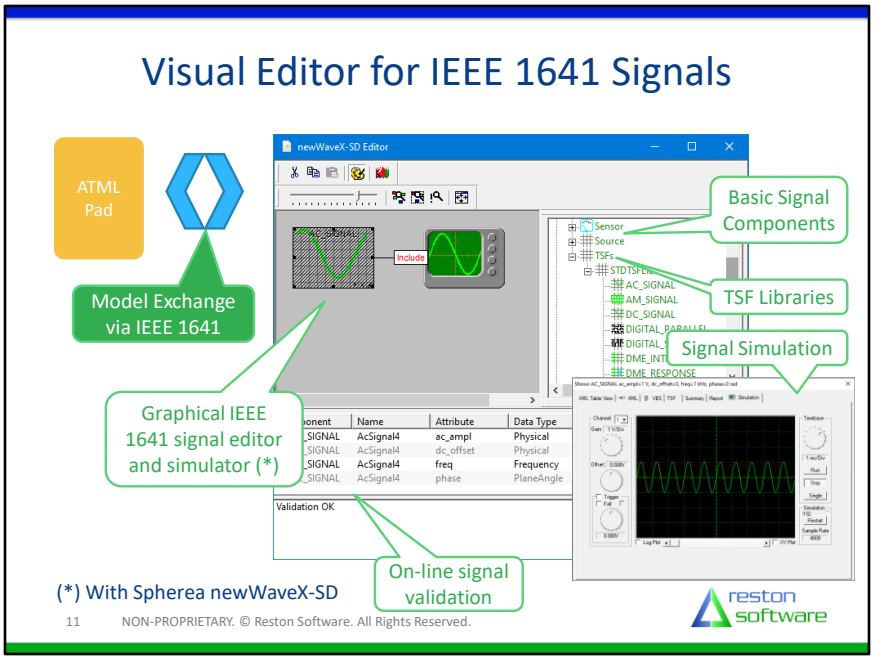
Dual screen support: the graphical editor pane can be undocked and moved to a separate screen.



The IEEE 1641 signal definitions embedded in ATML documents can be edited in a table. The values of signal attributes are displayed and can be changed.

Incorrect data input is identified through red text (see examples). The error is displayed in the message area of the editor.

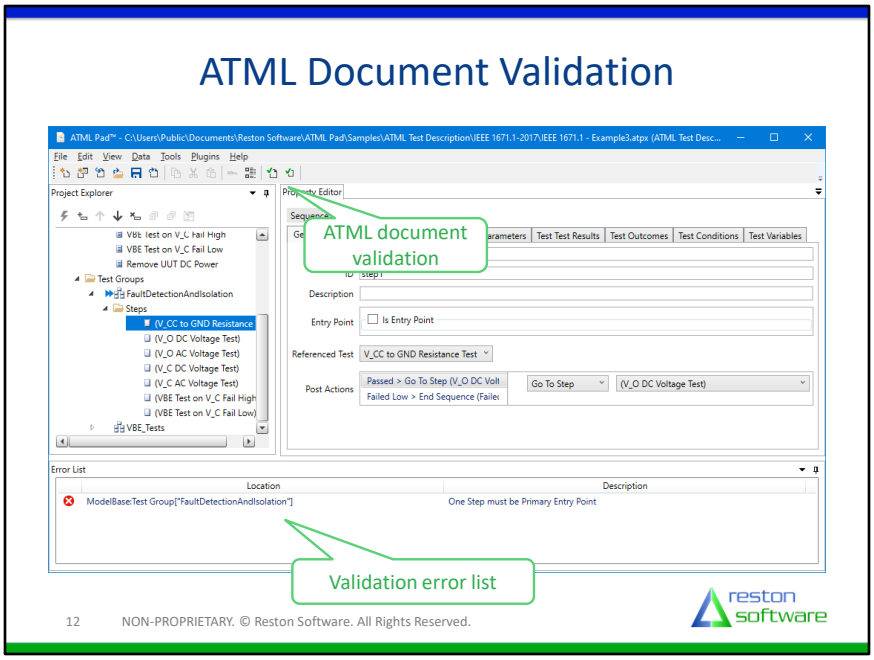
Note: For this feature to be available, a licensed installation of newWaveX-SD is required on the computer where ATML Pad runs.



newWaveX-SD (Signal Development) is a graphical design environment for signal-based test & measurement developed by Spherea Technology. It provides the facilities to design, build and simulate test signals prior to their inclusion in a test program.

newWaveX-SD is integrated natively within **ATML Pad** through the standard format.

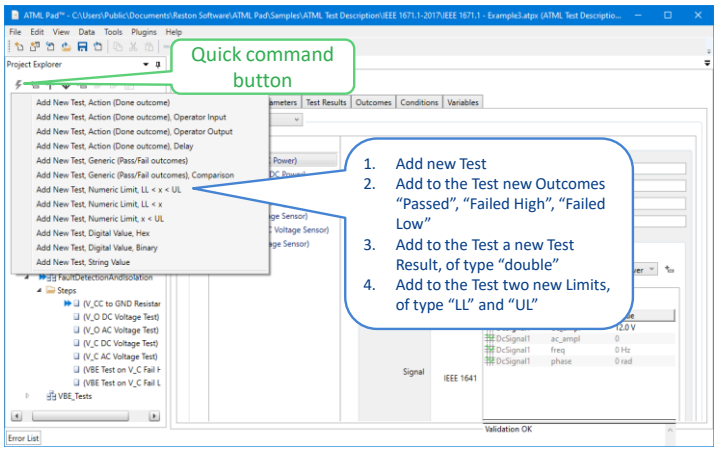
- Signal definitions can be viewed and edited in a pop-up windows that displays the **newWaveX-SD** signal editor.
- The editor is used to configure signals, create new signal definitions, and simulate signals.



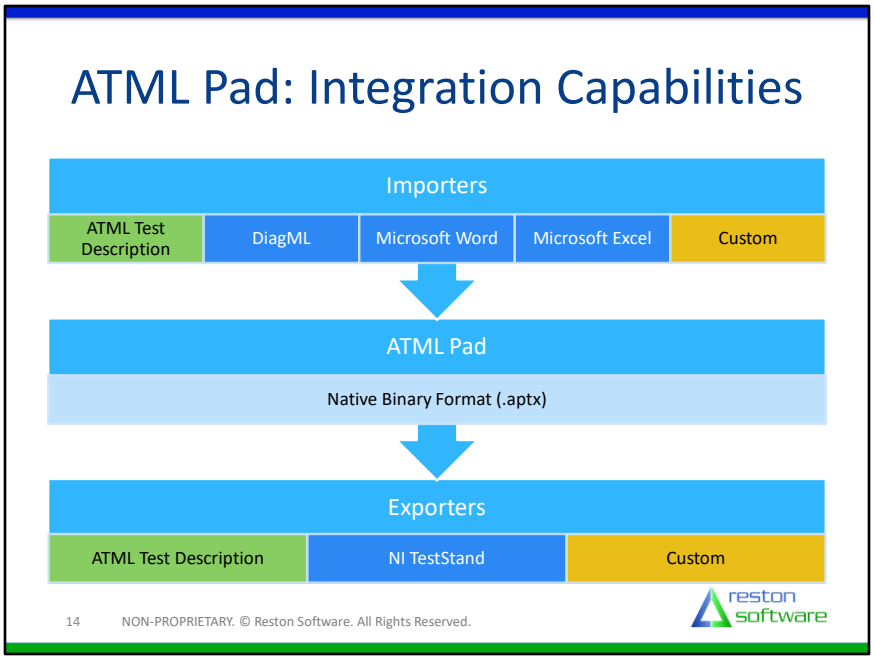
On-demand validation is available in two forms:
1. Document validation: validates the entire document against the ATML schemas. Detects all data input errors normally detected by on-line validation, plus additional structural errors of the document (see example).
2. Element validation: validates the selected element only. This command is much faster, which can be useful during development.

Note: Full document validation is always required before data export.

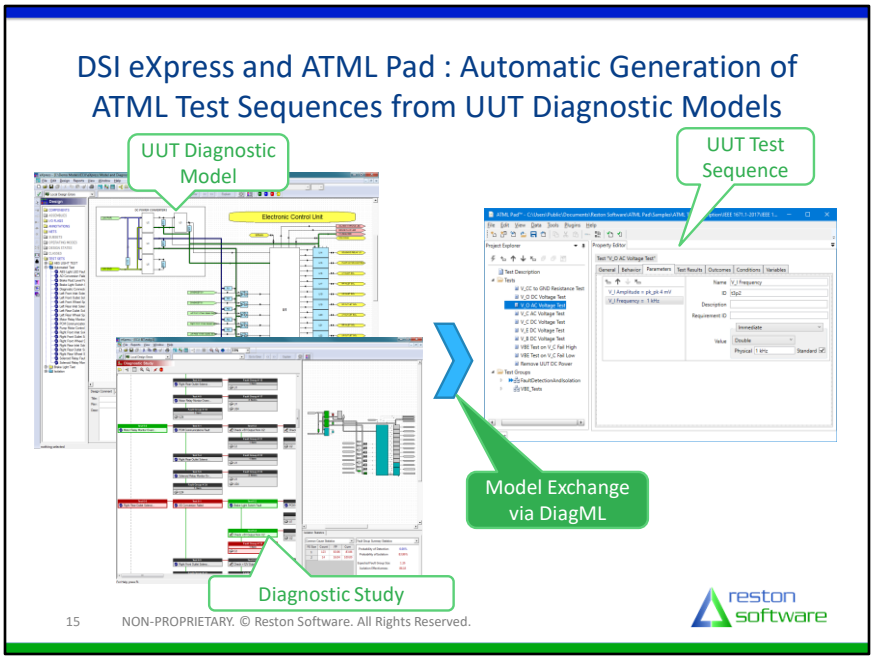
Productivity Features



Sequences of operations commonly used during document creation are available as "Quick Commands" (see example)



ATML Pad has built-in import and export capabilities for ATML Test Description and several other document / file formats. Custom importers and exporter for other formats can be implemented by end users (details in the following slides).



eXpress is a model-based *diagnostics engineering* application developed by DSI International. **eXpress** supports the design, capture, integration, evaluation and optimization of system diagnostics, prognostics health management (PHM), systems testability engineering, failure mode and effects analysis and system safety analysis.

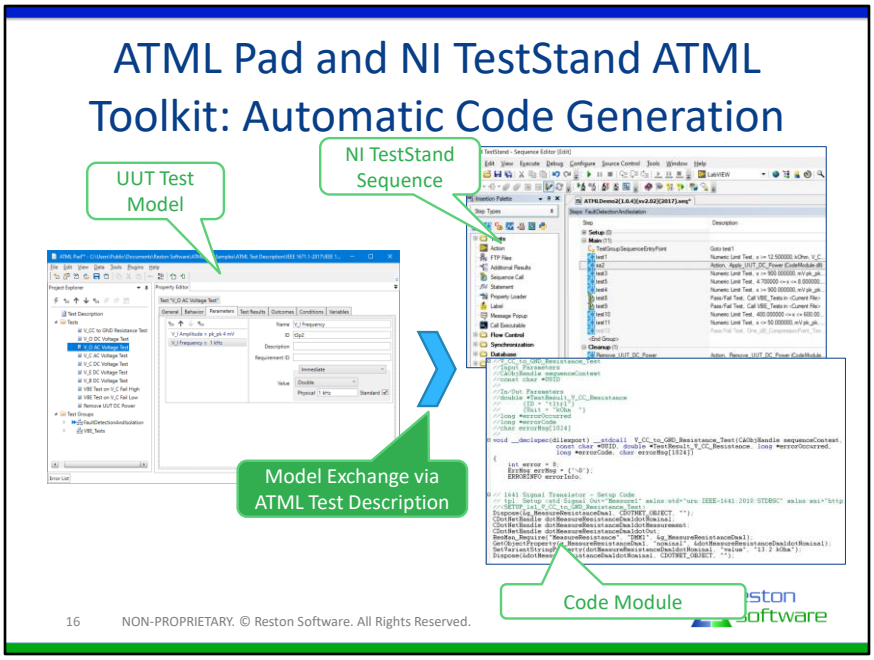
eXpress is integrated with **ATML Pad** through ***DiagML***, an open XML-based format used to represent UUT, test, and diagnostic data. ***DiagML*** is a precursor of ATML Test Description.

Design-to-test development flow using eXpress, DiagML, and ATML Pad:

1. Import UUT design data from CAD, SysML, or spreadsheets to eXpress (optional)
2. Develop diagnostic model in eXpress, adding functional dependencies, failure information, …
3. Develop diagnostic study in eXpress, generating fault trees from the diagnostic model
4. Export fault tree data to DiagML
5. Import DiagML into ATML Pad. The ATMl document will contain “sequence” test groups, tests, and test points. The test behavior will be undefined.
6. Use ATML Pad to add detailed test information: test operations, signals,

measurements, limits, …

7. Export ATML Test Description from ATML Pad
8. Use ATML Test Description to generate test programs

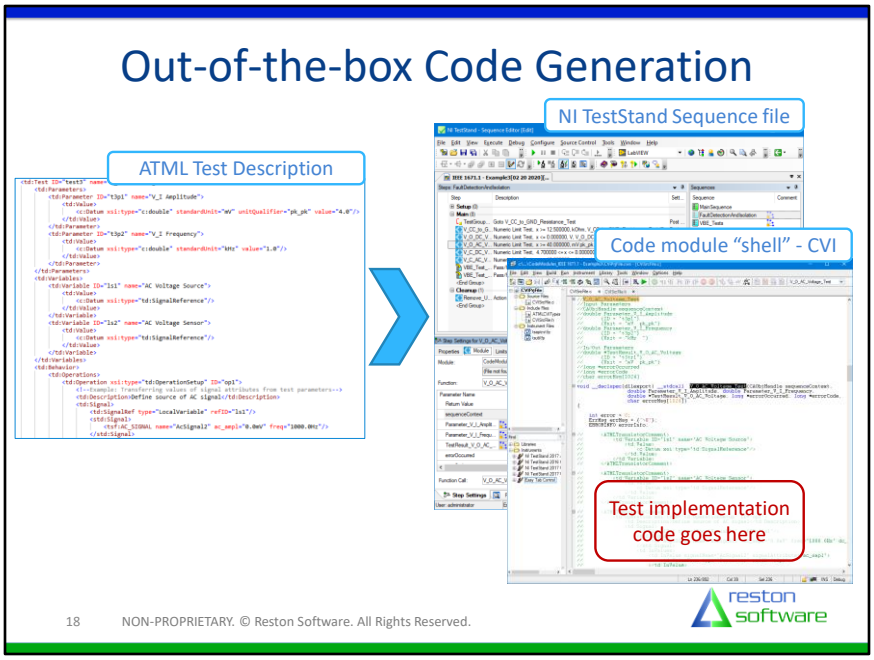


The **NI TestStand ATML Toolkit** is an add-on component of National Instruments (NI) TestStand. It allows TestStand to translate ATML Test Description documents into TestStand sequences and code modules written in LabVIEW or LabWindows™/CVI.

The **NI TestStand ATML Toolkit** is integrated with **ATML Pad** through a plug-in, using the standard ***ATML Test Description*** format. ATML Pad invokes the translator on the model that is currently loaded. The translator generates the test program and opens the generated sequence file in TestStand.

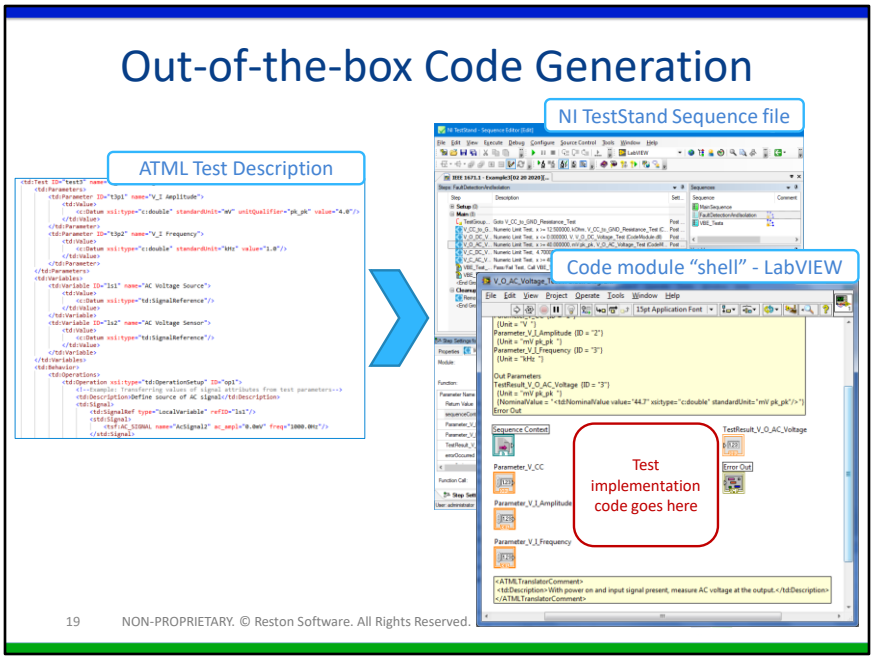
ATML Test Description Translator

- Part of TestStand ATML Toolkit , an add-on product for NI TestStand
- Translates ATML Test Description into a partial test program composed of
  - TestStand sequence files
  - Code modules (LabVIEW or LabWindows/CVI) with “shell code”
    - Interface implementation (VIs or C function definitions)
    - Test behavior description imported as code comments
- The test program is completed through manual coding or user-developed Custom Code Generators

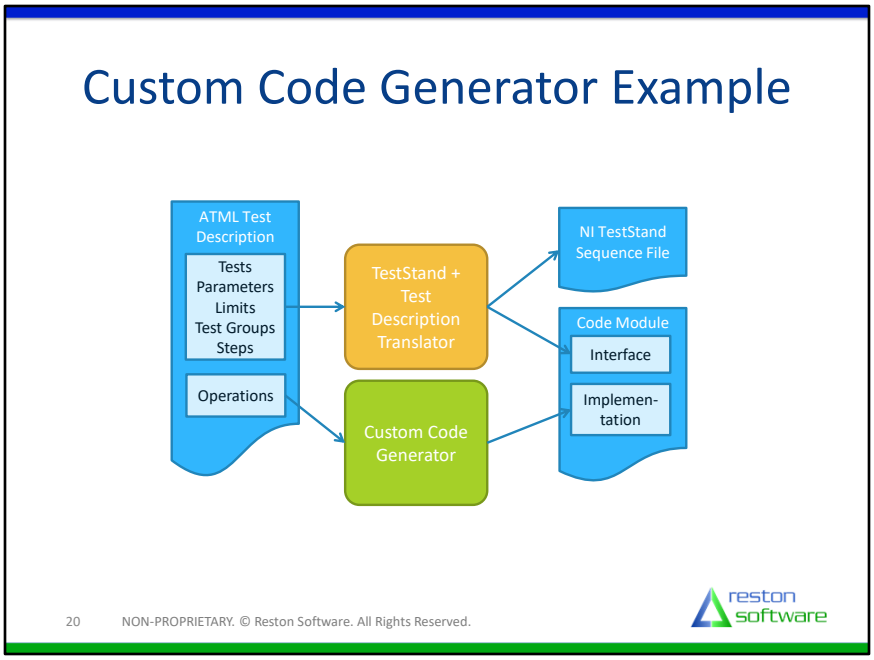


The default conversion process supported by the Test Description Translator is as follows:

1. The translator performs the automatic conversion of ATML Test Description document to a partial test program, which consists of:
    - A TestStand sequence file
    - “Shell” code modules (LabVIEW or LabWindows/CVI)
        - The default translator produces interface implementations only (VIs in LabVIEW; C function definitions in LabWindows/CVI)
        - Test behavior description is imported as code comments
2. Test engineers complete the test program through manual coding. This includes:
    - Implementation of test behavior in code modules, such as:
        - Instrument control
        - UUT commands
        - Calculations
        - Operator I/O, etc.
    - Possibly, additions to the sequence file



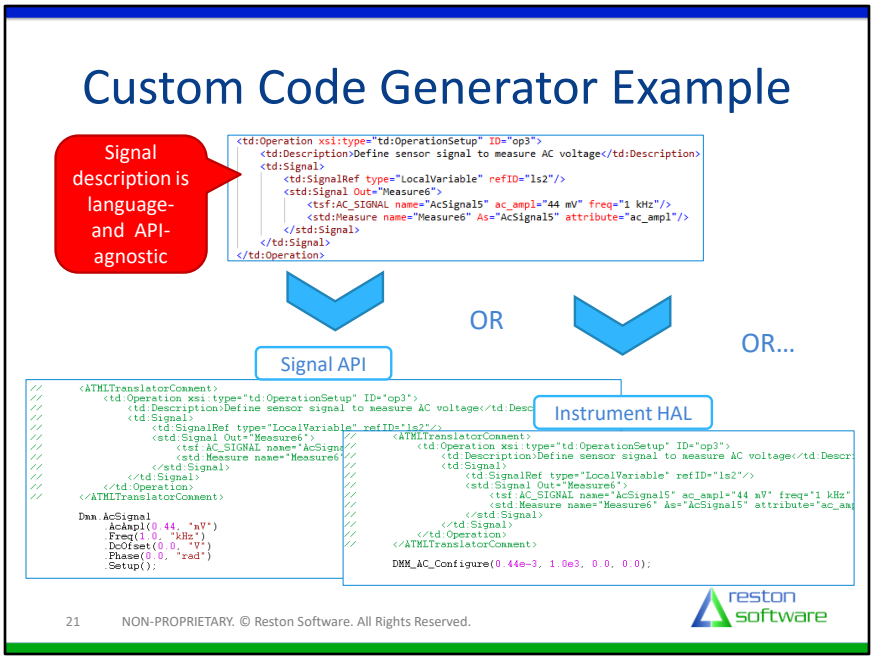
Same process with LabVIEW code



The translation process can be enhanced by *developing custom code generators*

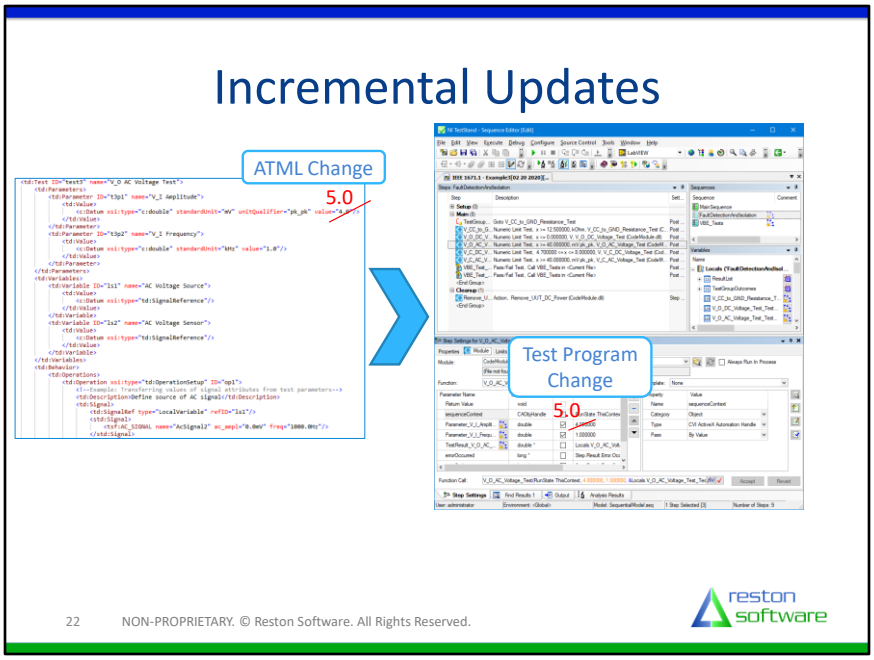
- A generator consists of callback functions written in LabVIEW. These functions are invoked automatically by the translator when specific elements are being translated.
- Custom code generators can be used, for example, to produce code that implements test behavior, thus reducing the need for manual coding.

The custom code generator translates test behavior information (usually Operations, but can be custom format) into source code implementing that behavior.



The generated code could contain:

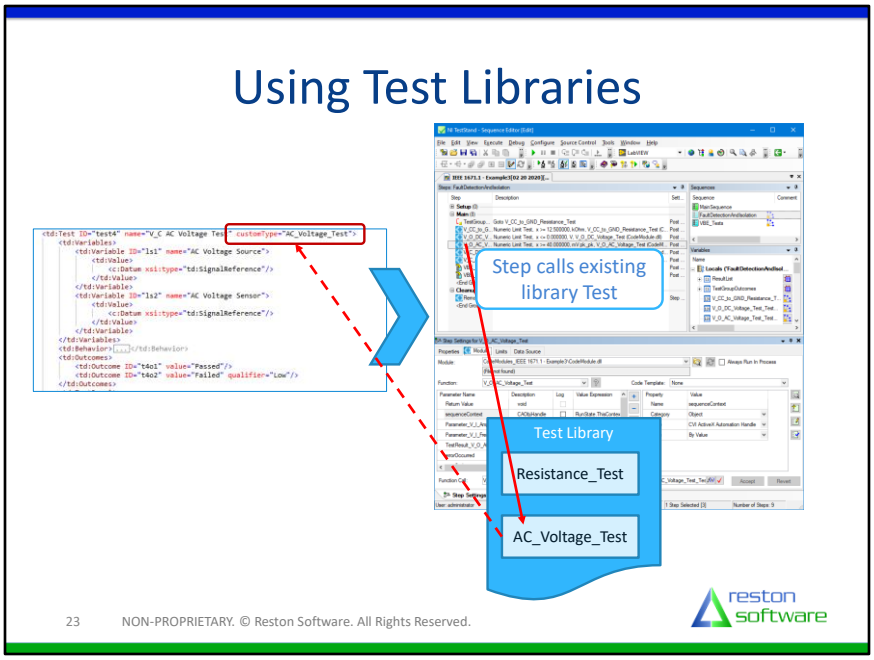
- Calls to a signal-oriented API
- Calls to an instrument-oriented Hardware Abstraction Layer (HAL)
- Calls to instrument drivers
- IEEE 1641 Test Procedure Language (TPL), which can be:
    - Converted to IEEE 1641 API calls and executed on 1641 run time
    - Translated to a programing language in a later step



When the ATML document changes, the translator can be invoked in “update mode”.

In this mode, the translator modifies the previously generated test program, propagates changes made to the ATML document.

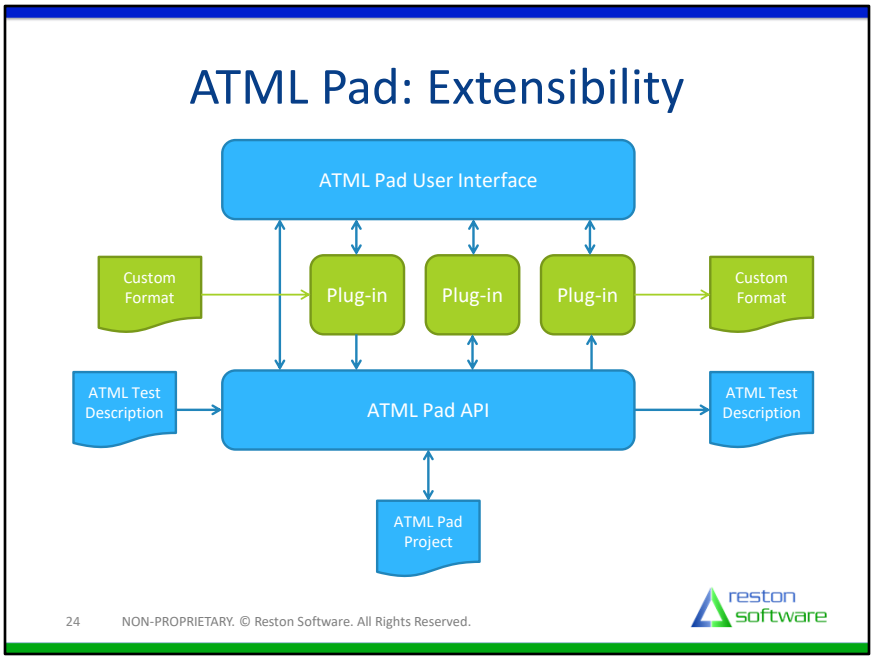
The translator preserves changes that were made manually.



Large test organization often maintain a library of commonly used tests. In TestStand , these are typically implemented as custom step types.

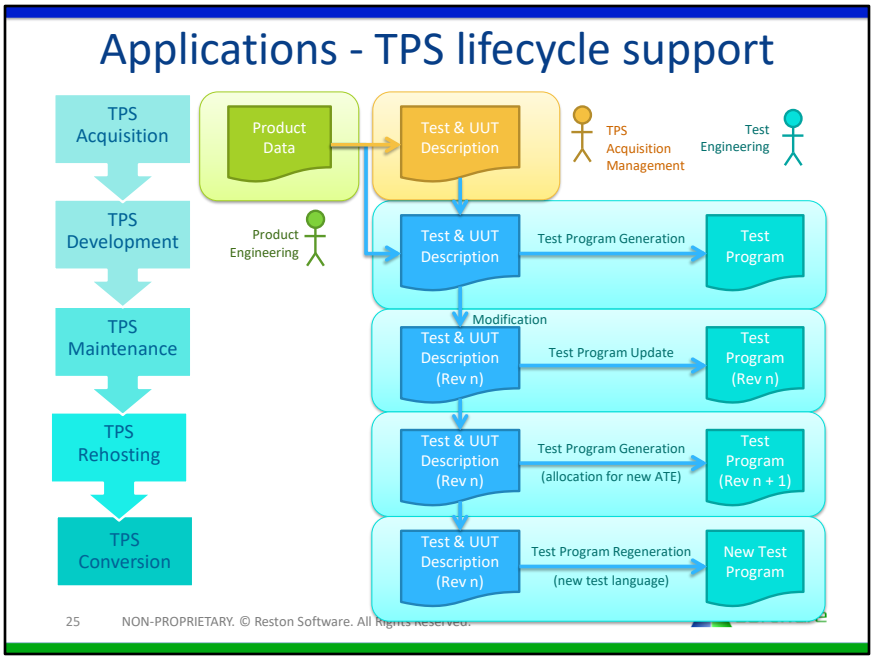
Tests described in the ATML document can be mapped to these custom step types through a specialized XML element.

The translator recognizes the element and generates a call to the custom step type, instead of generating a new code module.



End users can create plug-ins to

- Import data from custom formats
  - Multiple importers can aggregate data from multiple sources (design data, simulation, diagnostic analysis)
  - Custom importers can extract test requirements from legacy code
- Export data to custom formats
  - Including automatic code generation for custom test languages
- Process data “in place”
  - Example: automatic design verification (check that specific data items exist)
  - Example: reorder tests to reduce average test execution time



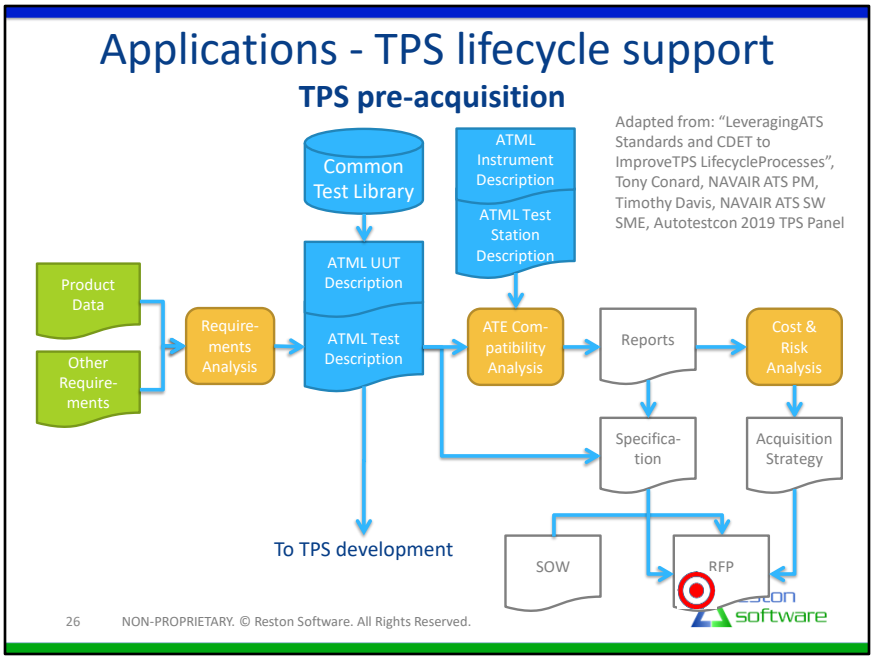
TPS Acquisition: An initial, high-level Test Description is created. This includes, for example, power requirements, signal requirements, and a test list. The high-level description allows the estimation of development time, cost, risk, and reuse potential through comparison with past developments.

TPS Development: Details are added to the Test Description. This includes test parameters, test results, test sequences, and detailed test operations. The description has a sufficient level of detail to enable automatic test program generation.

TPS Maintenance: Changes are made to the Test Description; the auto-generated test program is updated automatically. This ensures consistency between the Test Description (and thus test program documentation) and the test program.

TPS Rehosting: The test program requirements are re-allocated to the capabilities of a new ATE. A new revision of the test program is generated.

TPS Conversion: New code is generated in a different test language. A new test program is generated.



Requirements Analysis

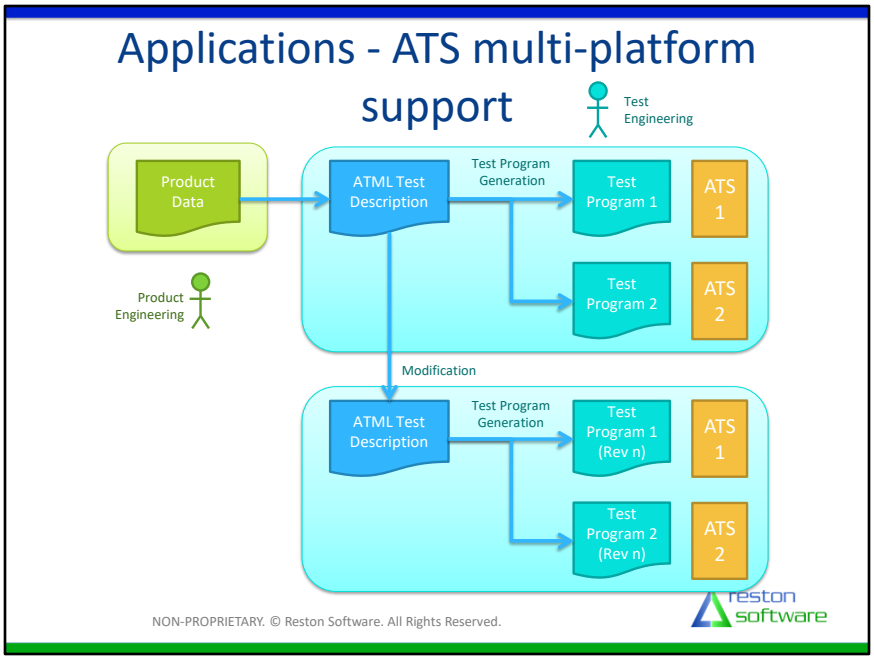
- Inputs
  - UUT Physical Characteristics
  - UUT Electrical I/O
  - Theory of Operations
  - Critical Components
  - Failure Modes
  - Logistics Analysis
  - etc.
- Outputs
  - ATML UUT Description
  - ATML Test Descriptions
    - Can pull preexisting tests from library

ATE Compatibility Analysis

- Inputs:
  - TPS requirements (signal-oriented)
  - ATE capabilities (signal-oriented)
- Outputs

- Exception Report
- Complexity Report
- TAR Report

Assemble RFP for SRR-1

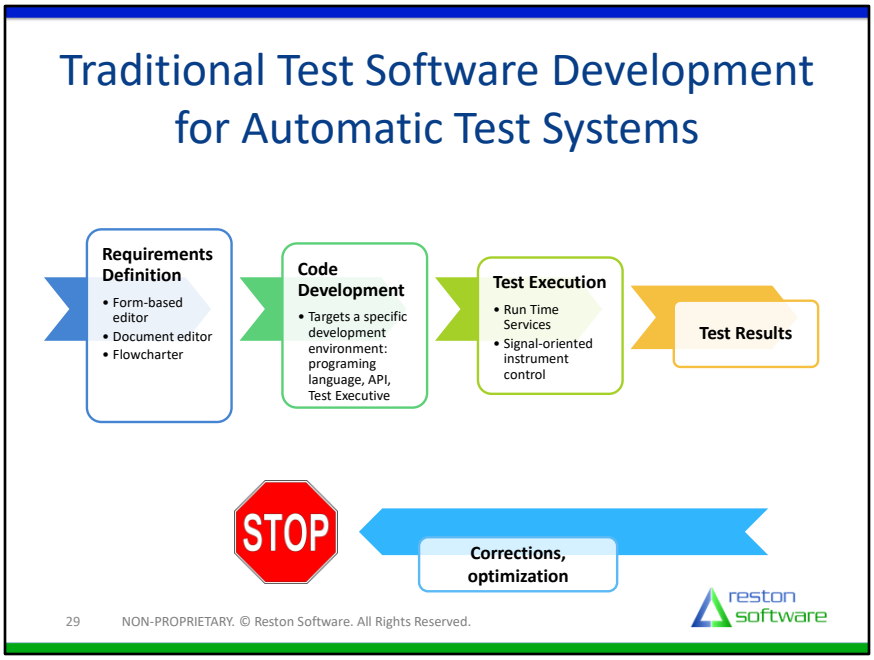


1. From a common set of requirements, two test programs are generated, targeting two different hardware platforms (ATS 1 and ATS 1)
2. If a modification is needed (ex. correction or optimization), the modification is performed on the Test Description model. New revisions of the Test Programs are auto-generated.

This approach eliminates the need to modify two different test programs and preserves their consistency automatically.

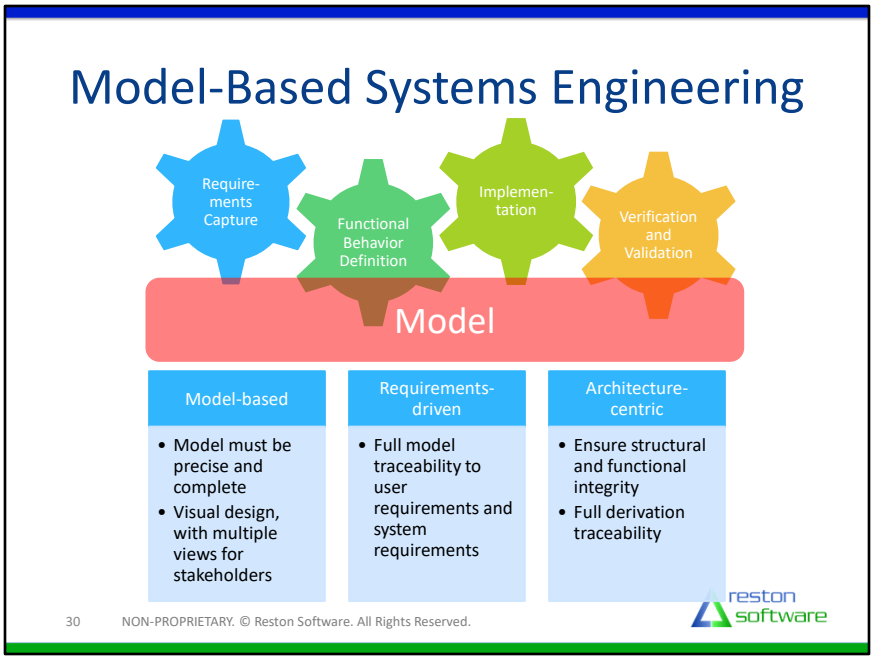
Model-Based Test Development with ATML Pad

reston software

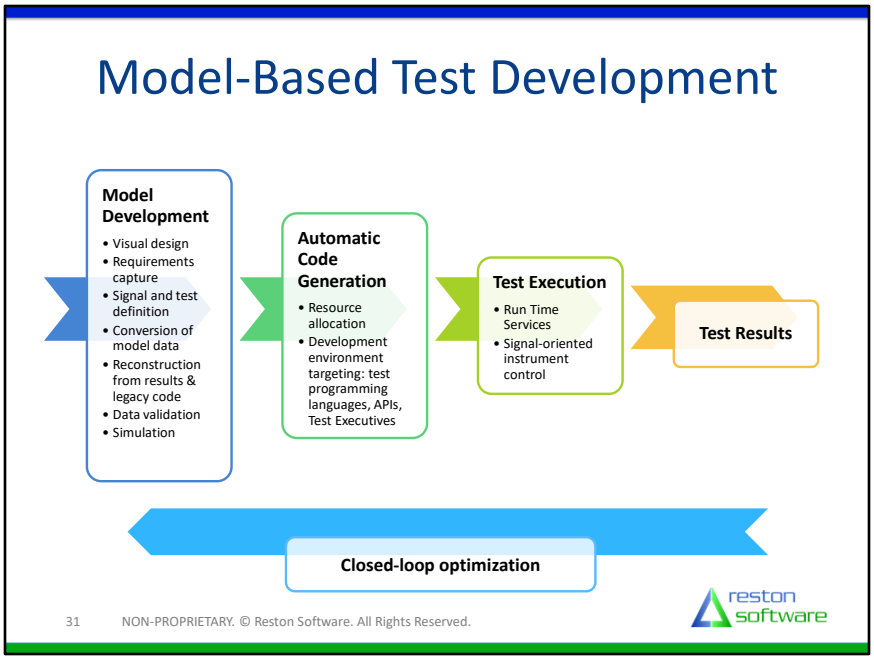


Problem: corrections and optimizations are made in code and not in requirements.

- In time, the code diverges from requirements
- Original requirements become obsolete, can no longer be reused to support code changes during code maintenance, rehosting, or conversion



The discipline of Model-Based Systems Engineering advocates the use of a common digital model to support all phases of the System Engineering process



Making use of model-based systems engineering principles, the test development process can be modified as shown.

Model development

Models for UUT, Test Requirements, ITA, Test Station, Instruments, …
Graphical / visual design
Model data import from: EDA, systems simulation, diagnostic modeling, legacy systems
Model extraction / reconstruction from: test results, non-standard models (ex. TRD), unstructured legacy data (ex. ATLAS code)
Data validation
Simulation

Automatic code generation

Resource allocation – targeting implementation to a specific ATE
Targeting implementation to a specific test development environment: test programming languages, APIs, test executives

Test execution

Run-time signal services
Signal-oriented instrument control

Generation of test results

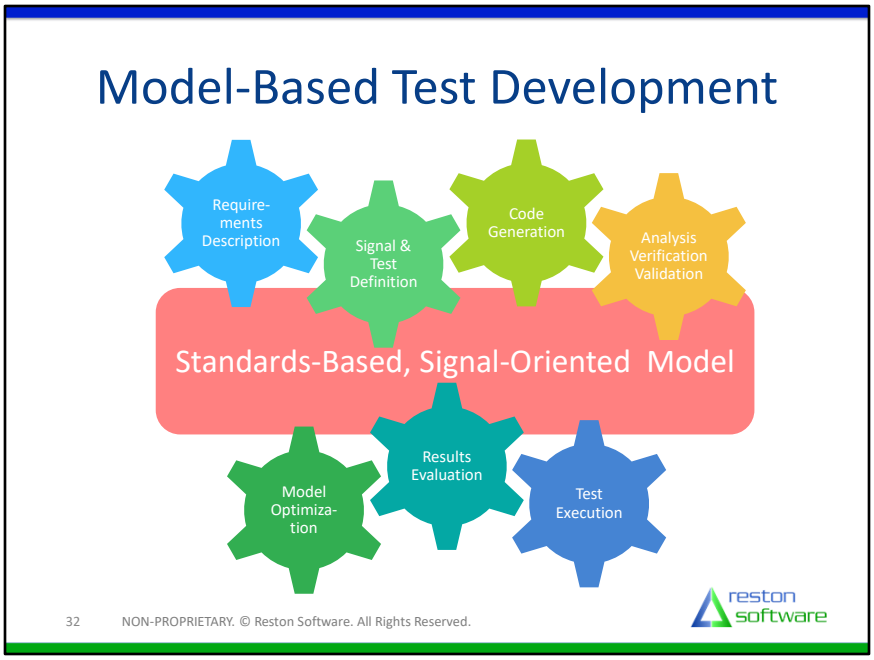
Closed-loop optimization of

Maintenance

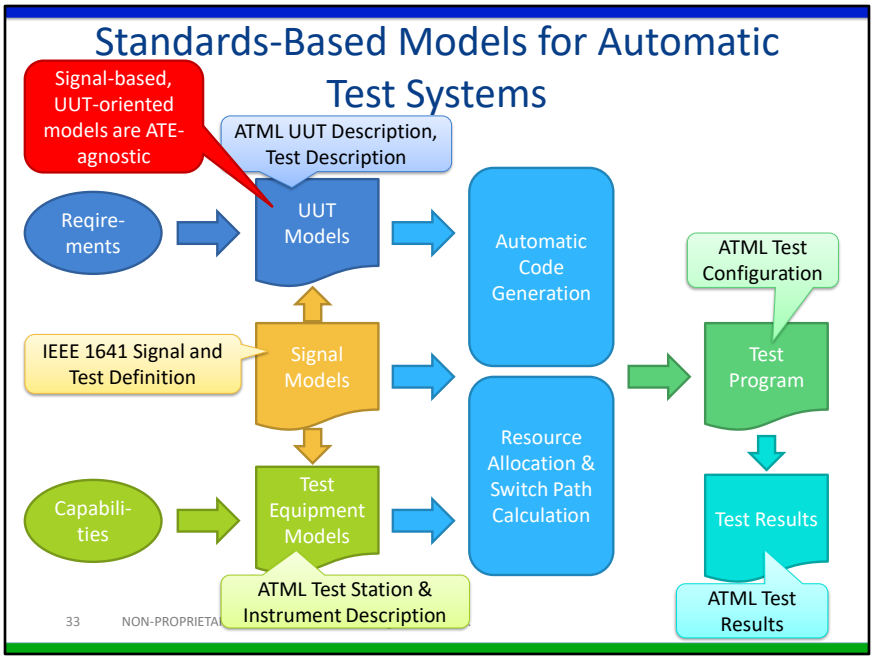
Diagnostics

Test

Closed-loop optimization is performed on the Model, not on code. The code is regenerated automatically.



All stages of test program development (requirements analysis, design, implementation, use, maintenance, and optimization) are based on a *common model*, using *industry-standard formats* and *signal abstractions*.



What standard data formats are available for modeling Automatic Test Systems?

- IEEE 1671 ATML (Automatic Test Markup Language)
- IEEE 1641 Signal and Test Definition

Signal-based models tell you what to do, but not how to do it (in terms of instrument operations). This allows them to be implemented & re-implemented on a variety of ATE platforms.

Resource Allocation is the selection of an instrument or instrument subsystem for each signal operation.

Switch Path Calculation identifies signal paths through the ATE switching subsystem, from instruments to the UUT, for each Connect / Disconnect signal operation.

COTS Tools and Industry-Standard Data Formats

Storage of UUT, Test, and Maintenance data in vendor-independent formats	• Data ownership • Long-term sustainability of Automatic Test Systems
Model-based test development	• Automatic code generation • Full traceability to requirements
Signal-based, UUT-oriented models	• Multi-platform support
Multi-vendor solutions	• Through-life support for UUTs • Feedback loops to optimize design, test, and maintenance

reston software

Glossary & Abbreviations

- **UUT = Unit Under Test**: The entity to be tested. It may range from a simple component to a complete system
- **Test program**: A program specifically intended for the testing of a **UUT**
- **TPS = Test Program Set**: The complete set of hardware, software, and documentation needed to evaluate a **UUT** on a given test system
- **ATE = Automatic Test Equipment**: a system providing a test capability for the automatic testing of one or more **UUTs**. The ATE system consists of a controller, test resource devices, and peripherals. The controller directs the testing process and interprets the results. The test resource devices provide stimuli, measurements, and physical interconnections.
- **ATS = Automatic Test System**: Includes the **ATE** as well as all support equipment, software, **test programs**, and adapters.
- **ATML = Automatic Test Markup Language**: a family of standards specified in IEEE 1671, IEEE 1636.1, and IEEE 1641

reston software

Thank you!

reston software