

Model-Based Design for High Integrity Software Development

Mike Anthony
Senior Application Engineer
The MathWorks, Inc.
Tucson, AZ USA

Model-Based Design for High Integrity Software Development

Agenda

Development and V&V of the Model

- Building a Model from Requirements
 - Introduction to Simulink
- Traceability of a Model to Requirements
 - Using the Requirements Management Interface
 - The Requirements Report
- Conformance to Modeling Standards
 - Using the Model Advisor
 - Customizing the Model Advisor
 - Model Advisor Report
- Verification of the Model against Requirements
 - Introduction to SystemTest
 - SystemTest Report
 - Introduction to Simulink Design Verifier: Property Proving

Development and V&V of the Code

- Production Code Generation
 - Creating Data Objects
 - Function Prototype Control
- Traceability of the Generated Code to the Model
 - Code-to-Model Linking
 - Model-to-Code Linking
 - Traceability Report
- Conformance to Coding Standards & Code Verification
 - PolySpace
 - MISRA-C Compliance
 - Proving the Absence of Runtime Errors
- Verification of the Generated Code against the Model
 - Introduction to Simulink Design Verifier: Test-Vector Generation
 - SystemTest
 - Embedded IDE Link Products for PIL
- Verification of the Generated Code against the Requirements
 - SystemTest: Test Case reuse
 - Embedded IDE Link Products for PIL

Aerospace Standards

- **RTCA/DO-178B Guidelines**
 - Commercial standard (FAA, JAA)
 - Software Integrity Levels A-E based on hazards
 - Level A if failure hazards can cause loss of life or limb
 - Structural coverage (MC/DC)

- **UK MOD 0055/0056**
 - Software Integrity Levels 1-4
 - Requires formal analysis and software proofs
 - Has SPARK language and data flow checks

- **MIL-STD-498**
 - Formerly DOD-2167A
 - US military and defense
 - Emphasizes verification and validation activities



DO-178B

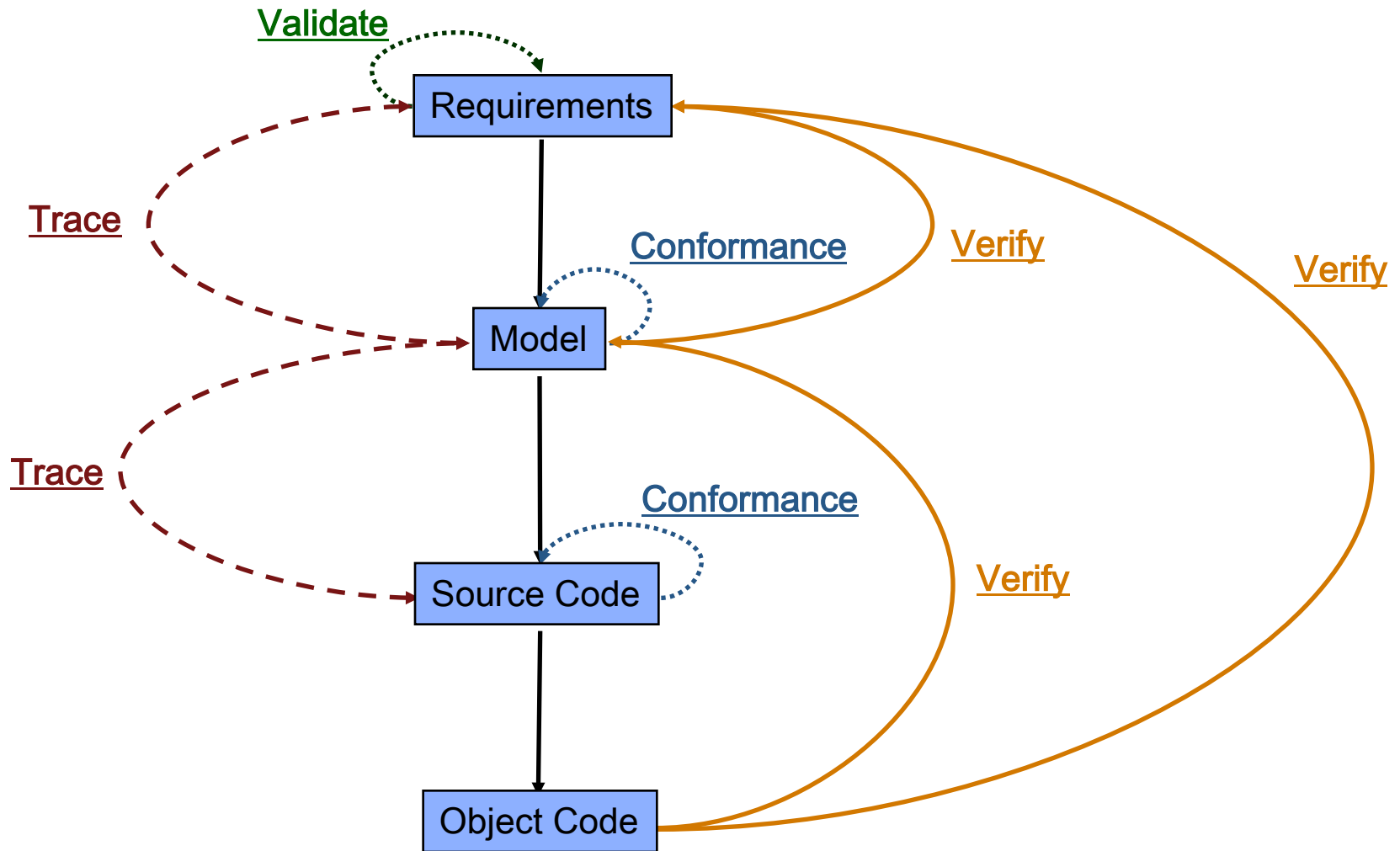
Methods for Verification and Validation

Verification: Did I do the design right?

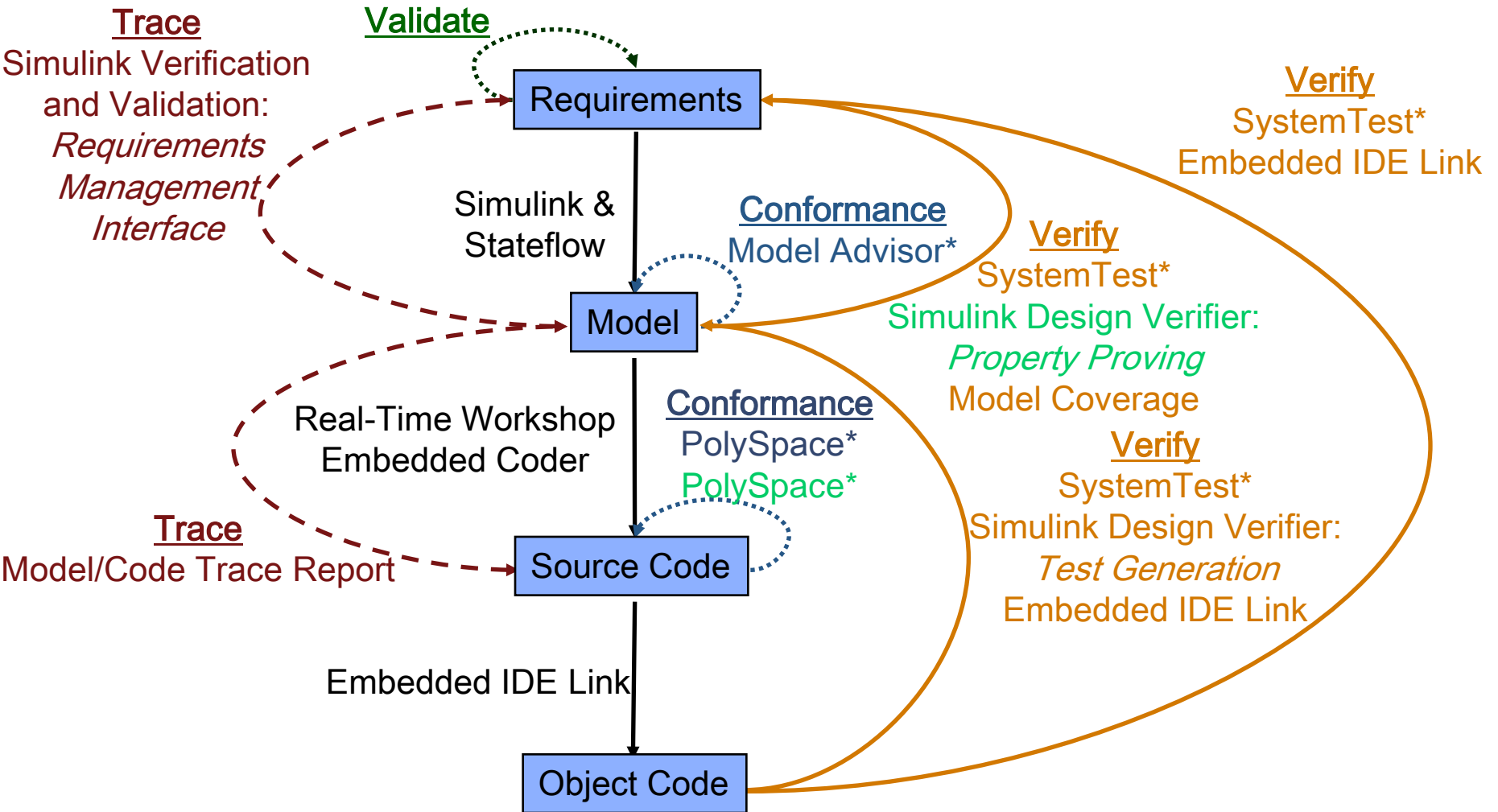
Validation: Did I do the right design?

- **Traceability**
 - Requirements to model and code
 - Model to code
- **Modeling and Coding Standards**
 - Modeling standards checking
 - Coding standards checking
- **Testing**
 - Model testing in simulation
 - Processor In the loop
- **Proving**
 - Proving design properties
 - Proving code correctness

Workflow Example

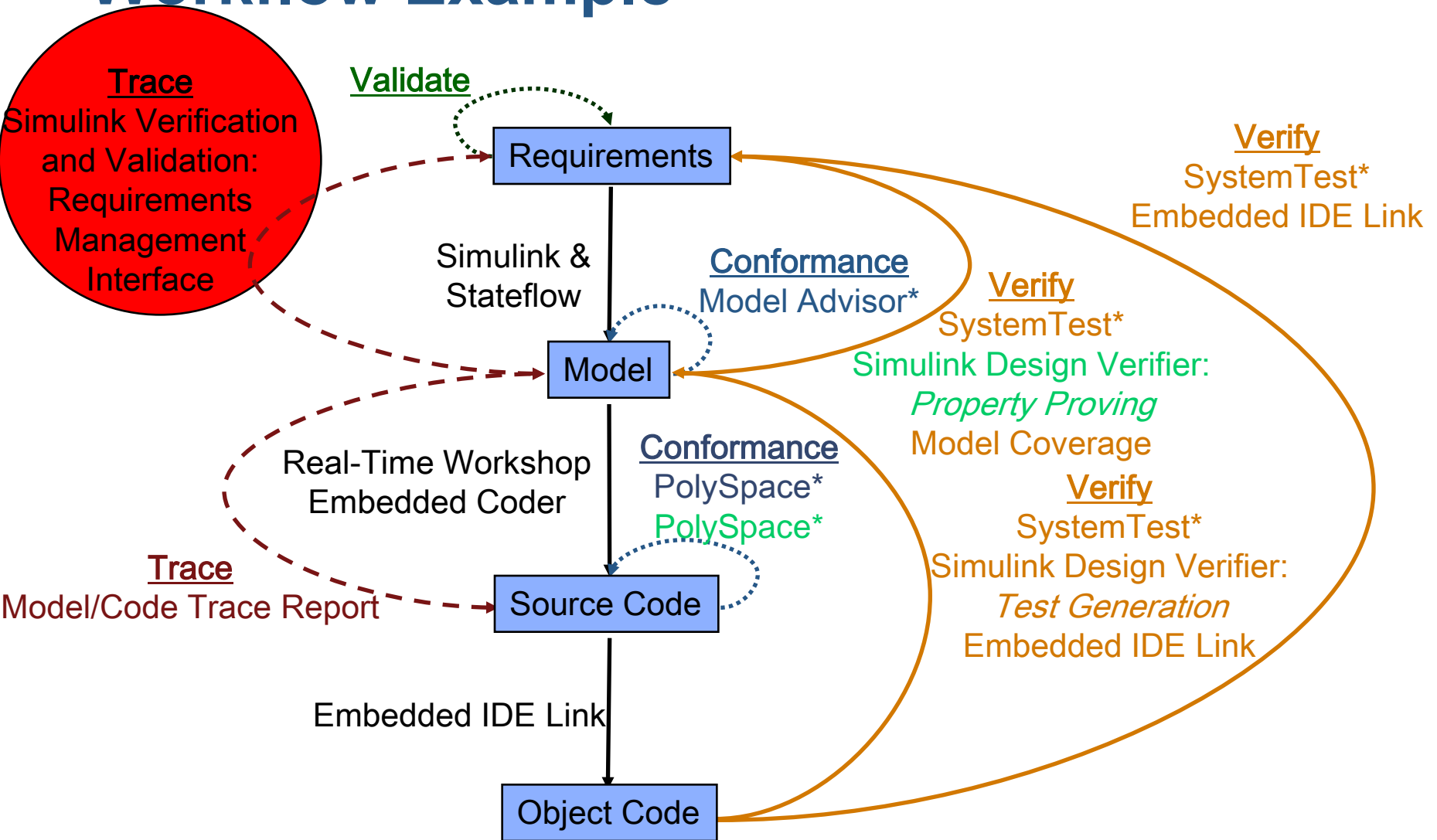


Workflow Example



* DO-178B Qualifiable Tool

Workflow Example

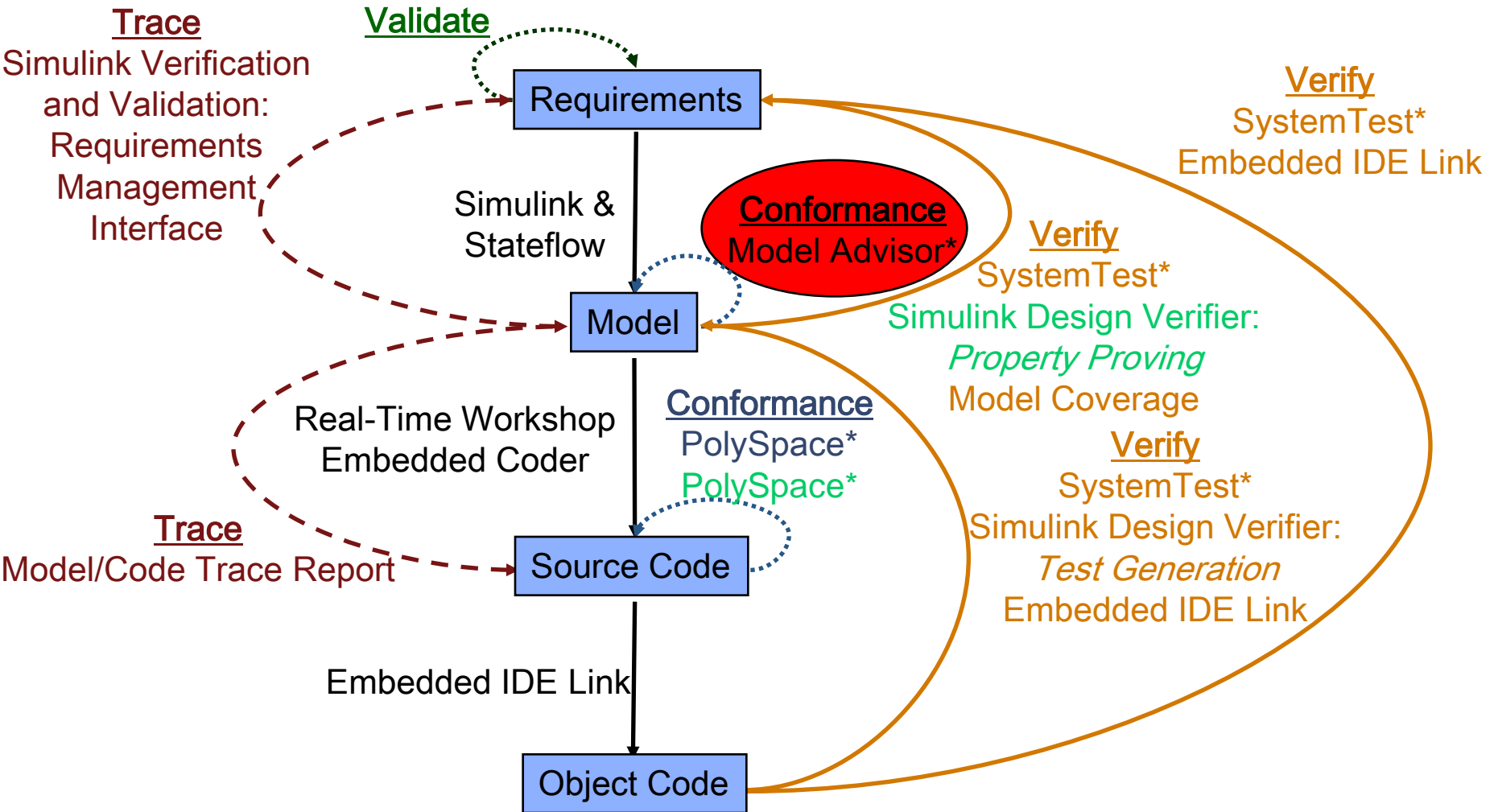


* DO-178B Qualifiable Tool

Requirements Management Interface Overview

- Associating models and requirements
 - Establishing a link from a model block or test case to requirement
 - Establishing a link from a requirement to a model block or test case
- Managing changes in models and requirements
 - Detecting a change in a requirement associated with a model block or test
 - Detecting a change in a model block or test associated with a requirement
- Reporting requirement coverage for model blocks and test cases
 - How many and what algorithmic blocks are covered by requirements
 - How many and what test cases are covered by requirements
 - How many of the requirements are covered by test cases within the model
 - How many of the requirements are associated with algorithmic blocks within the model

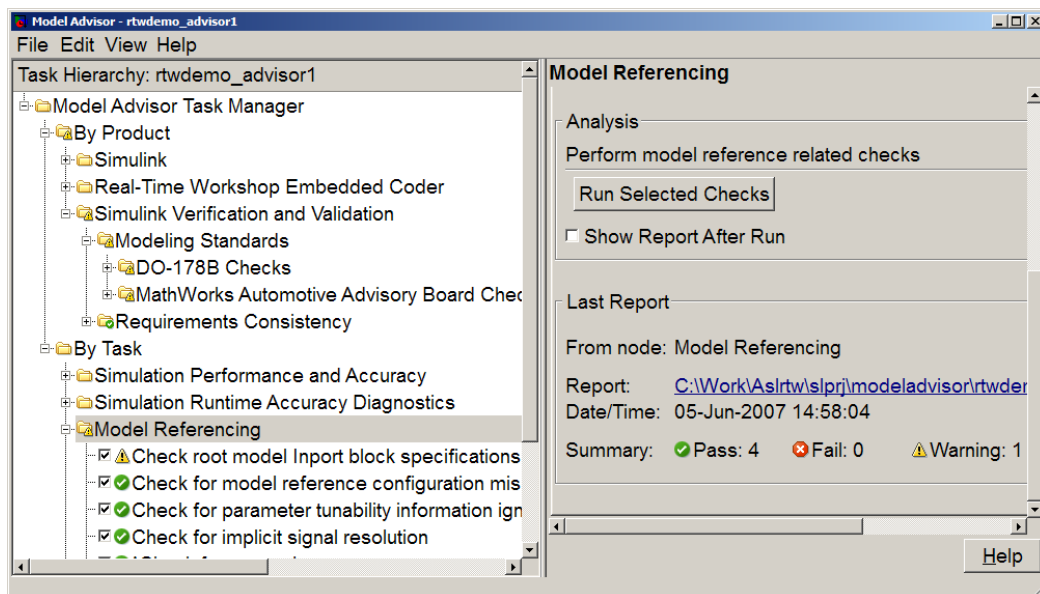
Workflow Example



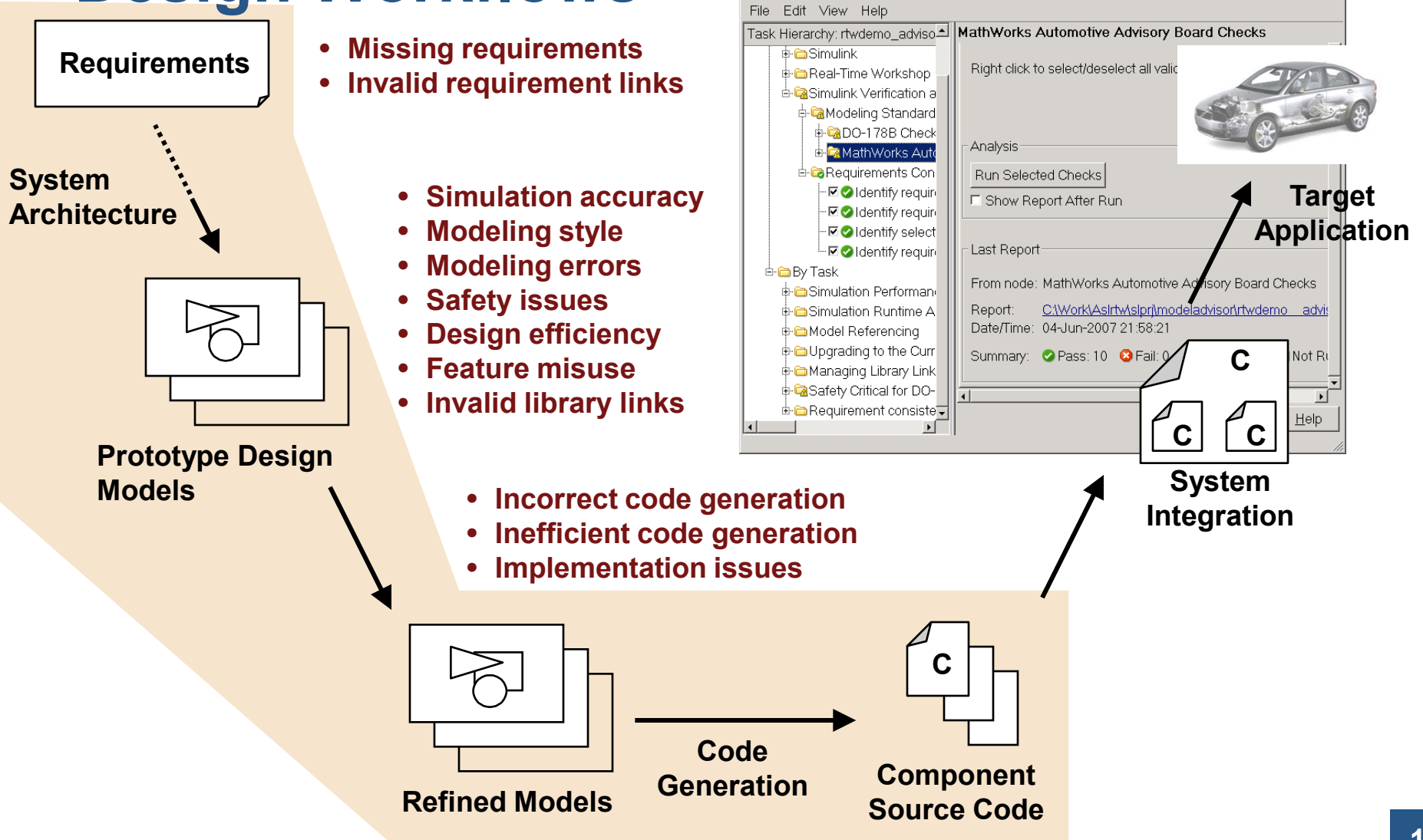
* DO-178B Qualifiable Tool

Simulink Model Advisor

- Model Advisor is used to
 - Enforce model standards and best practices
 - Detect and troubleshoot modeling and code generation issues
 - Check models for (a subset of) known version upgrade issues



Model Advisor Within Model-Based Design Workflows



Simulink Verification and Validation

Additional Model Standards Checking

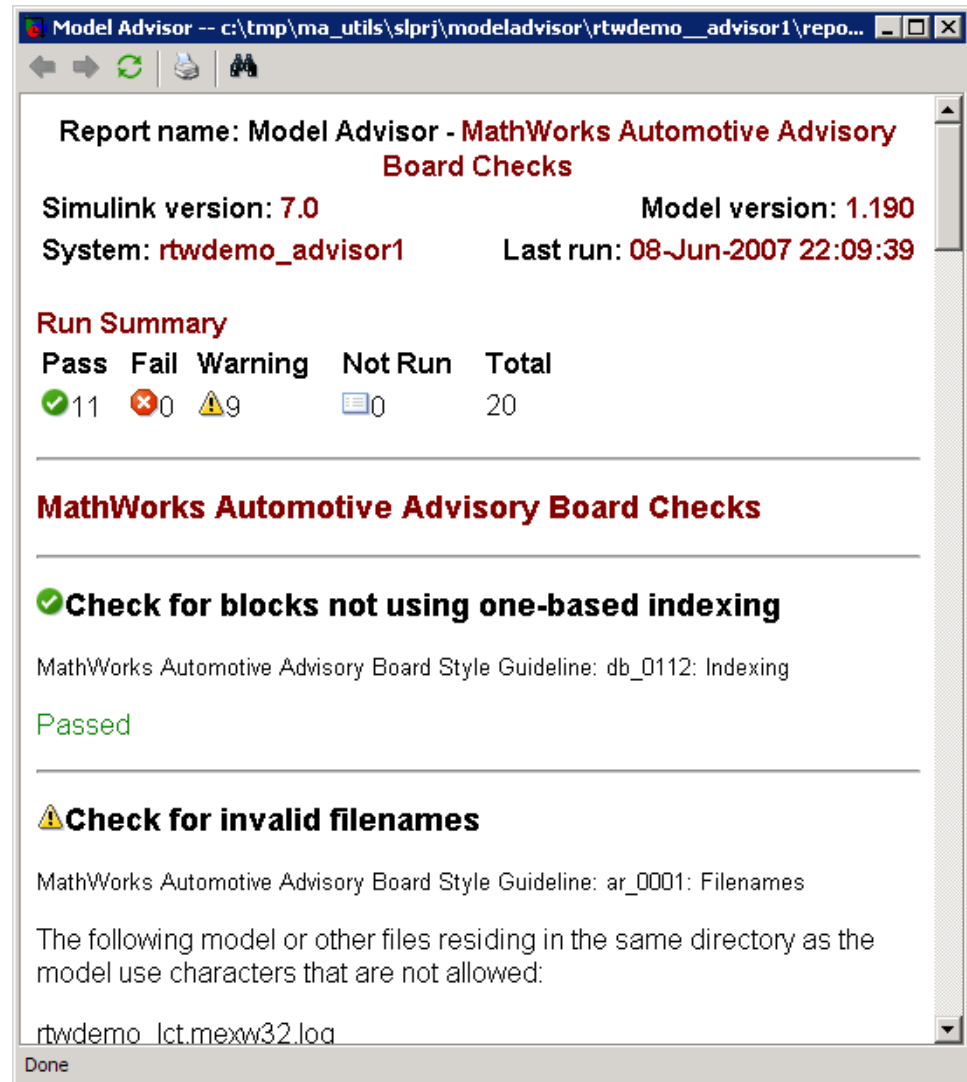
- DO-178B Checks
 - Focus on generation of safety critical code from models
 - Assist in MISRA-C compliance
 - Maximize traceability of code to model
 - Minimize differences between model coverage and code coverage
 - Maximize the use of built-in Simulink and Stateflow diagnostics during simulation
- MathWorks Automotive Advisory Board Checks
 - Simulink style guide created by MathWorks Automotive Advisory Board
 - Best practices for consistent and readable models

Model Advisor

Report

Report enhanced to be more useful as a process audit Document:

- More detailed summary
- Report follows exact order of the Model Advisor tree
- Valid check states: Pass, Fail, Warning, and Not Run



Model Advisor -- c:\tmp\ma_utils\slprj\modeladvisor\rtwdemo__advisor1\repo...

Report name: Model Advisor - MathWorks Automotive Advisory Board Checks

Simulink version: 7.0 Model version: 1.190
 System: rtwdemo_advisor1 Last run: 08-Jun-2007 22:09:39

Run Summary

Pass	Fail	Warning	Not Run	Total
11	0	9	0	20

MathWorks Automotive Advisory Board Checks

✓ Check for blocks not using one-based indexing

MathWorks Automotive Advisory Board Style Guideline: db_0112: Indexing

Passed

⚠ Check for invalid filenames

MathWorks Automotive Advisory Board Style Guideline: ar_0001: Filenames

The following model or other files residing in the same directory as the model use characters that are not allowed:

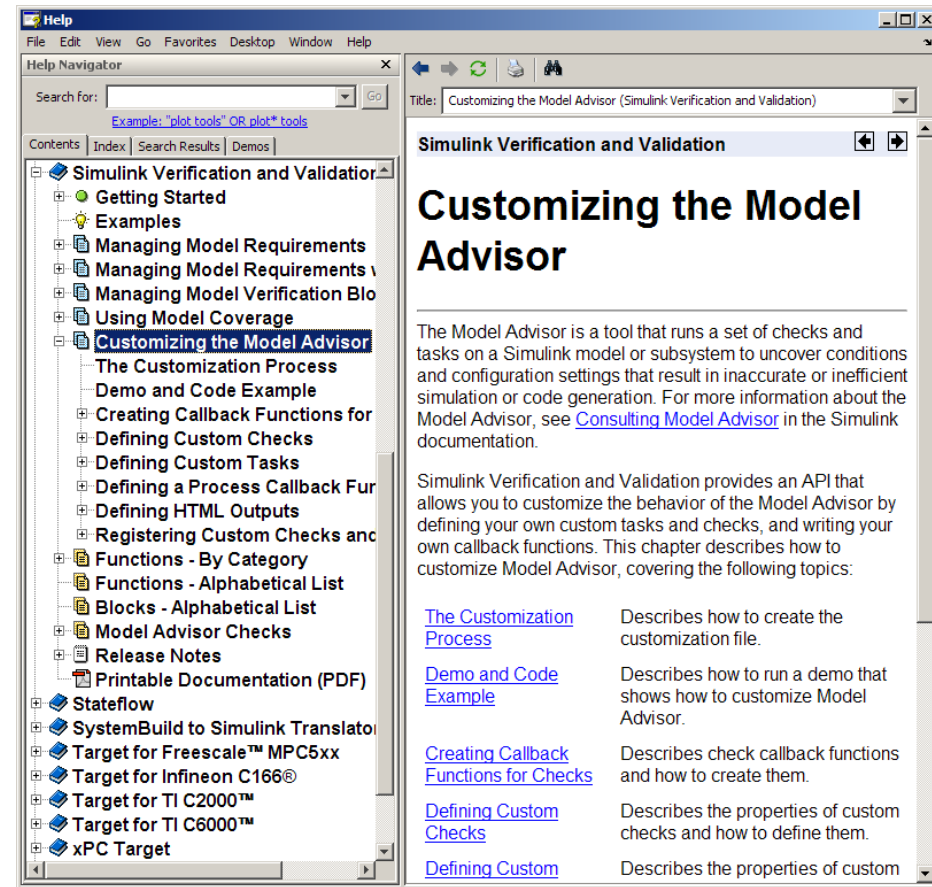
rtwdemo_lct.mexw32.log

Done

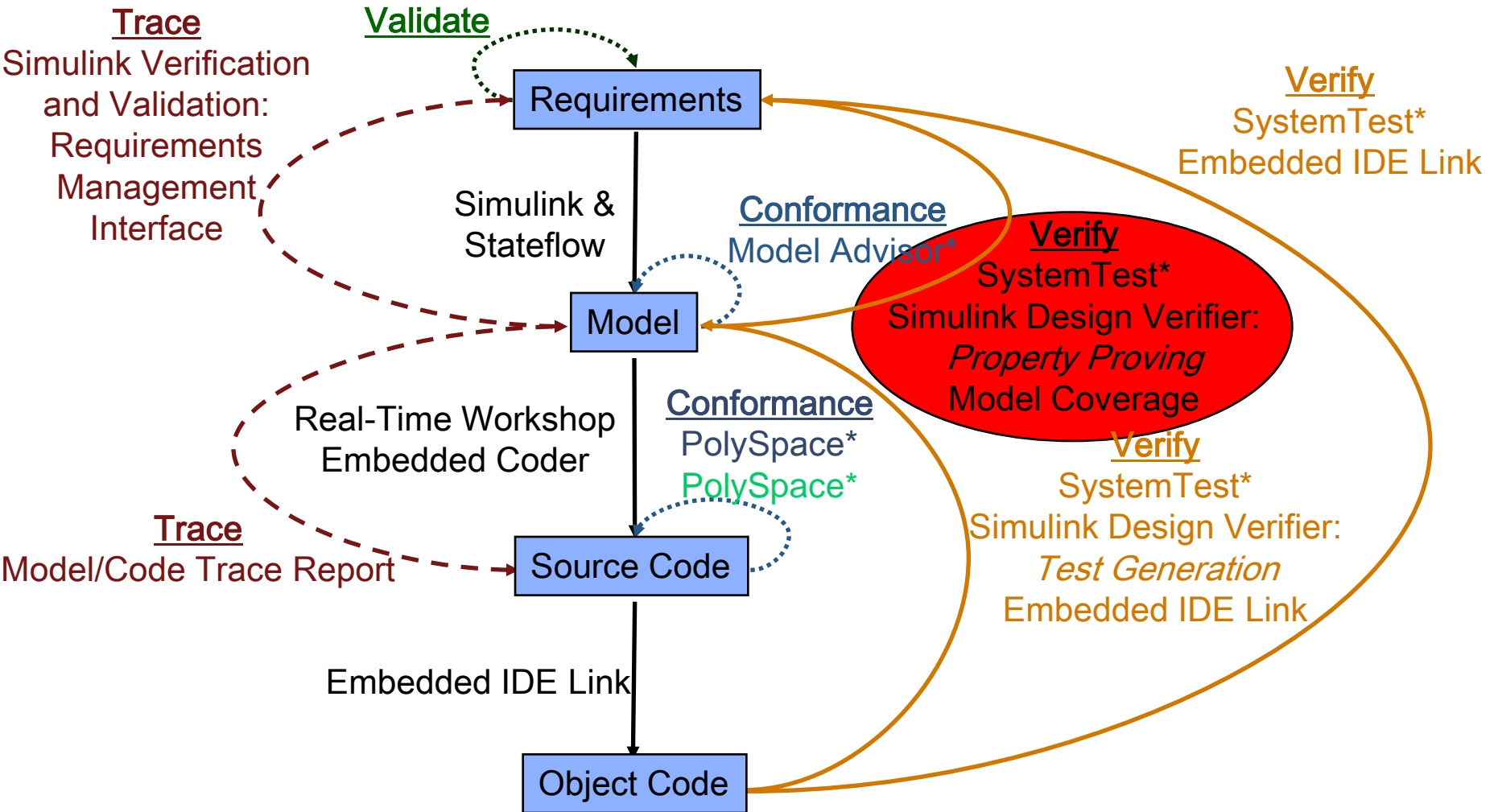
Model Advisor

Enterprise Deployment

- The Model Advisor is highly customizable:
 - Add additional task groups and checks
 - Permanently enable/disable, and hide specific checks
- Benefits
 - Enforce your specific process and standards
 - Prevent defects at specific points early in your design process



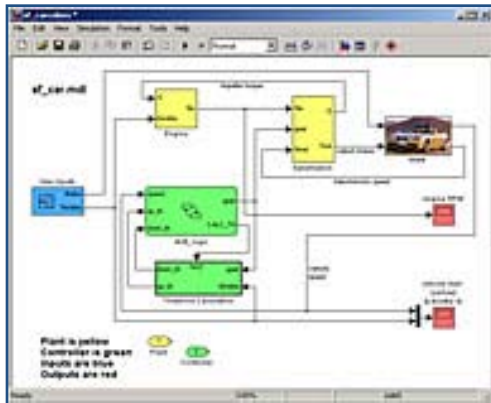
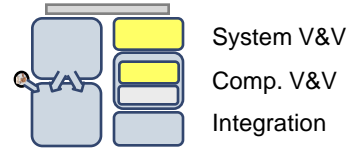
Workflow Example



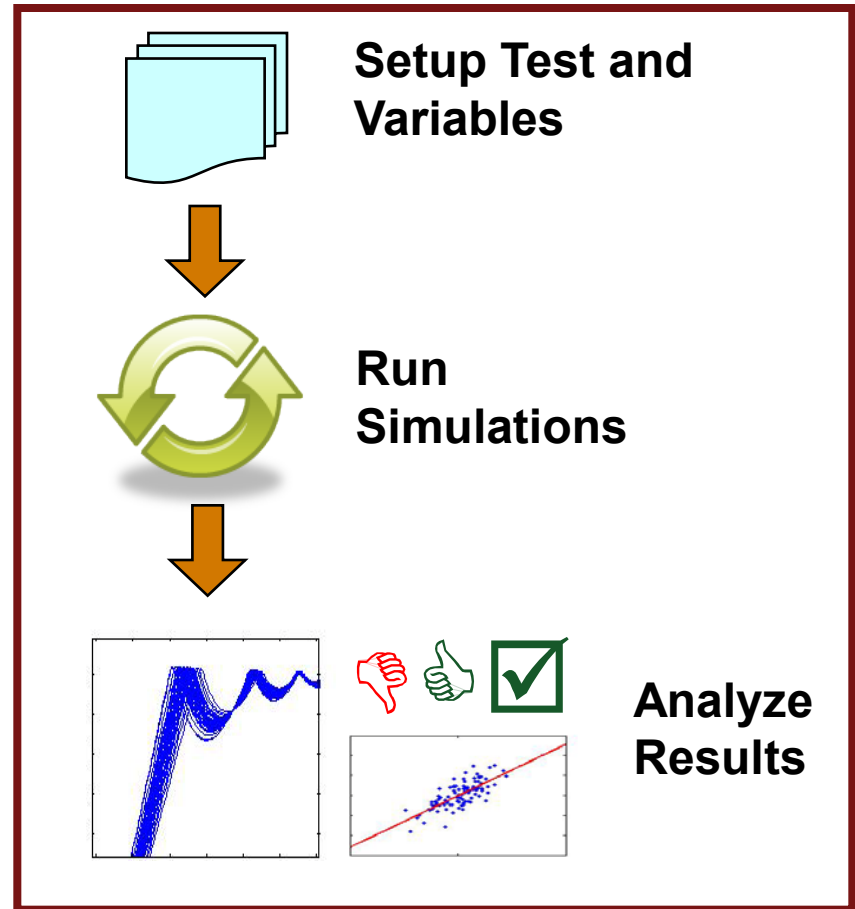
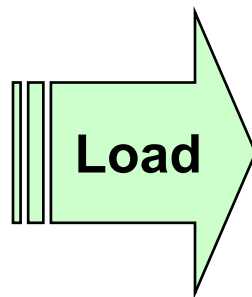
* DO-178B Qualifiable Tool

SystemTest Software

- Manage tests and analyze results for system verification and validation



Simulink System Model



SystemTest

SystemTest

Key Features

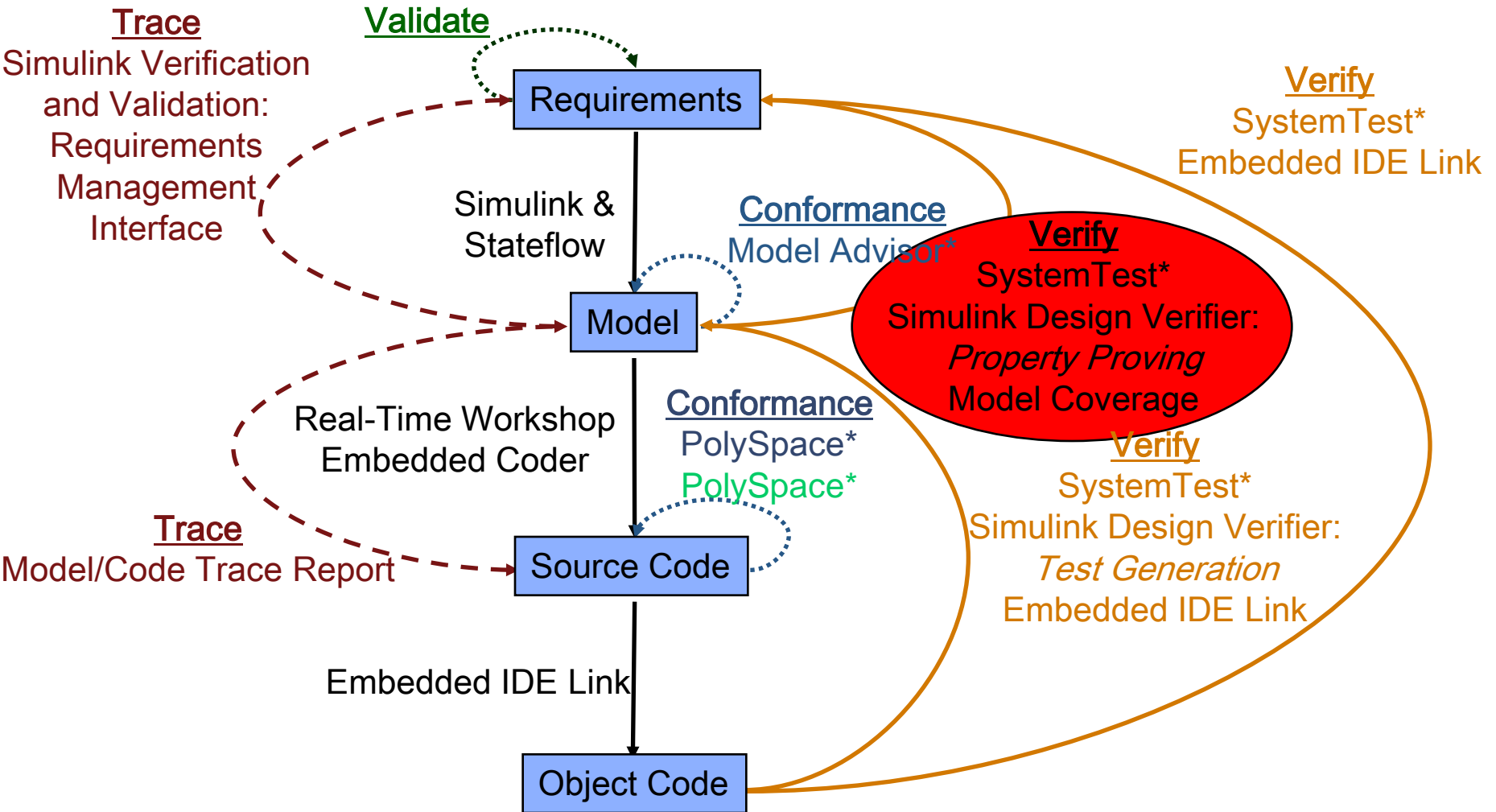
- Develops, manages, and edits test structures using predefined test elements in a graphical user interface
- Stores tests in a separate TEST-file independent of the model under test for repeatable test execution
- Defines pass/fail criteria for tests using Boolean constraints and tolerance limits
- Generates random test vector values using probability distribution functions, especially useful for Monte Carlo simulations
- Runs iterations, such as parameter sweeps, of Simulink models on multiple processors with Distributed Computing Toolbox (available separately)
- Generates reports of test execution and results
- Visualizes and analyzes multidimensional test results in Test Results Viewer

SystemTest

Sample Applications

- Stress testing
- Parameter sweeps
- Model verification and validation
 - Vary block parameters
 - Measure and report model coverage
(with Simulink Verification and Validation)
- Algorithm verification and validation
- Monte Carlo simulation

Workflow Example



* DO-178B Qualifiable Tool

Simulink Design Verifier

Property Proving

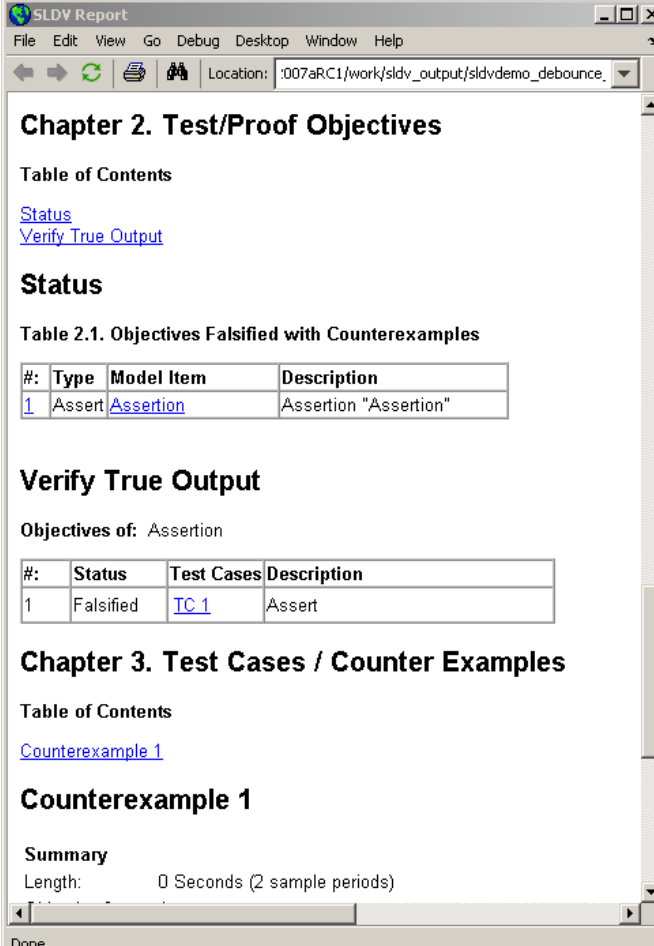
- Functional testing
 - Generates a proof for a requirement
 - For example: Thrust reversers shall not deploy in flight
 - Includes blocks for definition of properties
 - Proves model properties and generates example of violations
 - Produces detailed property-proving analysis reports
- Uses formal methods, not simulation

Property Proving

Verification Results

- Proof or assertion can be found:
 - Satisfied
 - Falsified
 - Undecidable

- If Falsified, a test case is generated and added to the model harness



The screenshot shows the SLDV Report window with the following content:

Chapter 2. Test/Proof Objectives

Table of Contents

[Status](#)
[Verify True Output](#)

Status

Table 2.1. Objectives Falsified with Counterexamples

#:	Type	Model Item	Description
1	Assert	Assertion	Assertion "Assertion"

Verify True Output

Objectives of: Assertion

#:	Status	Test Cases	Description
1	Falsified	TC 1	Assert

Chapter 3. Test Cases / Counter Examples

Table of Contents

[Counterexample 1](#)

Counterexample 1

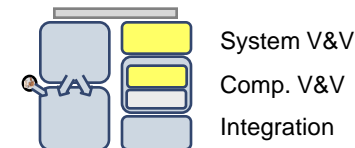
Summary

Length: 0 Seconds (2 sample periods)

Done

Model Coverage

Measure of Test Completeness



- Execution analysis
 - Based on the model structure
 - Dynamic – data collected during simulation
- Coverage results
 - Displayed directly in the model
 - Available in a separate HTML report linked with the model objects
- Supports
 - Simulink
 - Stateflow
 - Embedded MATLAB

Transition "TRD1_act[RD1_act]" from Junction #12 to "active"

Params: mode: logic: test: harness: Coverage Report - ModelA.flexflow

Overview Links: [mode: logic: test: harness: Mode Logic: SSMode Logic: Chart: Actuators: RD1.L](#)

Metric	Coverage
Cyclomatic Complexity	2
Decisions (D1)	100% (1/1) decisions: not covered
Conditions (C1)	75% (3/4) conditions: not covered
MC/DC (C1)	50% (1/2) conditions: reversed the outcome

Decisions analyzed:

Transition trigger expression	True	False
false	4135	22
true	124157	0

Conditions analyzed:

Description	True	False
Condition 1, "RD1_act"	4135	22
Condition 2, "LD_act"	0	4135

MC/DC analysis (recombination in parentheses did not occur):

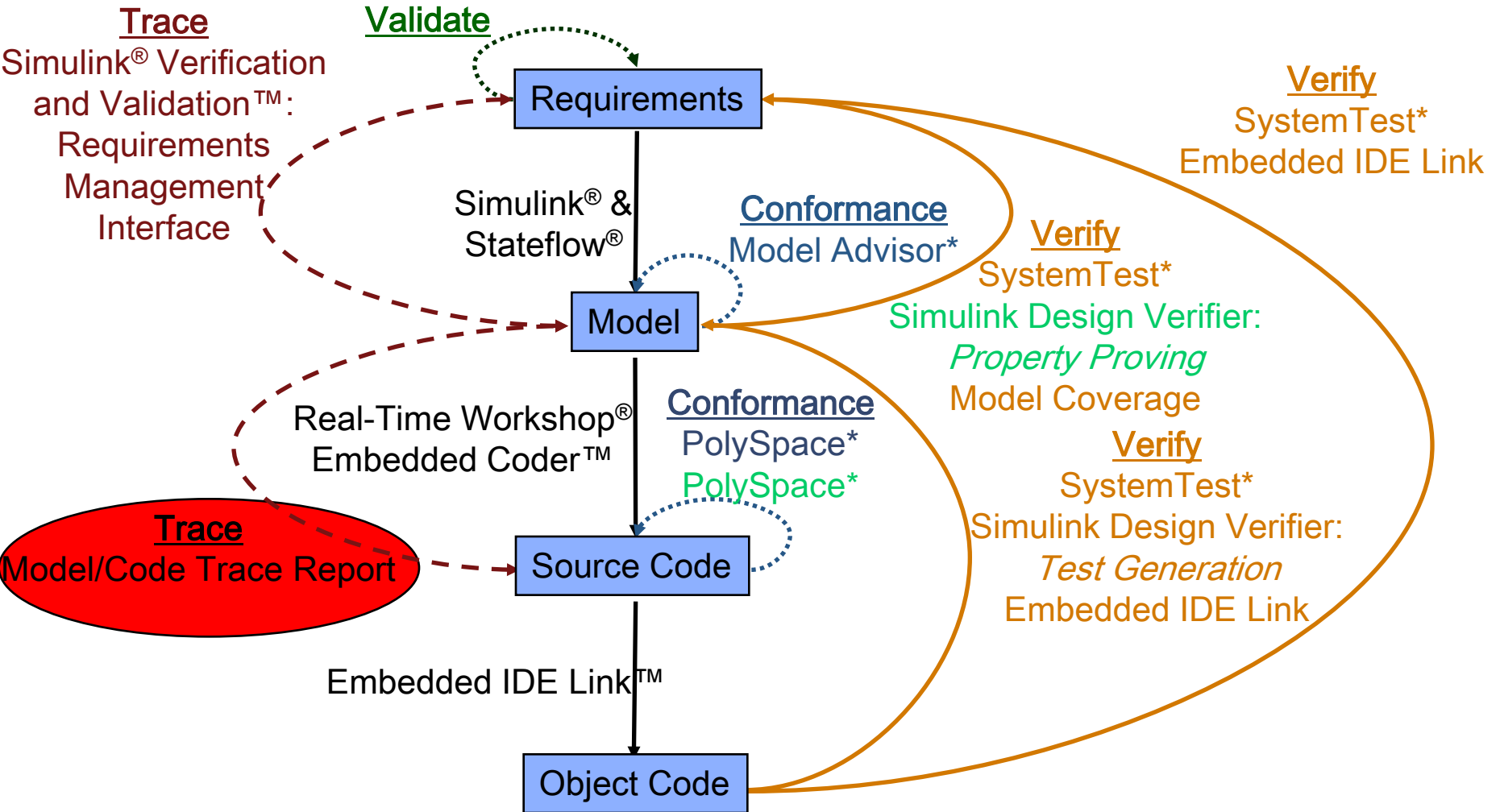
Decision Condition	True Out	False Out
Transition trigger expression		
Condition 1, "RD1_act"	TF	TF
Condition 2, "LD_act"	(TT)	TF

Full decision coverage. Condition "LD_act[]" has never been true.

- Decision coverage
- Condition coverage
- MC/DC
- Lookup table coverage
- Signal range coverage

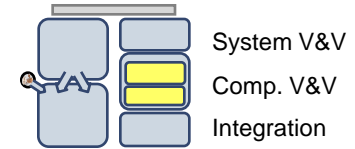
Supported coverage types

Workflow Example



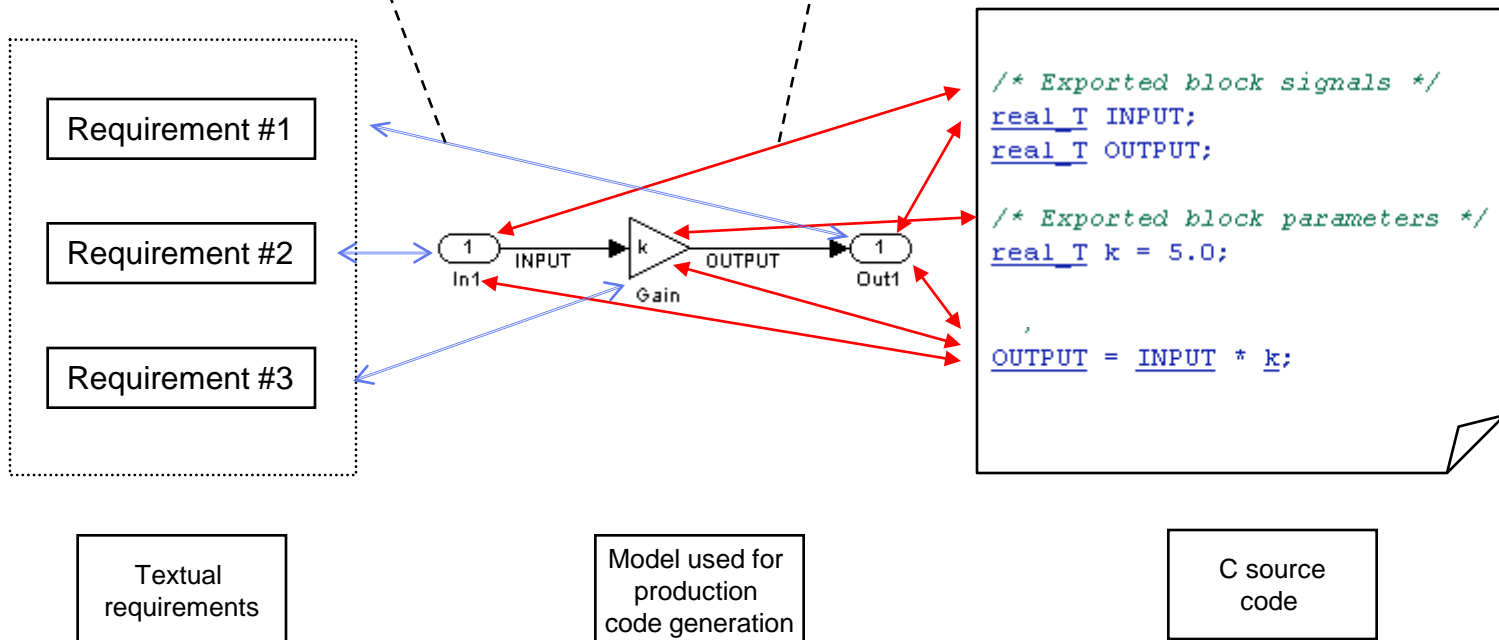
* DO-178B Qualifiable Tool

Model-to-Code and Code-to-Model Traceability



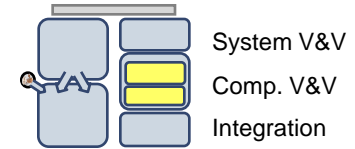
- Use Simulink Verification and Validation software to navigate and trace between model elements and requirements.

- Use Real-Time Workshop Embedded Coder software to navigate and trace between generated code back and its source model.



Traceability Report

Real-Time Workshop Embedded Coder



- Use the Traceability Report section of the Real-Time Workshop Embedded Coder code generation report to review mapping between model elements and generated code.

Target selection

System target file:

Language:

Documentation and traceability

Generate HTML report Code-to-block highlighting

Launch report automatically Block-to-code highlighting

Real-Time Workshop Report

Contents

[Summary](#)

[Traceability Report](#)

[Subsystem Report](#)

Generated Source Files

[ert_main.c](#)

[rtwdemo_hyperlinks.c](#)

[rtwdemo_hyperlinks.h](#)

[rtwdemo_hyperlinks_private.h](#)

[rtwdemo_hyperlinks_types.h](#)

[rtwtypes.h](#)

Traceability Report

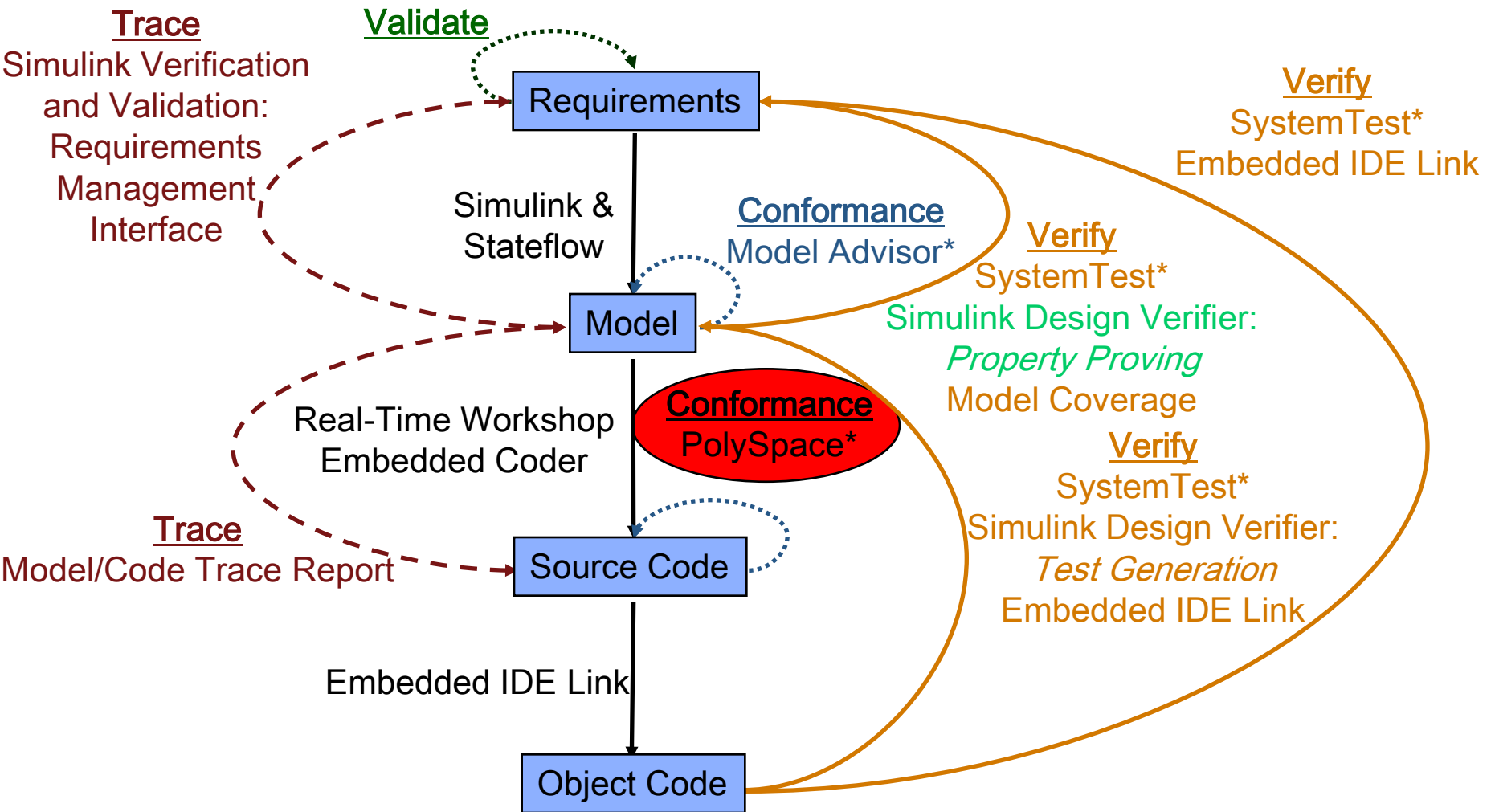
Eliminated / Virtual Blocks

Block Name	Comment
<Root>/Build ERT	Empty SubSystem
<Root>/Mux	Mux
<Root>/Scope	Eliminated unused block
<Root>/View RTW	Empty SubSystem

Traceable Blocks

Block Name	Code Location
<Root>/INC2	rtwdemo_hyperlinks.c:36
<Root>/INC3	rtwdemo_hyperlinks.c:37
<Root>/LIMIT	rtwdemo_hyperlinks.c:44
<Root>/RESET	rtwdemo_hyperlinks.c:54
<Root>/RelOpt	rtwdemo_hyperlinks.c:43
<Root>/Sum	rtwdemo_hyperlinks.c:35
<Root>/Switch	rtwdemo_hyperlinks.c:55

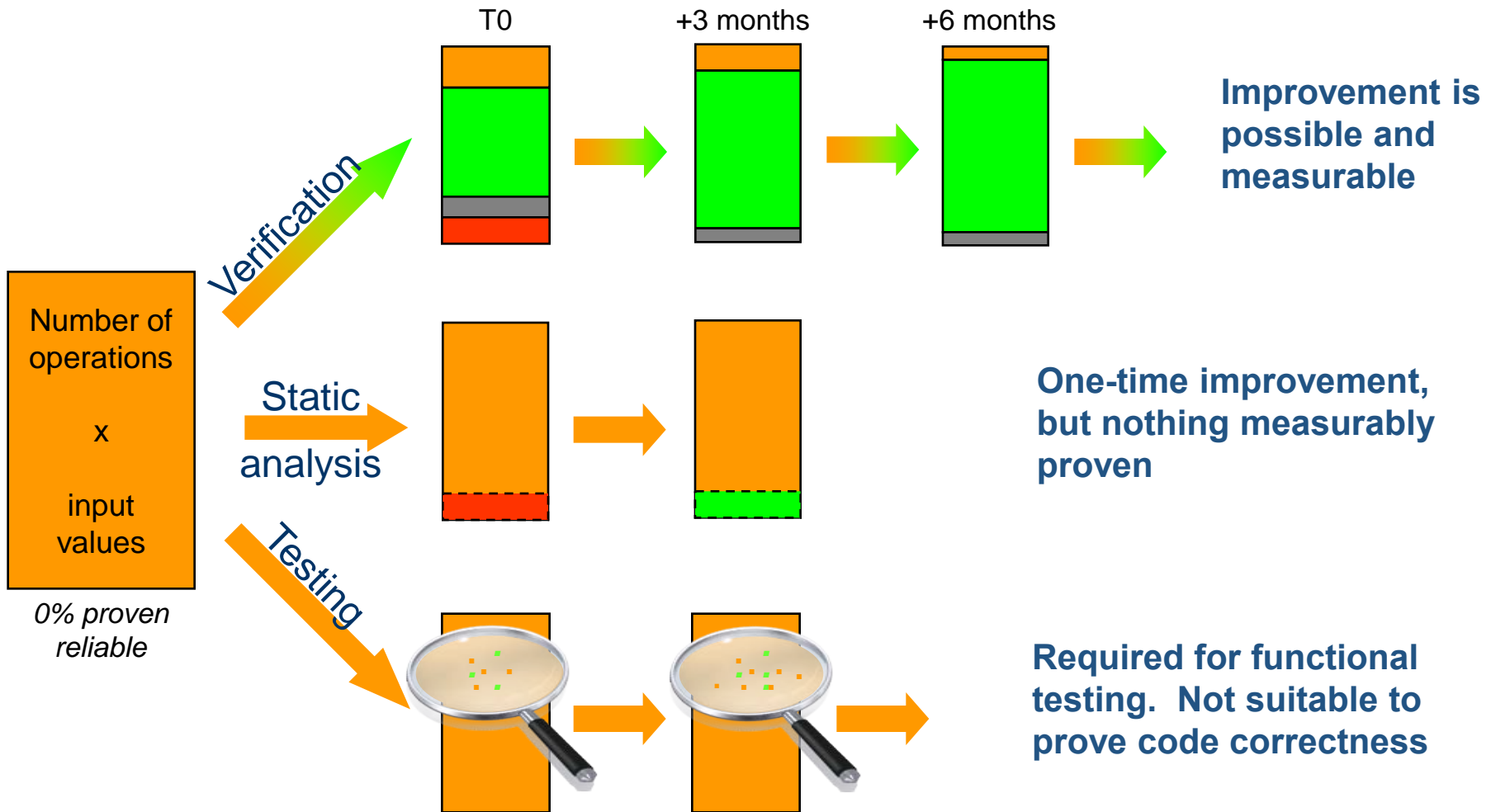
Workflow Example



* DO-178B Qualifiable Tool

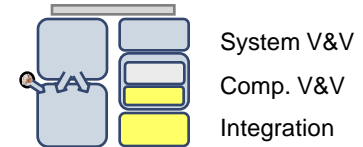
Why prove the absence of run-time errors?

Implications of verification, static analysis & unit testing



Code Correctness

Formal method:
Abstract Interpretation



never

Green
reliable

Red
faulty

Grey
dead

Orange
unproven

```
static void Pointer_Arithmetic ()
{
    int tab[100];
    int i, *p = tab;
```

```
    for(i = 0; i < 100; i++, p++)
        *p = 0;

    if(get_bus_status() > 0)
    {
        if(get_oil_pressure() > 0)
            *p = 5; /* Out of bounds */
        else
            i++;
    }
```

```
    i = random_int();
    if (random_int() > 0) *p = 10;
```

```
    if (0 < i && i <= 100)
    { p = p - i;
      *p = 5; /* Safe pointer access */
    }
}
```

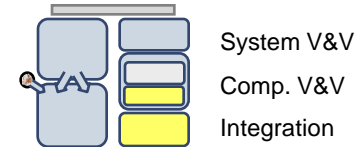
Green
reliable

Green
reliable

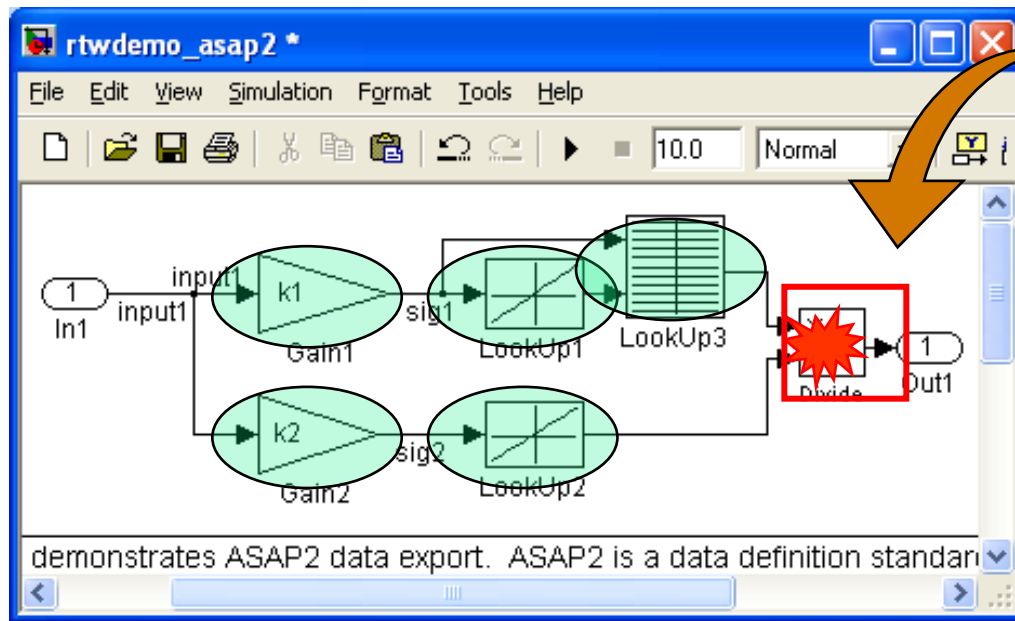
Green
reliable

**Results are proven for
all possible executions of the code!!**

PolySpace Link Solution

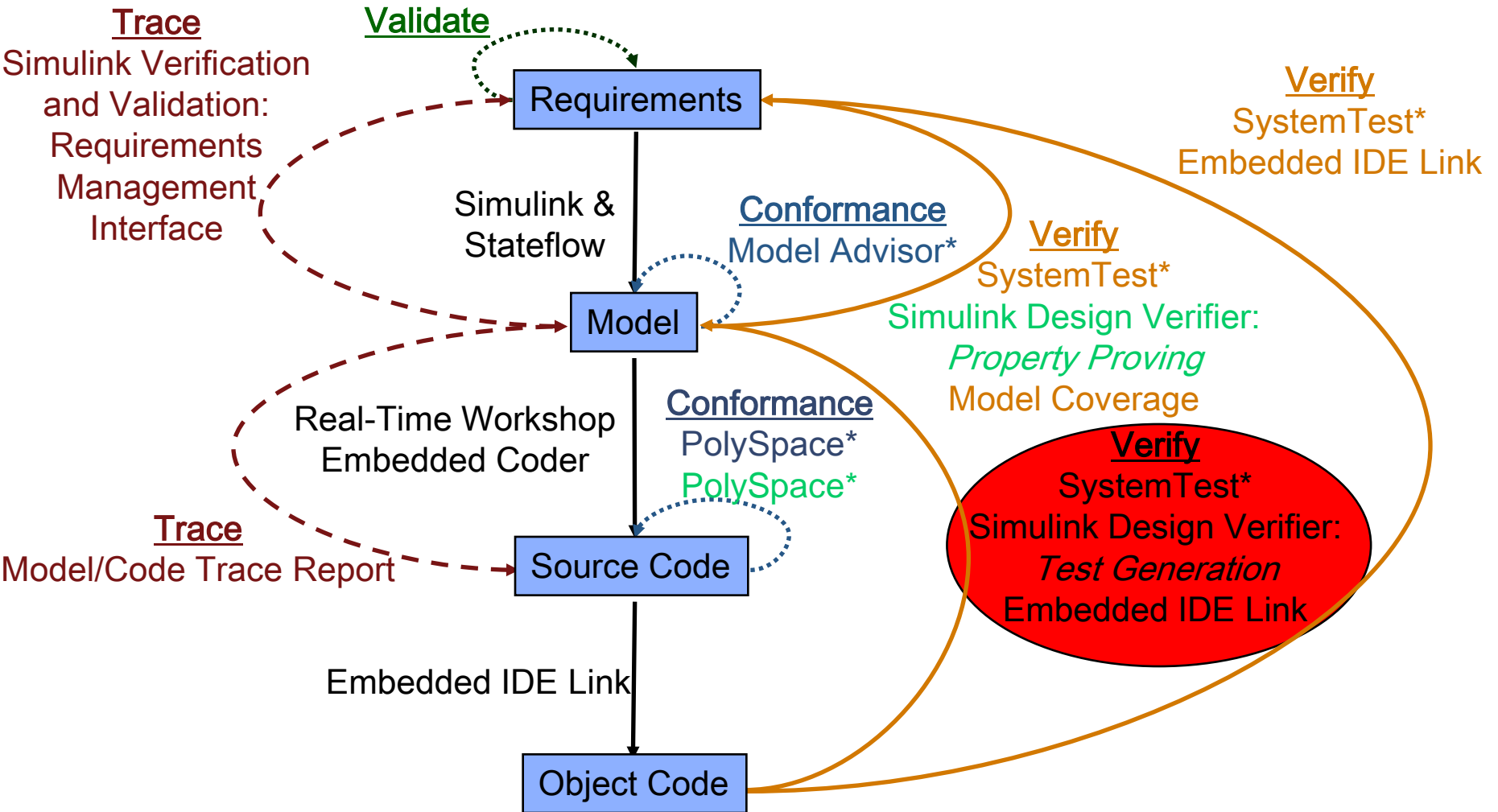


- Trace run-time errors back to the model
- Integrate code verification into the production code generation



PolySpace Viewer - C:\PolySpace_result
File Edit Tools Windows Help
rtwdemo_hyperlinks.c
37 /* Sum: '<Root>/Sum' inc
38 * UnitDelay: '<Root>/
39 */
40 rtb_Switch = (uint8_T) (1
41
42 /* RelationalOperator: '
43 rtb_RelOpt = (rtb_Switch
44
45 /* Outport: '<Root>/Out
46 rtY.Out = rtb_RelOpt;
47
48 /* Switch: '<Root>/Swit
49 if (rtb_RelOpt) {

Workflow Example



* DO-178B Qualifiable Tool

Simulink Design Verifier

Test Generation

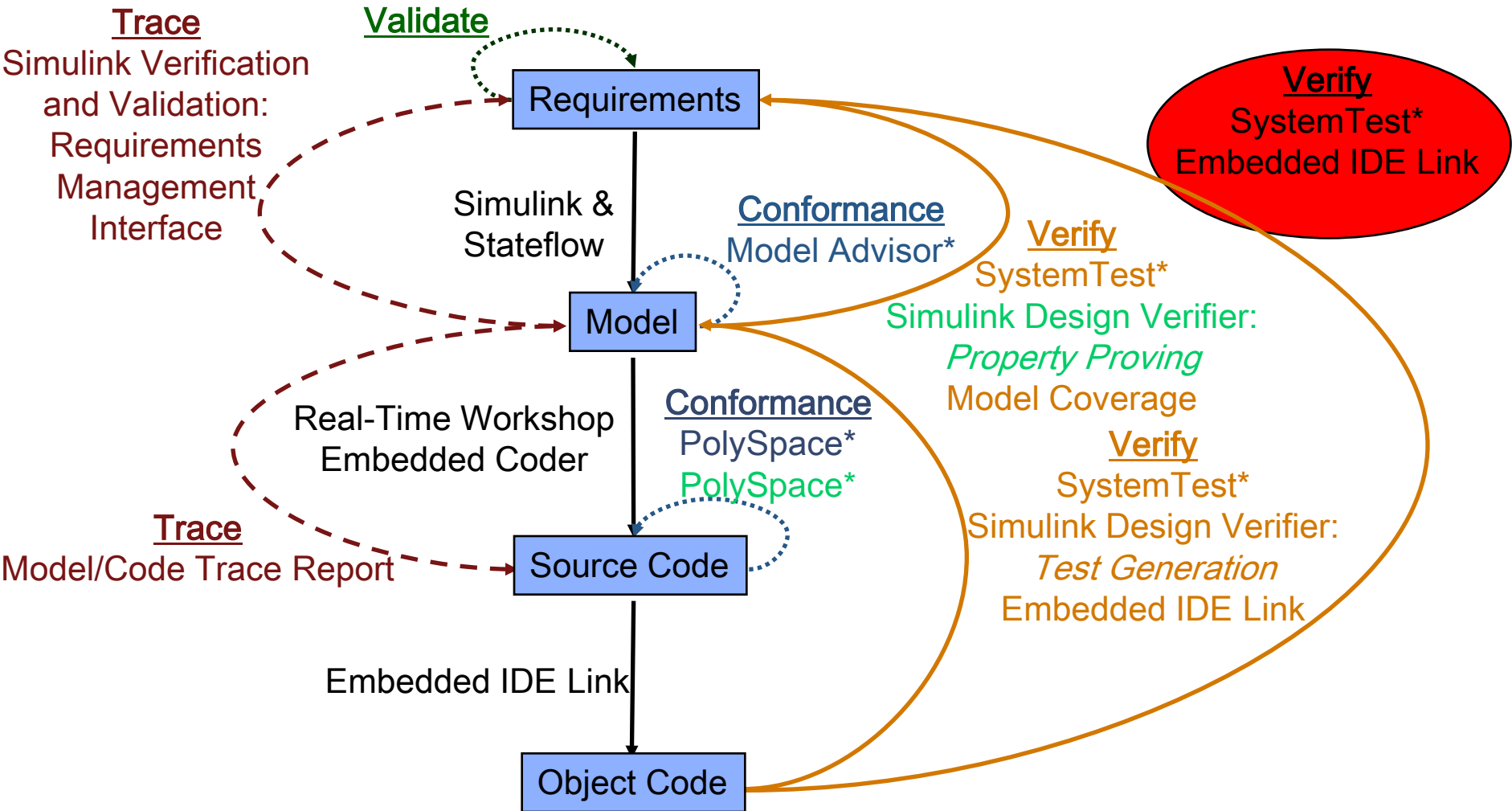
- Automatically generates test vectors to achieve 100% coverage
- Detects unreachable states
- Save test vectors
 - Automatically generate a separate model with test harness
 - Export test vectors to .CSV file
- Automatically generates test vector report
 - Two-way mapping of objectives and generated vectors
 - List of objectives and associated test vector
 - List of test vectors and associated objective(s)

Simulink Design Verifier

Test Generation

- Verify model satisfies requirements
 - Find test vectors for coverage not achieved by functional tests
 - Create unspecified requirement
 - Remove model function not traceable to a requirement
- Verify object code functions according to model
 - Generate test vectors for model coverage
 - Execute test vectors on model
 - Execute test vectors on object code
 - Compare model and code outputs for equality

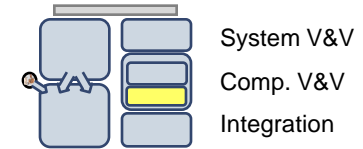
Workflow Example



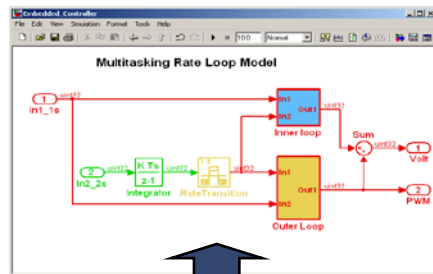
* DO-178B Qualifiable Tool

Processor-in-the-Loop Testing

Embedded IDE Link MU

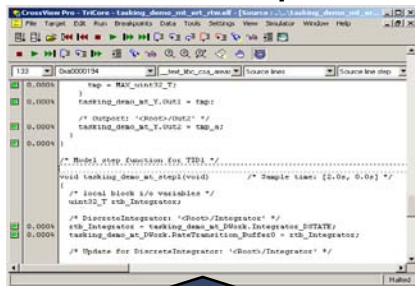


Simulink:

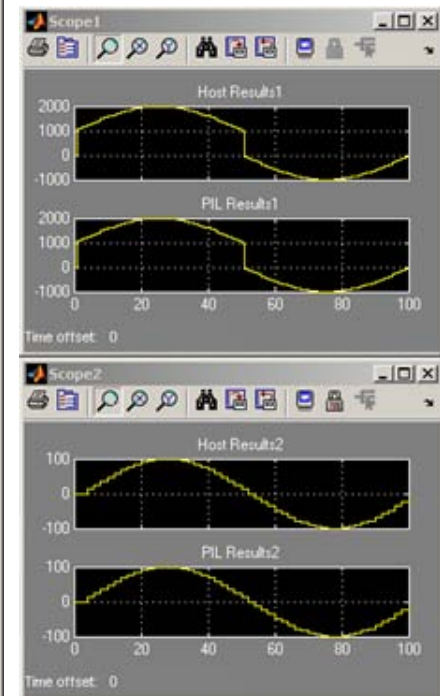
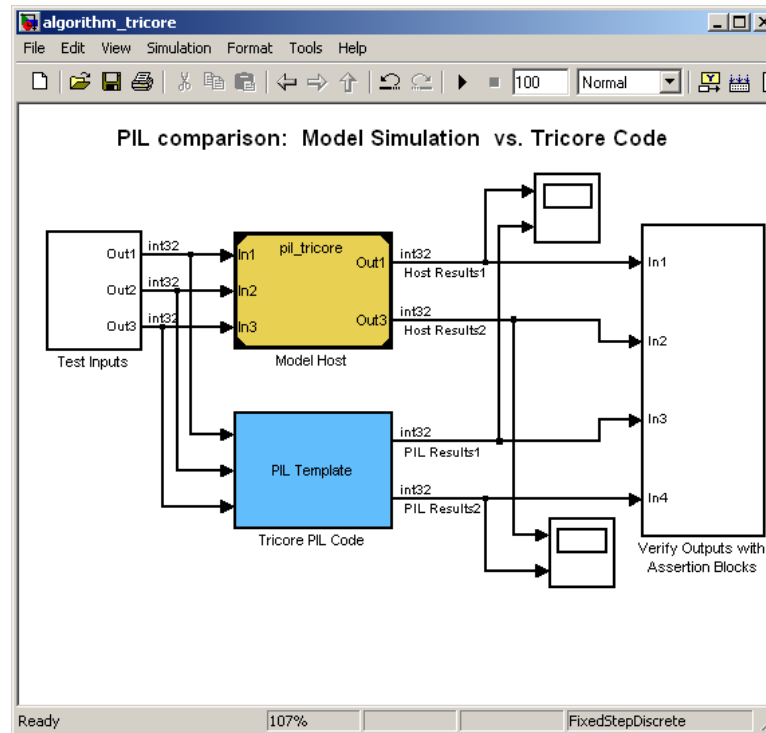


- Model in simulation and code on the processor running in parallel

Real-Time Workshop and TASKING:



ECU:



PIL also provides execution profiling, code coverage reports, and interactive debugging.

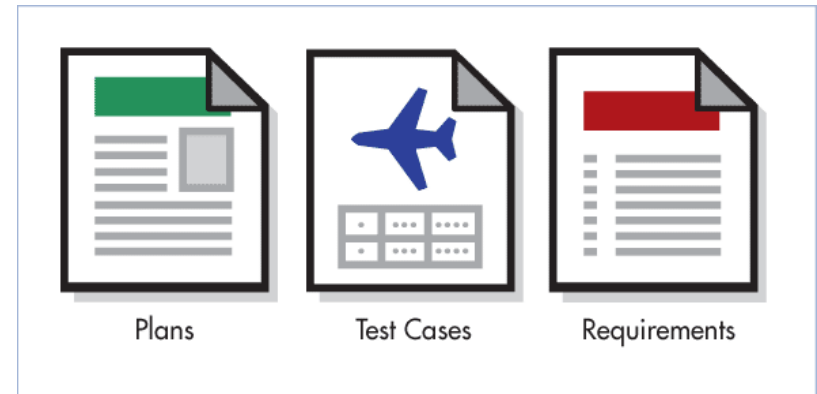
Introduction to DO Qualification Kit

- Provides documentation, test cases, and procedures that help you use Simulink or PolySpace software verification tools for projects based on the DO-178 standard
- Includes tool qualification plans, tool operational requirements, and other materials required for qualifying software verification tools
- Helps streamline certification of your embedded systems developed using Simulink or PolySpace products



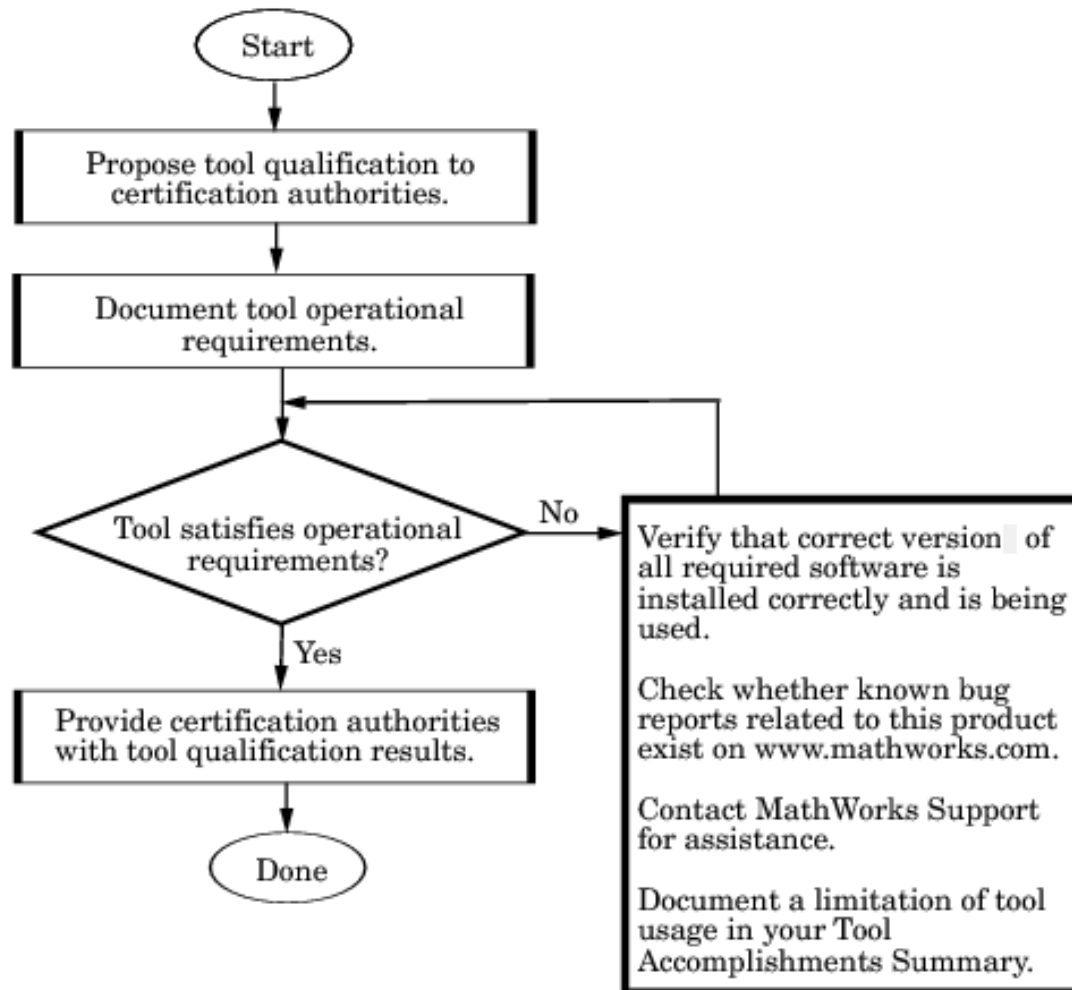
Key Features

- Tool Qualification Plan and Tool Operational Requirements
- Test case models and code, test procedures, and expected results
- Traceability tables mapping test cases to requirements
- Qualification materials for Simulink verification, validation, and test tools
- Qualification materials for PolySpace code verification tools



Working with DO Qualification Kit

To use DO Qualification Kit:



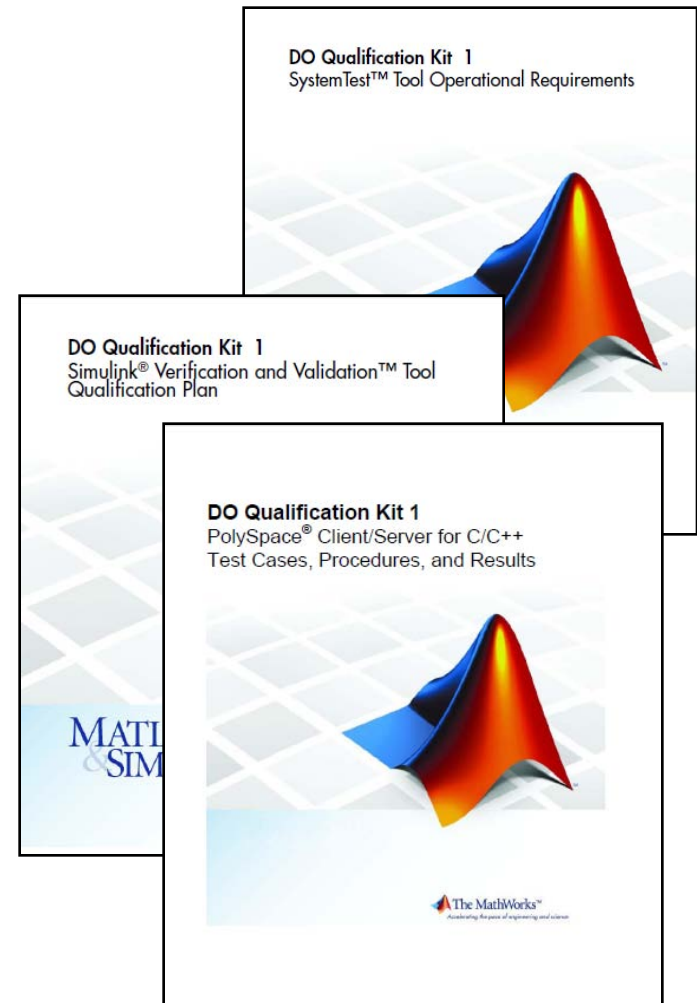
Tool Qualification Plan and Operational Requirements

DO Qualification Kit 1.0 contains qualification artifacts for the following products:

- Simulink Verification and Validation
 - DO-178B Model Checks
- SystemTest
 - Limit Test Element
- PolySpace verification products
 - PolySpace Client for C/C++
 - PolySpace Server for C/C++

Version support includes:

- Release R2008b
- Release R2009a

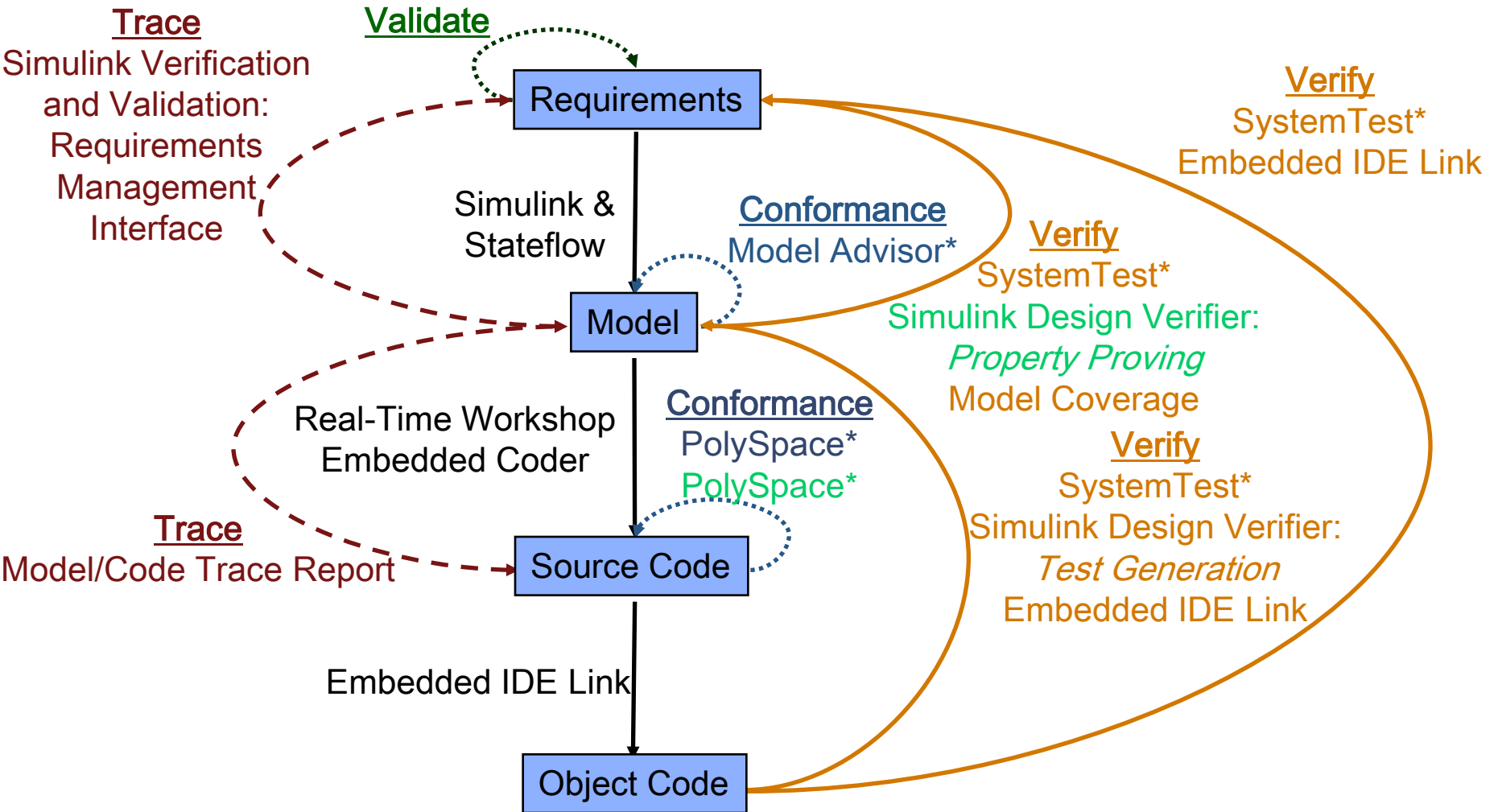


Summary

DO Qualification Kit:

- Eases your embedded system certification process
- Helps satisfy the objectives of verification tool qualification described in DO-178B (Section 12.2)
- Facilitates automated software verification for DO-178
- Enables use of state-of-the-art development tools for Model-Based Design with flight code generation
- Enables qualification of PolySpace products, including formal analysis capabilities

Workflow Summary



* DO-178B Qualifiable Tool