



Oprogramowanie
Naukowo-Techniczne
sp. z o.o.

Kraków, 19.10.2023 r.

MATLAB and Simulink

Model-Based Design for DO-178C/DO-331

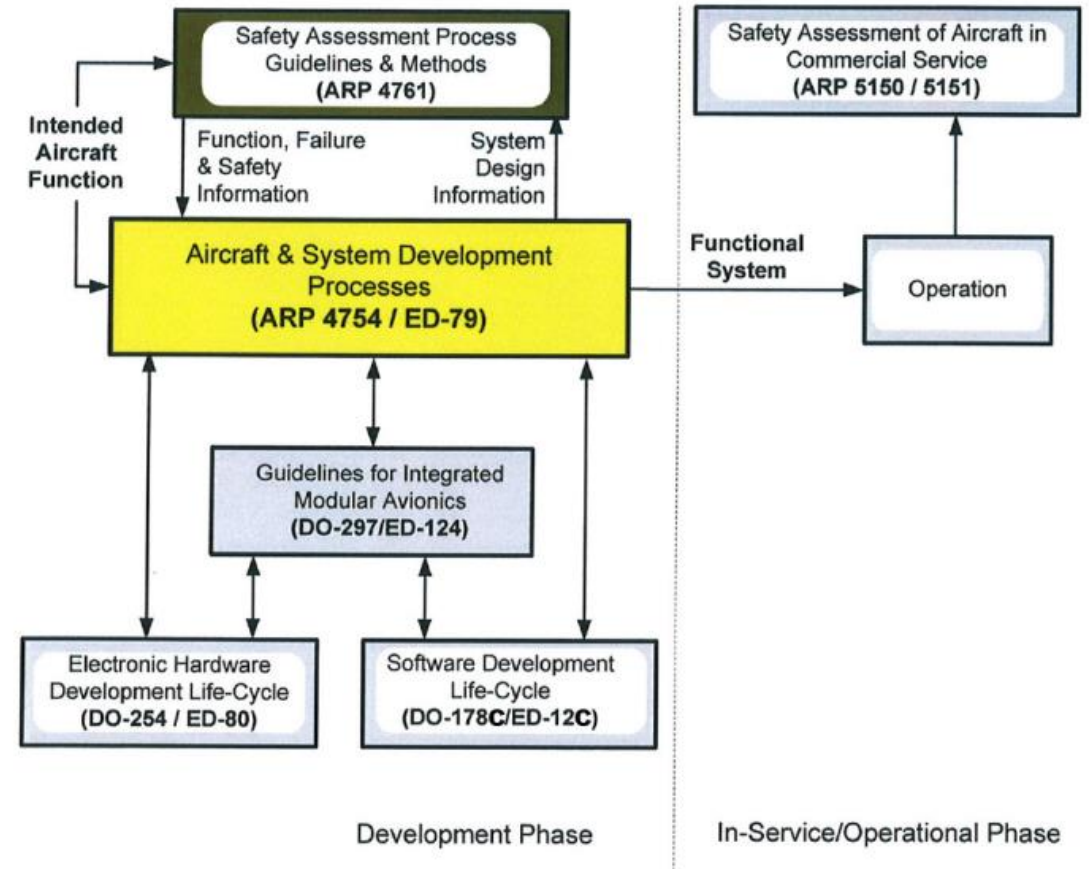
Konrad Kolski, ONT

Agenda

- DO-178C/DO-331 introduction and Lifecycle Overview
- System and Software Requirements
- Software Design Verification
- Source Code Generation and Verification
- Low-Level Test Generation
- Documentation and Analysis

System, Software and Hardware Life Cycles

- DO-178C/DO-331 is one part of a large development and verification process
 - Airborne software considerations
- ARP 4754A: Airborne systems-level guidance
- DO-254: Complex electronic hardware considerations



DO-178C Objectives and Independence

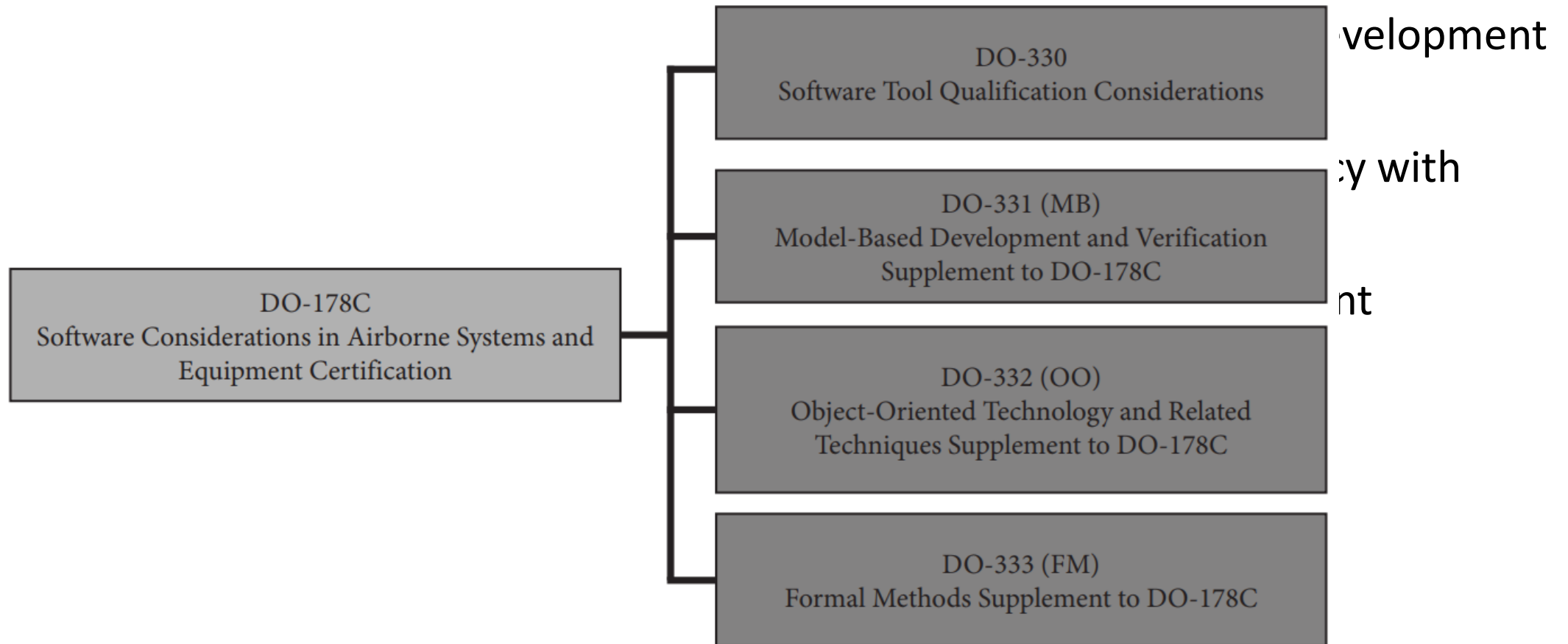
- Independence – separation of development and verification responsibilities

DO Table	Description	Software Life Cycle Process	Number of Objectives
A-1	Software Planning Process		7
A-2	Software Development Processes		7
A-3	Verification of Outputs of Software Requirements Process	Software Verification Processes	7
A-4	Verification of Outputs of Software Design Process		13
A-5	Verification of Outputs of Software Coding & Integration Processes		9
A-6	Testing of Outputs of Integration Process		5
A-7	Verification of Verification Process Results		9
A-8	Software Configuration Management Process	Integral Processes	6
A-9	Software Quality Assurance Process		5
A-10	Certification Liaison Process		3

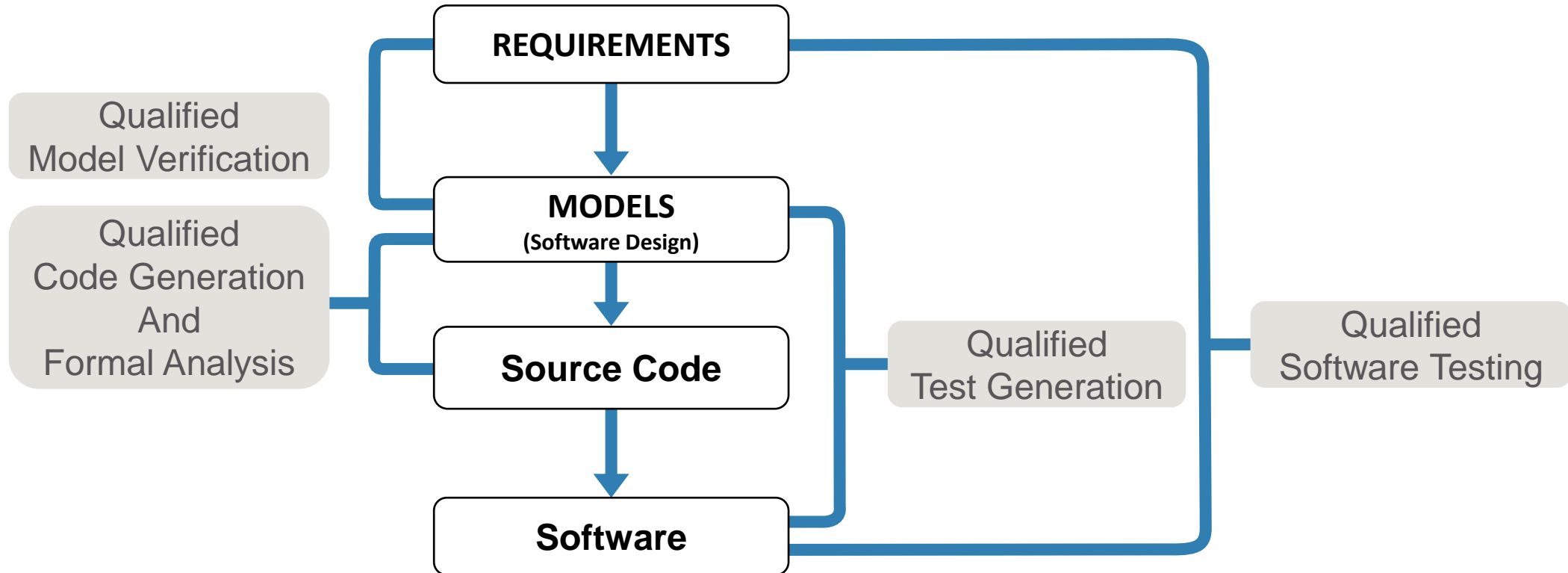
DO-178C Supplements

DO-331 Key Aspects

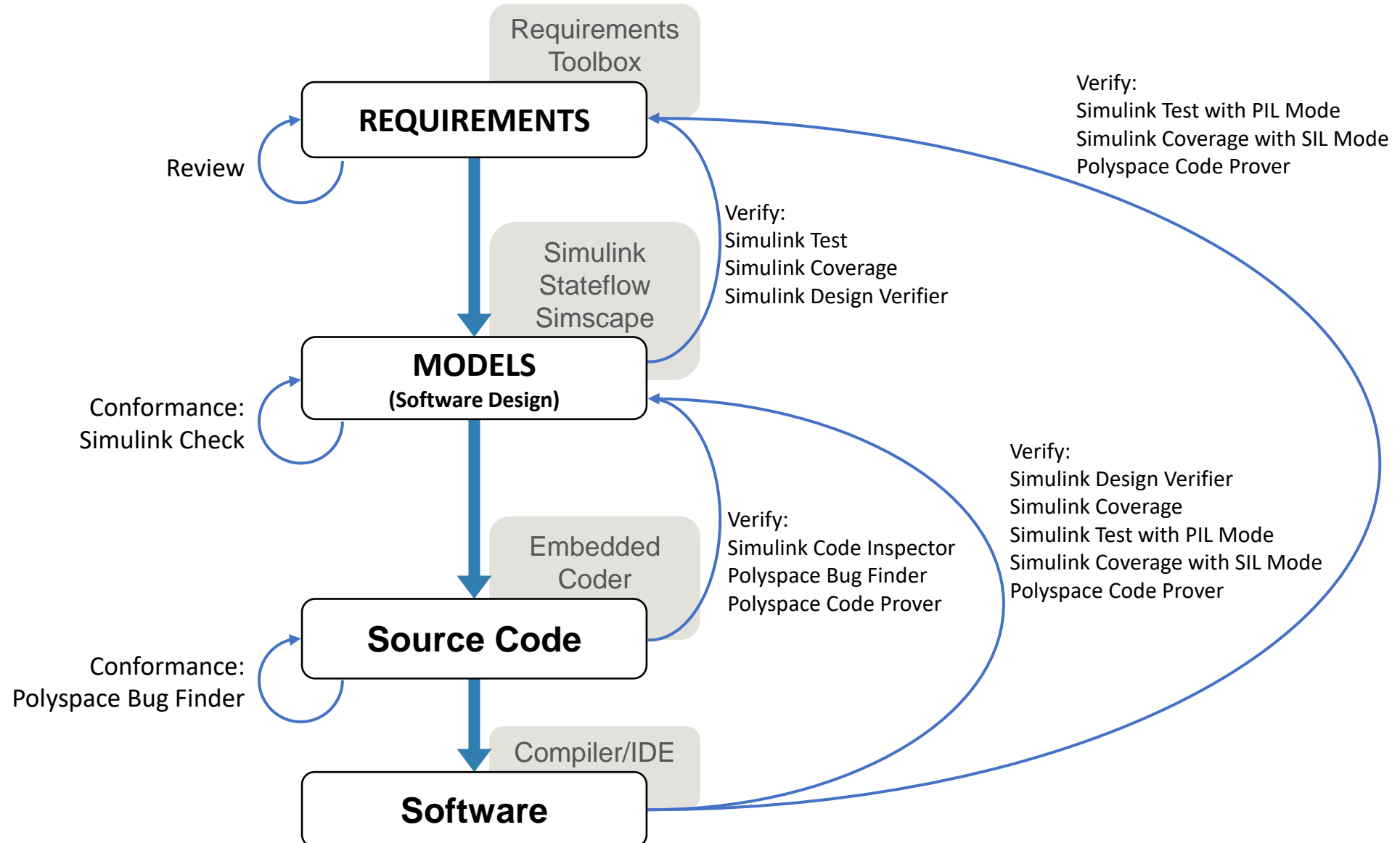
- Develop process
- Verification requirements
- Qualification



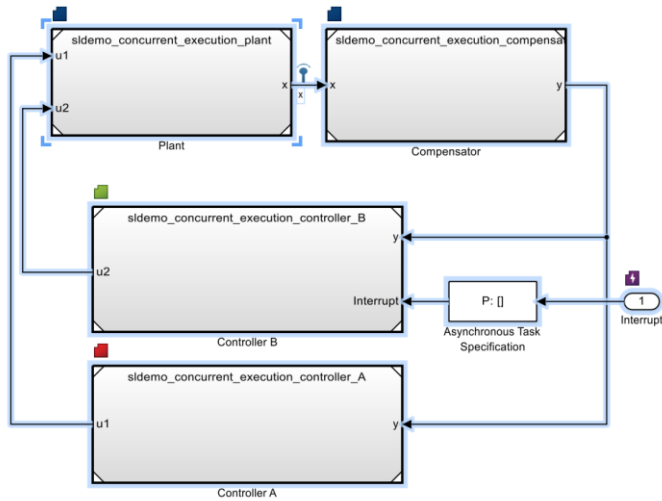
Lifecycle Overview



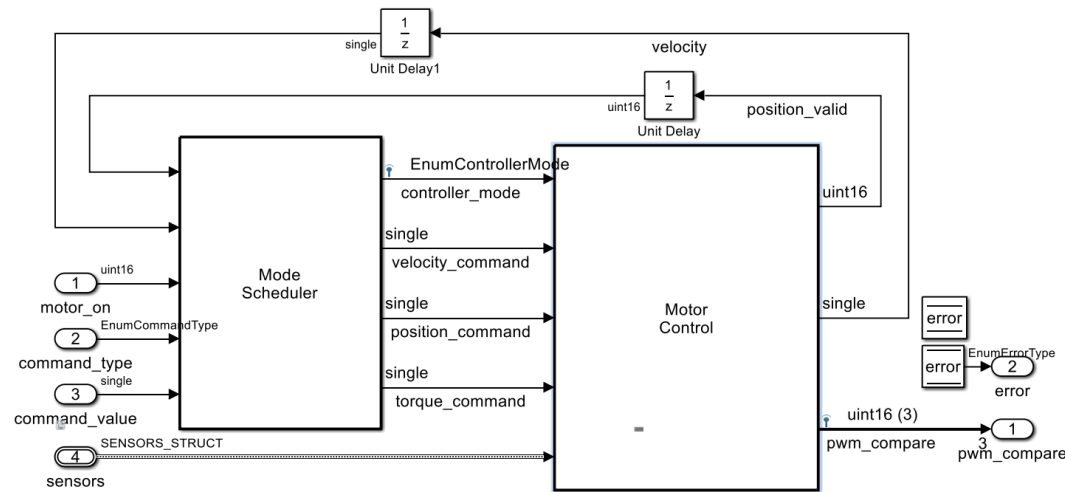
DO-178C/DO-331 with MathWorks tools



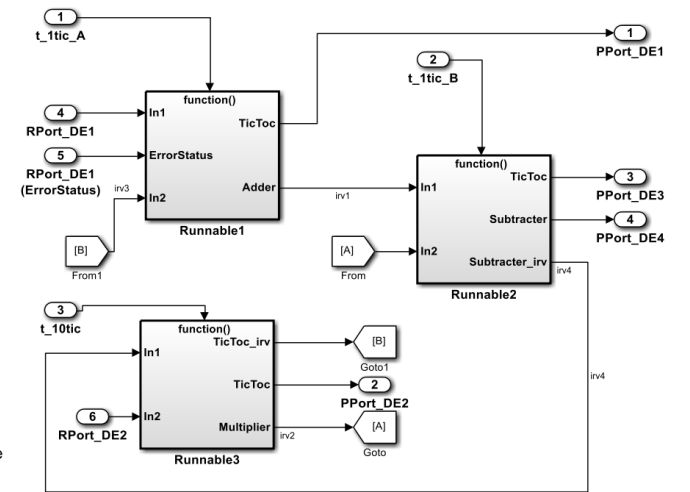
Develop detailed software design with Simulink



Model References

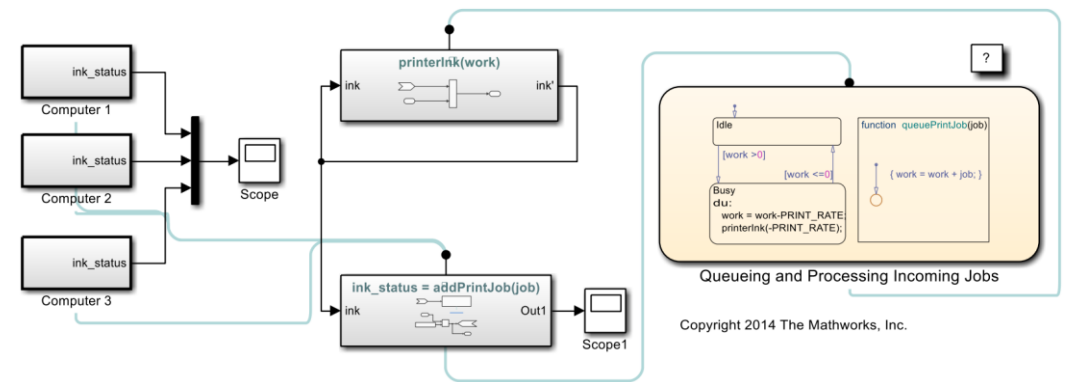


Atomic Subsystems



Function Call Subsystems

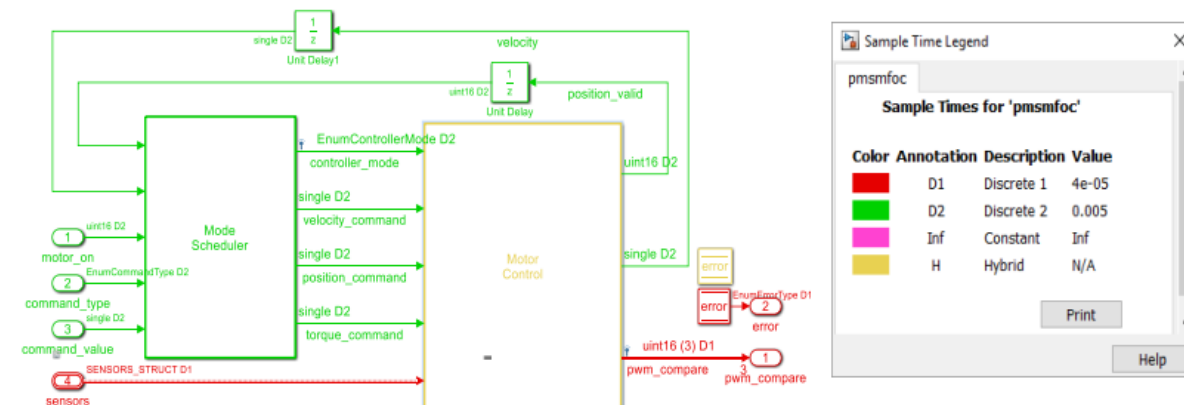
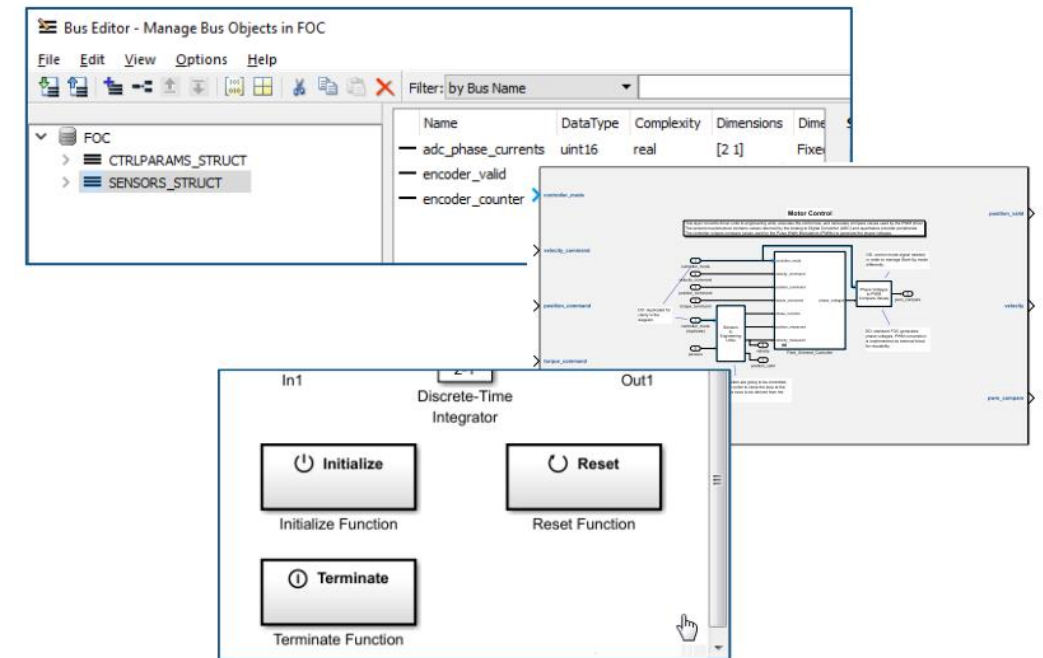
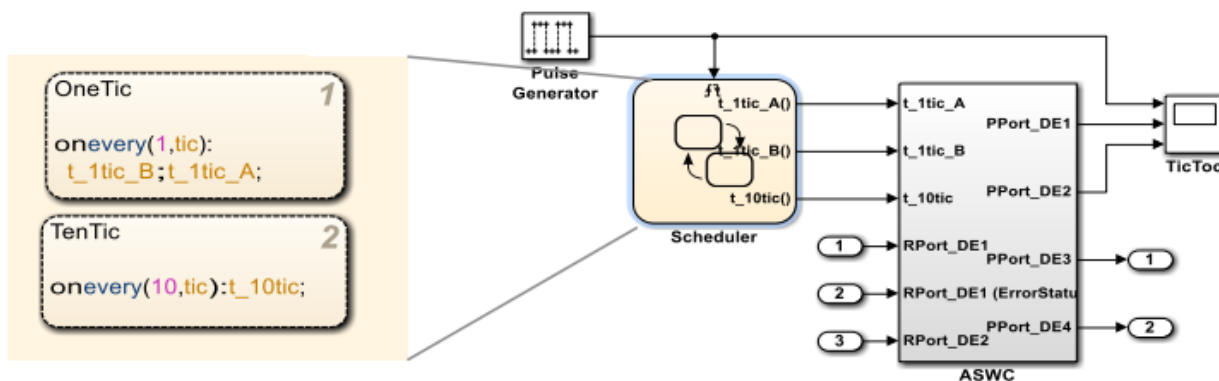
- Functional decomposition
 - Model Reference Blocks
 - Atomic Subsystems
 - Function-Call Subsystems
 - Simulink Functions



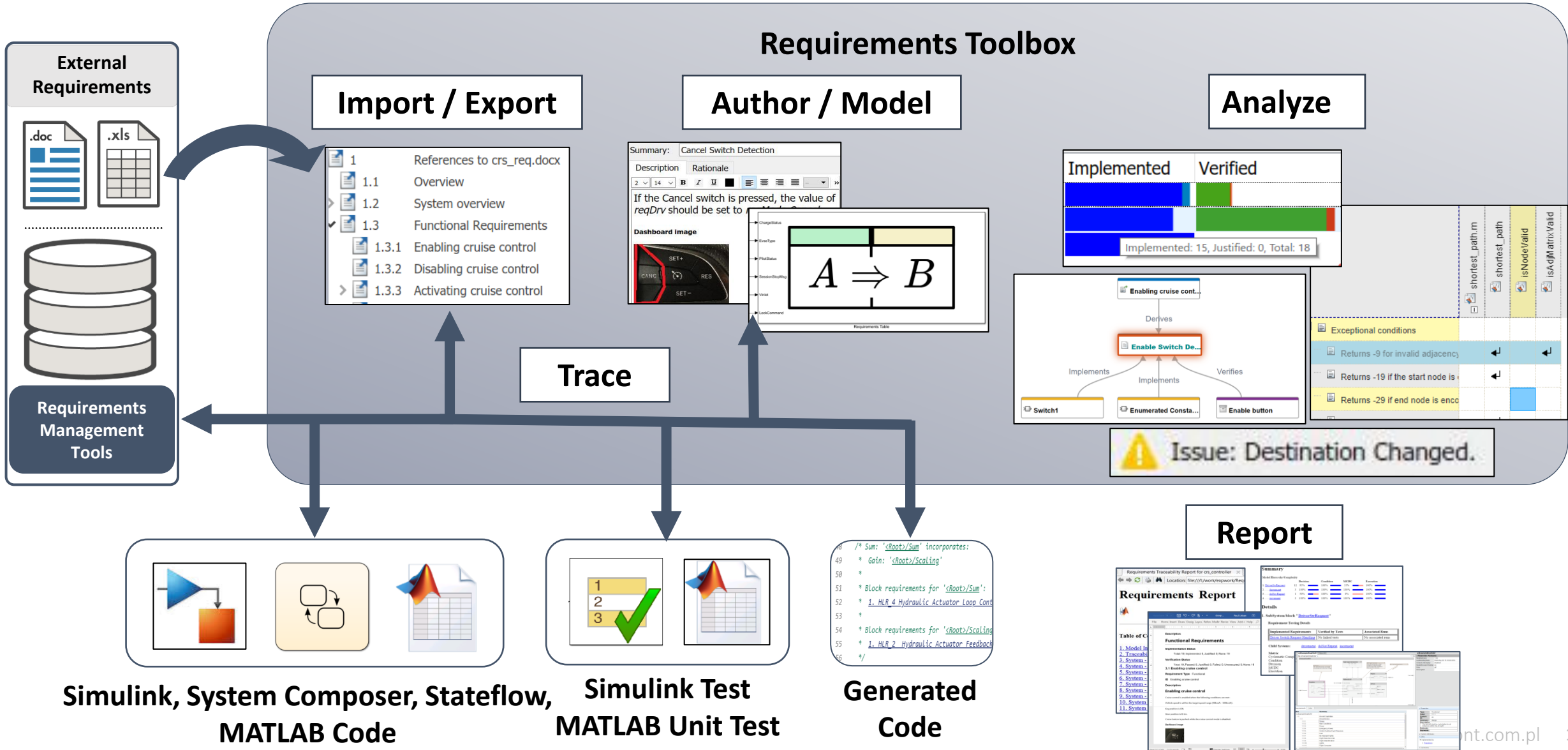
Simulink Functions

Develop detailed software design with Simulink

- Define signals with data dictionaries
- Switch to interface view
- Specify operating modes: Initialize, Reset, Terminate
- Single, multi-rate/task modeling of dynamic behavior
- Explicitly model timing with state machines



Requirements Toolbox – Authoring, Linking, Validating for design and tests

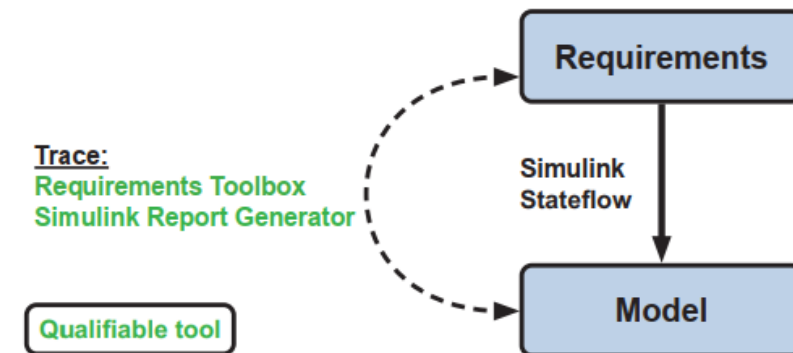


System and Software Requirements - Traceability

- Requirement linking between model and software requirements and test cases
- Perform traceability analysis to detect gaps in testing and coverage
- Generate System Design Description (SDD) reports to demonstrate completeness of traceability mapping
 - DO-331 table MB.3/MB.4

DO Table	Description
A-2	Software Development Processes
A-3	Verification of Outputs of Software Requirements Process
A-4	Verification of Outputs of Software Design Process
A-7	Verification of Verification Process Results

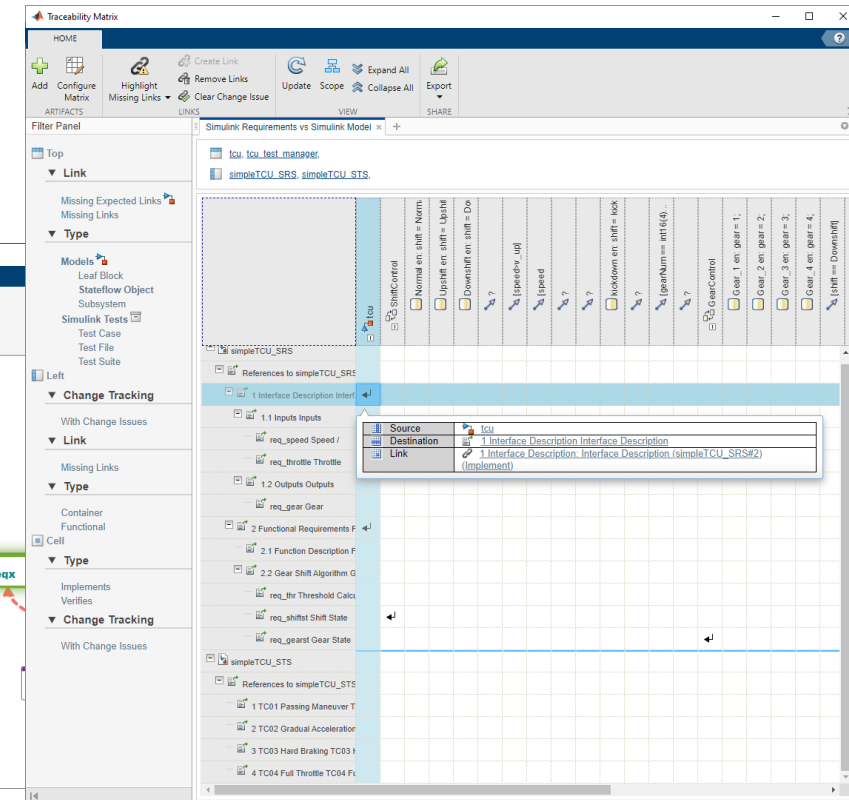
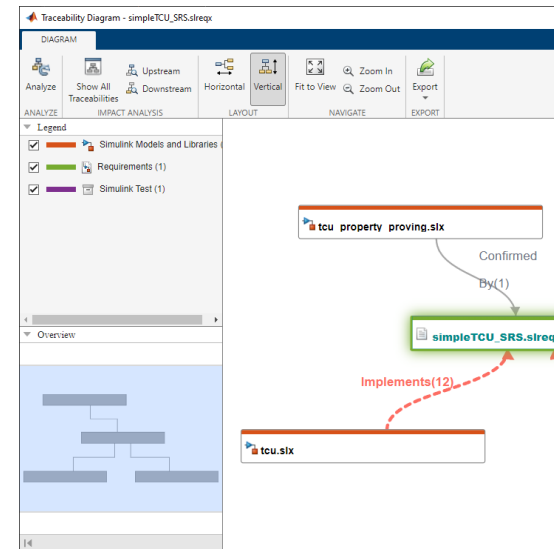
*„Low – level requirements comply with high-level requirements
 Low-level requirements are verifiable
 Low-level requirements are traceable to high-level requirements”*



Review and Analyze Traceability

1. Traceability Matrix
2. Traceability Diagram
3. Traceability Report

- Review links between requirements, model and tests
- Directly add links to address gaps
- Display items linked to a requirement, block or test
- Assess how a change impacts upstream and downstream items



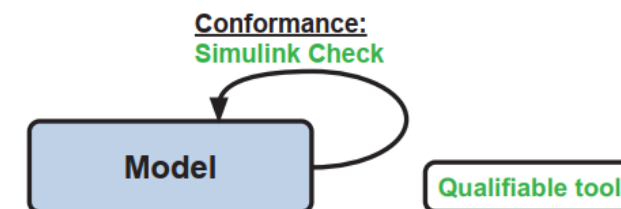
Comparable artifacts:

- Requirement Sets
- Simulink and System Composer models
- Simulink Test test files
- Simulink data dictionaries
- MATLAB M-files

Modeling Standards Compliance

- Improves implementation and adherence to modeling standards
- Provides checks for High-Integrity Systems i.e.
 - Prevents non-deterministic modeling constructs
 - Helps avoiding unsupported modeling constructs
 - Facilitates proper requirements implementation
- Satisfies objectives in *DO-331 table MB.A-3: Low-level requirements conform to standards*

DO Table	Description
A-4	Verification of Outputs of Software Design Process



Model Advisor – three step solution

- Modeling Standards for DO-178C/DO-331
 - Display model version information
 - High-Integrity Systems
 - Simulink
 - Check usage of Abs blocks
 - Check usage of remainder and reciprocal
 - Check usage of square root operations
 - Check usage of log and log10 operations
 - Check usage of While Iterator blocks
 - Check usage of For and While Iterator sut

Detect errors or potential issues



Identify Abs blocks that have unreachable code or produce overflows.

Warning

The following Abs blocks can cause unreachable code or produce overflows:

- [sl_example_HI/Abs](#)

Recommended Action

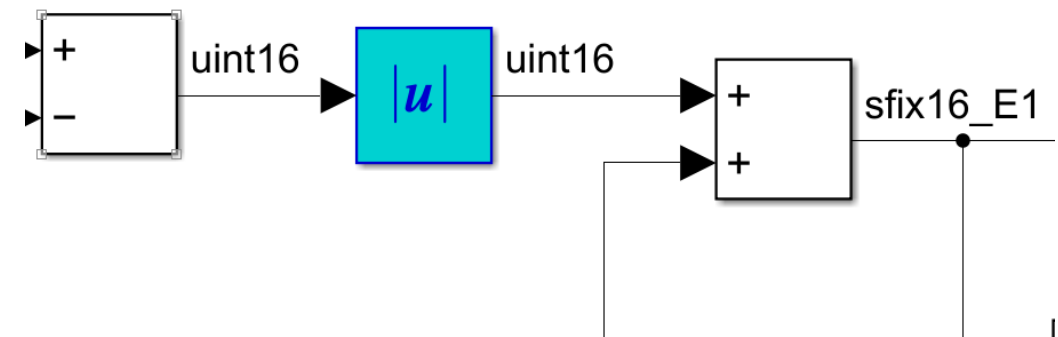
Change the inputs of the Abs blocks identified in the results to signed input data types, or consider removing them.

Guidance provided to address issues



Identify Abs blocks that have unreachable code or produce overflows.

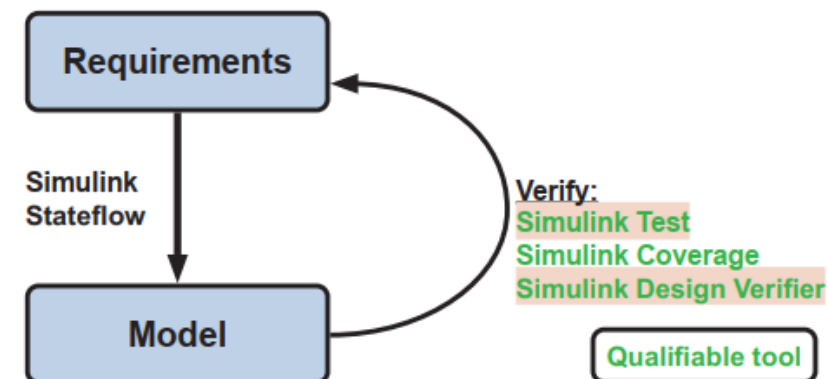
Navigate to problematic blocks



Model Verification and Design Error Detection

- Construct test cases to verify specific software high-level requirements
- Simulate in many configurations (normal, SIL, PIL and so on)
- Find design errors before creating source code
- Prove that algorithm works as intended, requirements are consistent and verifiable to satisfy objectives from *DO-331 table MB.3/MB.4*







DO Table	Description
A-4	Verification of Outputs of Software Design Process
A-6	Testing of Outputs of Integration Process
A-7	Verification of Verification Process Results



Simulink Test capabilities

Test Case

Inputs

-  baseline
-  in_A
-  in_B
-  out
-  time
-  timeseries



Excel file

Workspace



MAT File

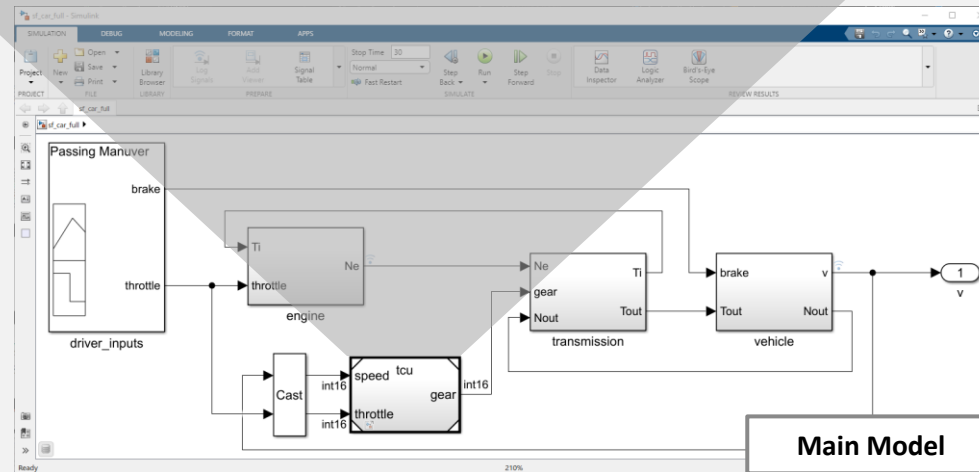
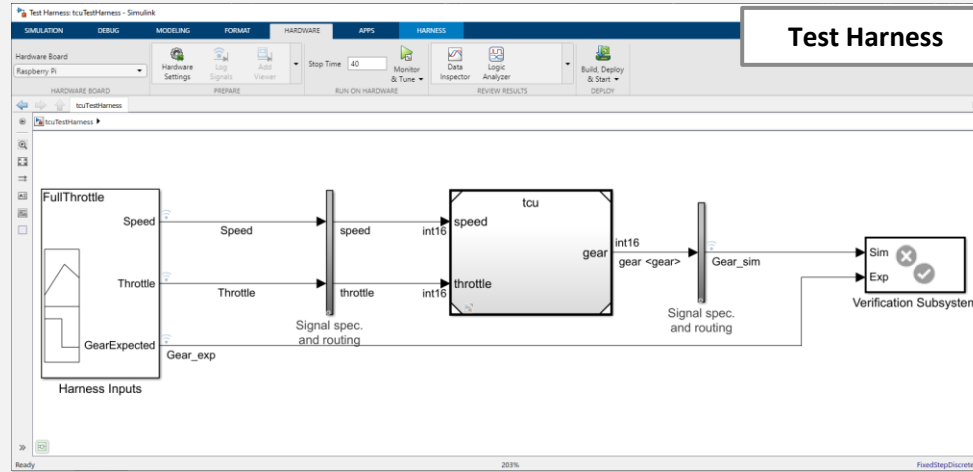


Simulink blocks



Signal Editor/Builder

and more!



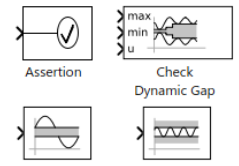
Assessments



Excel file (baseline)



MAT file (baseline)



Simulink blocks

```

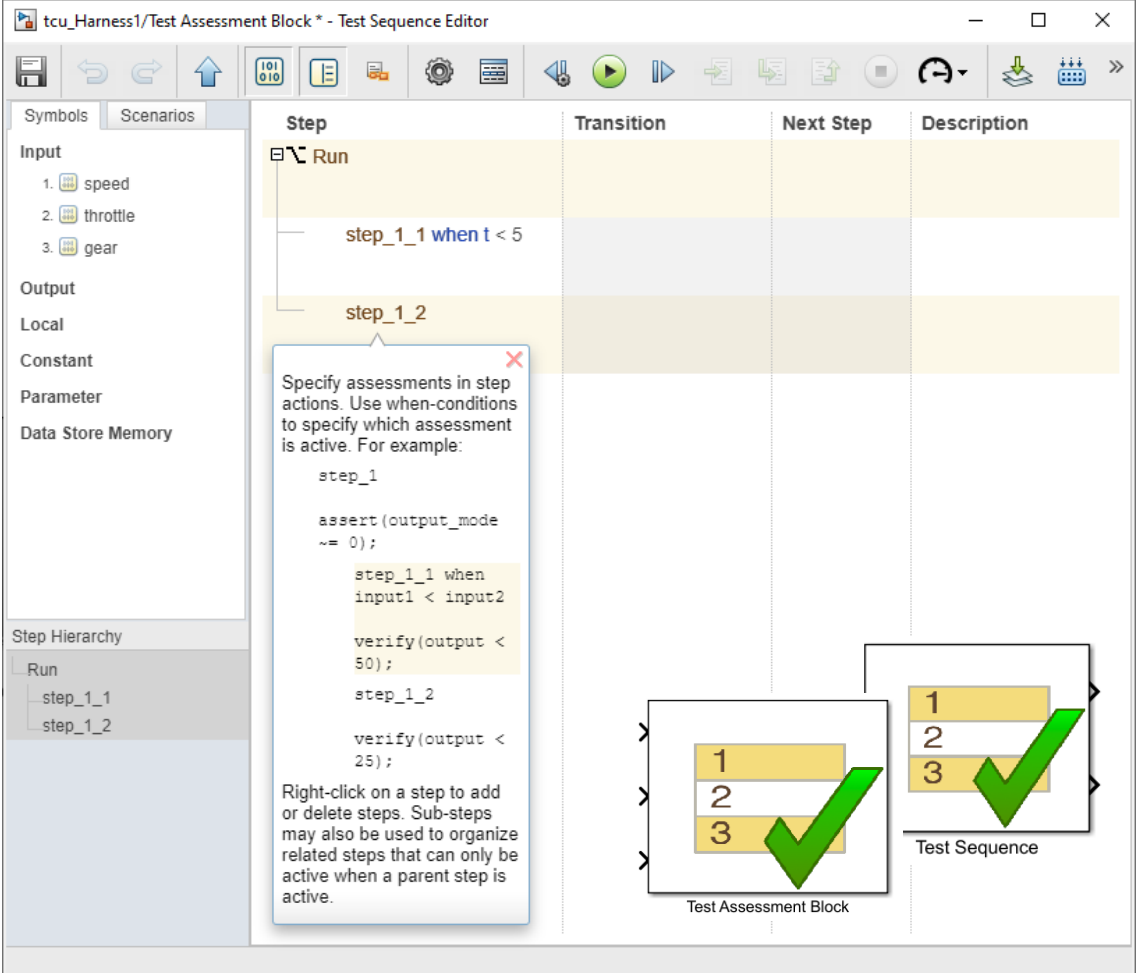
function customCriteria
Perform custom criteria
test.verifyThat(test.s
  
```

MATLAB function

and more!

Simulation-time dynamic testing

- State-machine based test sequence and assessments for logic based testing
- Define testing sequence using logic and temporal operators
- Source test data from sequence and use live simulation data for pass/fail assessment
- Allows closed-loop testing with test feedback during simulation
- Interactive testing with own inputs, outputs, parameters and local data



The screenshot shows the 'Test Sequence Editor' window for 'tcu_Harness1/Test Assessment Block'. The interface includes a toolbar, a left sidebar with 'Symbols' and 'Scenarios' tabs, and a main workspace with a table of test steps.

Step	Transition	Next Step	Description
Run			
step_1_1	when t < 5		
step_1_2			

A tooltip is displayed over the 'step_1_2' cell, providing instructions on how to specify assessments in step actions. The tooltip text is as follows:

```

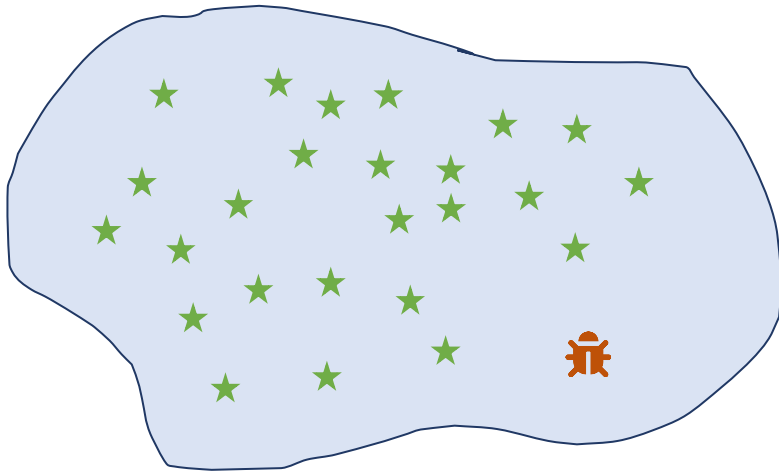
Specify assessments in step actions. Use when-conditions to specify which assessment is active. For example:
step_1
  assert(output_mode
    ~= 0);
  step_1_1 when
    input1 < input2
  verify(output <
    50);
  step_1_2
  verify(output <
    25);
  
```

Right-click on a step to add or delete steps. Sub-steps may also be used to organize related steps that can only be active when a parent step is active.

At the bottom right, there are two diagrams illustrating the 'Test Assessment Block' and 'Test Sequence'. The 'Test Assessment Block' diagram shows a vertical list of steps 1, 2, and 3, with a large green checkmark indicating a successful assessment. The 'Test Sequence' diagram shows a similar vertical list of steps 1, 2, and 3, also with a large green checkmark, indicating a successful sequence execution.

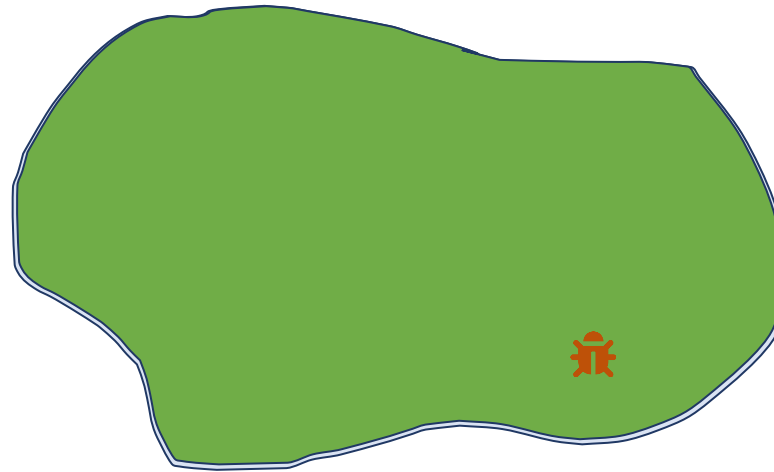
Simulation-Based Testing vs Formal Verification

Design Space



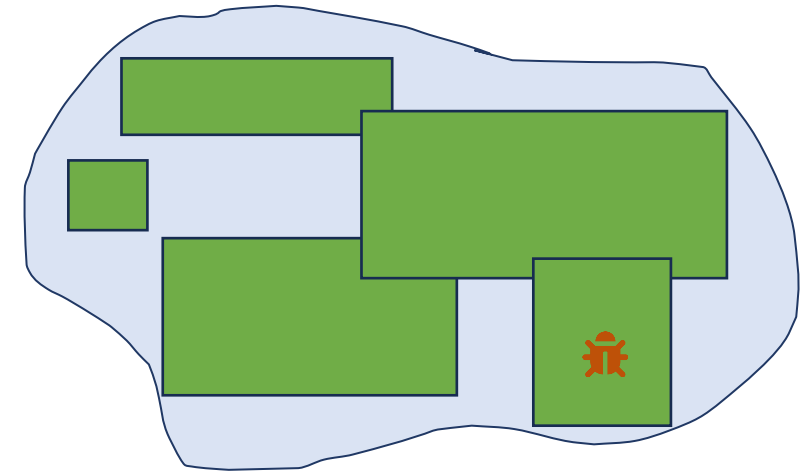
Testing

Point Coverage
through simulation



Formal Verification (Ideal)

Complete coverage of design
space (formal proof)



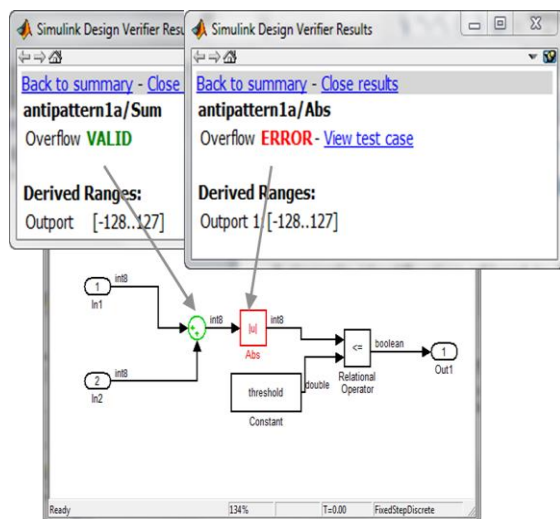
Formal Verification (Real-World)

Real-World application of formal
verification

Simulink Design Verifier

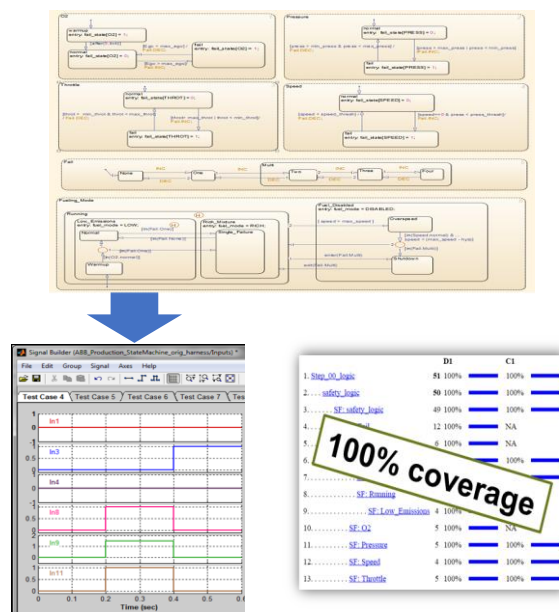
Design Error Detection

Uncover hard to find dead logic and design flaws



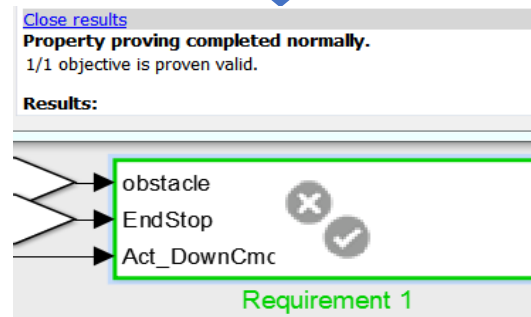
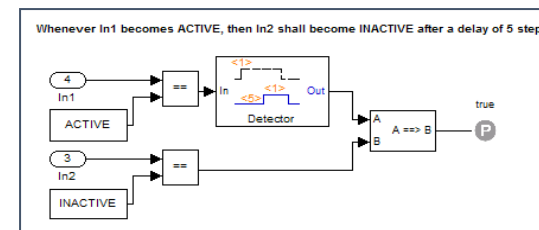
Test Generation

Automate test vector generation to analyze missing coverage



Property Proving

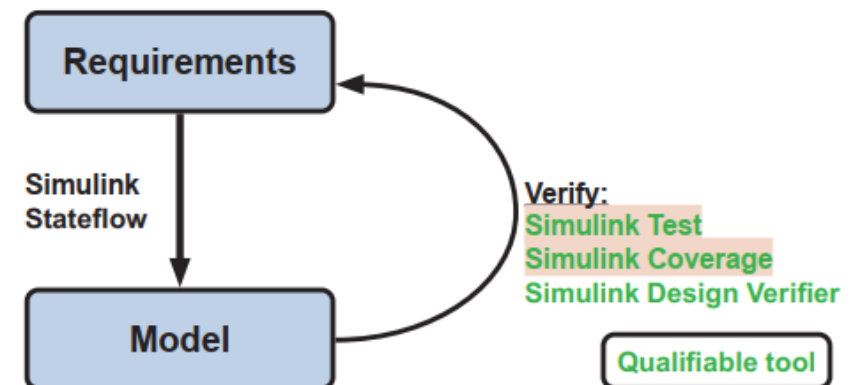
Prove design meets requirements



Model Coverage

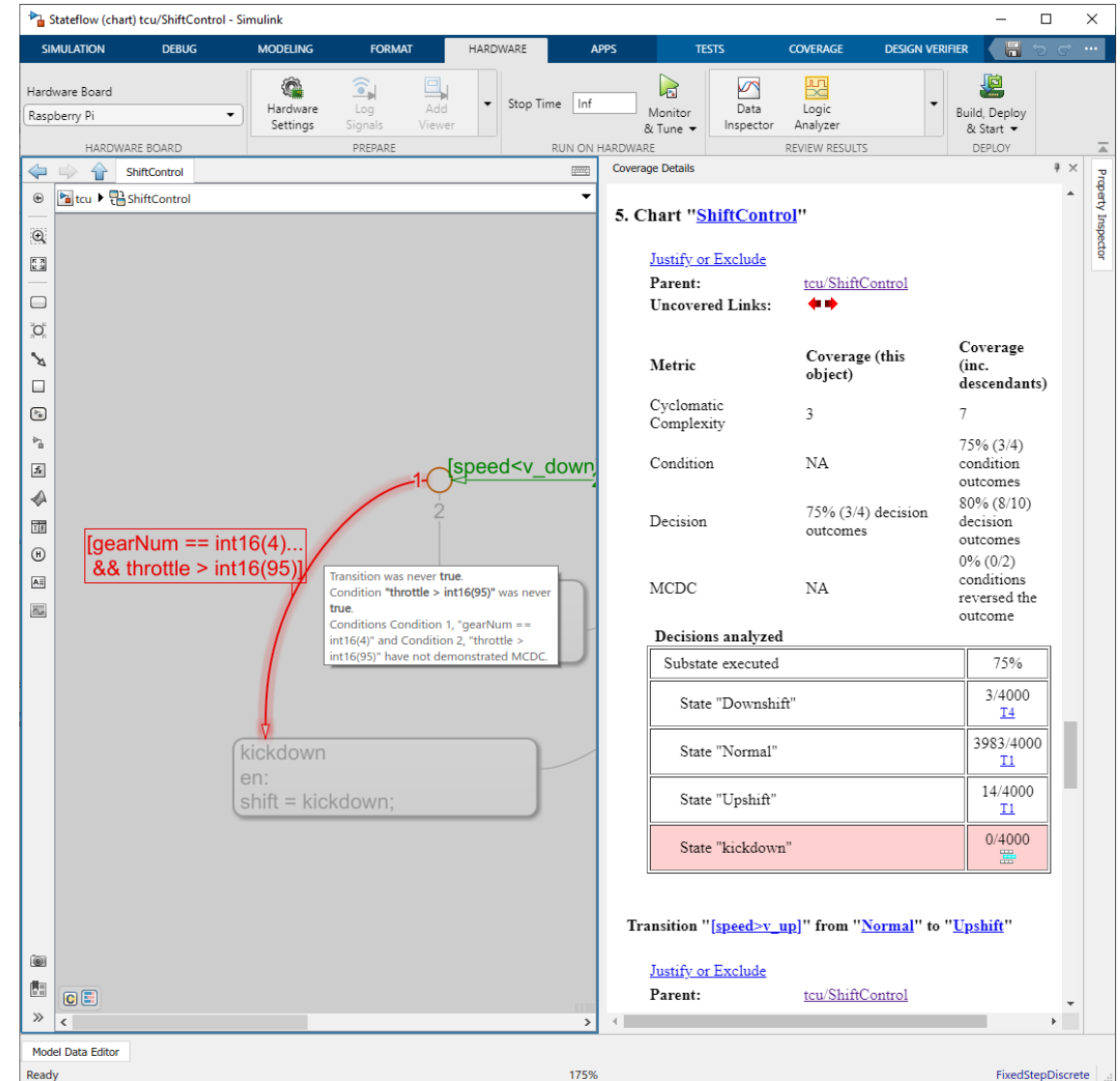
- Use Simulink Coverage to collect model coverage information
- Analyze how much of model algorithmic structure is executed during simulation and code execution
- Satisfy objectives from *DO-331 table MB.3/MB.4-A*

DO Table	Description
A-4	Verification of Outputs of Software Design Process
A-7	Verification of Verification Process Results



Coverage analysis

- Measure test suite completeness
- Dynamic simulation based analysis
- Identify testing gaps
- Find missing requirements
- Detect unintended functionality and unnecessary elements
- Supports many coverage criteria
 - Condition, Decision, MCDC, LUT, Signal Range



The screenshot shows the Stateflow Coverage Analysis tool interface. The main window displays a state transition diagram for the 'ShiftControl' chart. A transition labeled 'speed < v_down' is highlighted with a red circle and a red arrow pointing to a tooltip that reads: 'Transition was never true. Condition "throttle > int16(95)" was never true. Conditions Condition 1, "gearNum == int16(4)..." && throttle > int16(95)'. Below the transition, a state box is labeled 'kickdown en: shift = kickdown;'. The right-hand panel, titled 'Coverage Details', shows the following information:

5. Chart "ShiftControl"

[Justify or Exclude](#)

Parent: [tcu/ShiftControl](#)

Uncovered Links: ↔

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	3	7
Condition	NA	75% (3/4) condition outcomes
Decision	75% (3/4) decision outcomes	80% (8/10) decision outcomes
MCDC	NA	0% (0/2) conditions reversed the outcome

Decisions analyzed

Substate executed	75%
State "Downshift"	3/4000 I
State "Normal"	3983/4000 I
State "Upshift"	14/4000 I
State "kickdown"	0/4000 I

Transition "[speed > v_up]" from "Normal" to "Upshift"

[Justify or Exclude](#)

Parent: [tcu/ShiftControl](#)

Coverage Analysis in Simulink

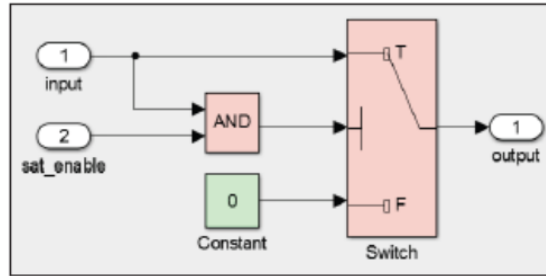
```

47 */
48 rtb_inputGE1ower = (rtb_input >= slvndemo_counter_U.lower);
49
50 /* Switch: '<Root>/Switch' incorporates:
51 * Inport: '<Root>/upper'
52 * Logic: '<Root>/And'
53 * RelationalOperator: '<Root>/upper GE input'
54 */
55 if (!(slvndemo_counter_U.upper >= rtb_input) && rtb_inputGE1ower) {

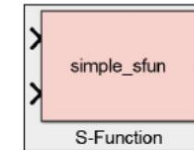
```

Decisions analyzed:	
!((slvndemo_counter_U.upper >= rtb_input) && rtb_inputGE1ower)	50%
false	51/51
true	0/51

Generated Code



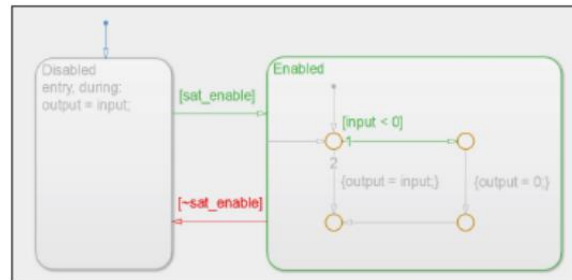
Simulink Models



Decisions analyzed:	
!(u[0]<0) && u[1]	50%
false	101/101
true	0/101

Conditions analyzed:		
Description:	True	False
u[0]<0	50	51
u[1]	0	50

C/C++ code S-Functions



Stateflow charts

```

1 function output = myFcn(input,sat_enable)
2 %#codegen
3
4 if (input<0 && sat_enable)
5     output = 0;
6 else
7     output = input;
8 end

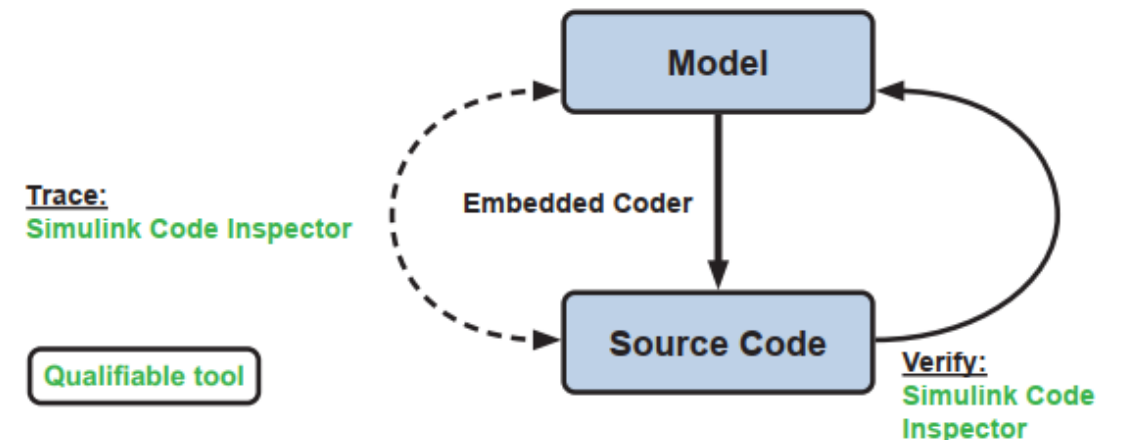
```

MATLAB function blocks

Model-Code Traceability

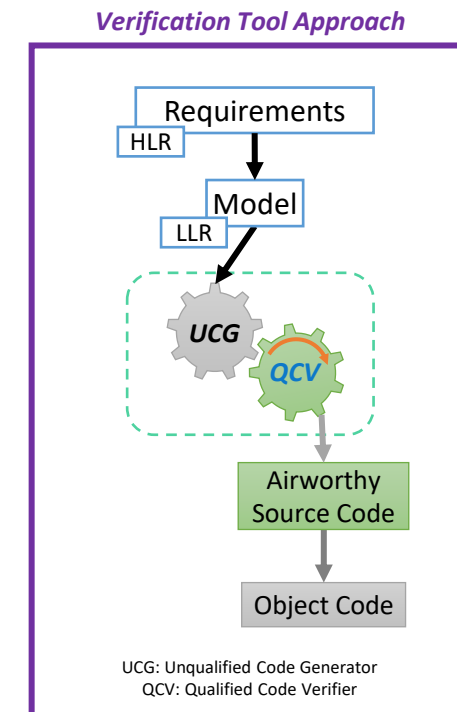
- Perform code verification and review with Simulink Code Inspector
- Compare model to it's generated code and determine structural equivalence
- Prove complete traceability between model components and code components

DO Table	Description
A-5	Verification of Outputs of Software Coding & Integration Processes



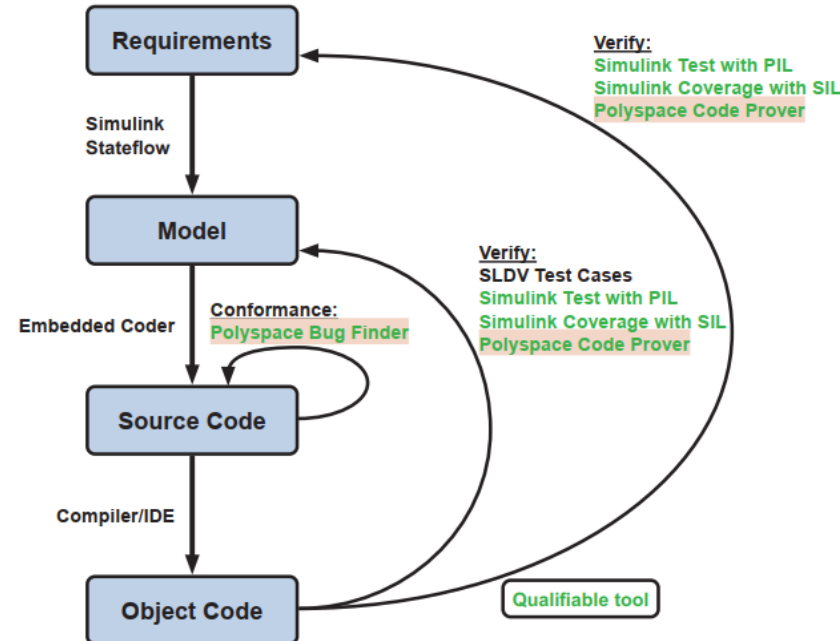
Model-Code Traceability

- Automate compliance with objectives from DO-331 table MB.A-5
 - „Source code complies with low-level requirements
 - Source code complies with software architecture
 - Source code is verifiable
 - Source code is traceable to low-level requirements”



Coding Standards Compliance

- Use Embedded Coder to generate MISRA-compliant C Code
- Perform Static C/C++ code analysis with Polyspace Bug Finder
- Demonstrate compliance with project's coding standards to comply with objectives from DO-178C table A-5
*„Source code conforms to standards
 Source code is accurate and consistent”*



DO Table	Description
A-5	Verification of Outputs of Software Coding & Integration Processes

Formal Code Verification

- Analyze source code to detect run-time errors
 - Arithmetic overflows, illegal memory access, division by zero and so on
- Prove absence of these run-time errors
- Demonstrate general robustness of code to comply with objectives from DO-178C table A-6

„Executable Object Code is robust with high-level requirements
 Executable Object Code is robust with low-level requirements”
- Using and qualifying Polyspace Code Prover satisfies objectives from DO-333 tables FM.A-5 and FM.A-7

DO Table	Description
A-5	Verification of Outputs of Software Coding & Integration Processes
A-6	Testing of Outputs of Integration Process
A-7	Verification of Verification Process Results

Formal Verification of Code



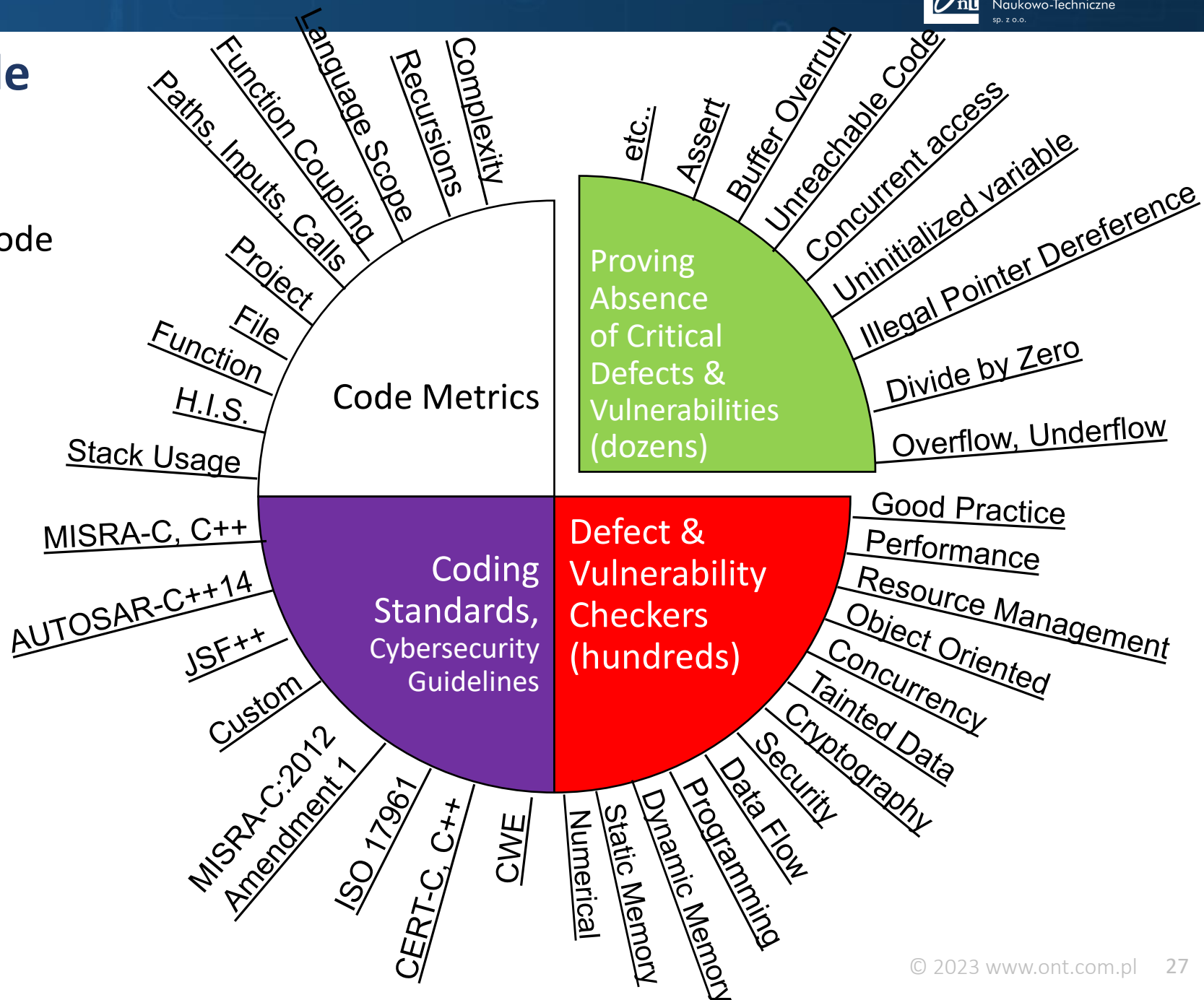
Polyspace Bug Finder

- High Quality, Secure, Compliant Code
 - Measurable, Maintainable, Consistent
 - Minimal number of defects or vulnerabilities
 - Credits for Functional safety and cybersecurity standards



Polyspace Code Prover

- Fully Trusted Components
 - Reliable, Robust, Safe, Secure
 - Proven free of critical runtime defects and vulnerabilities
 - Additional credits for standards

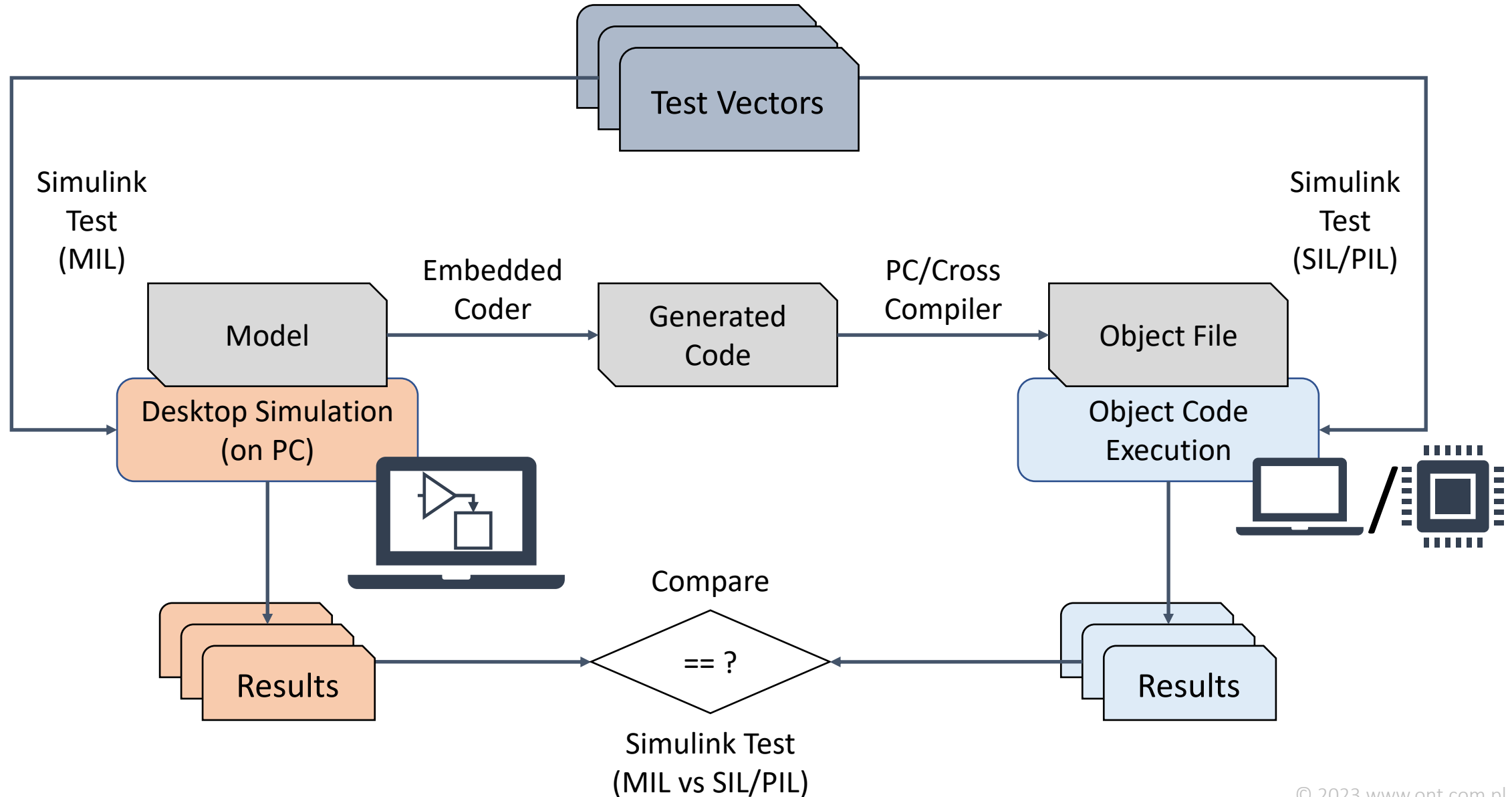


Object Code Verification and Coverage

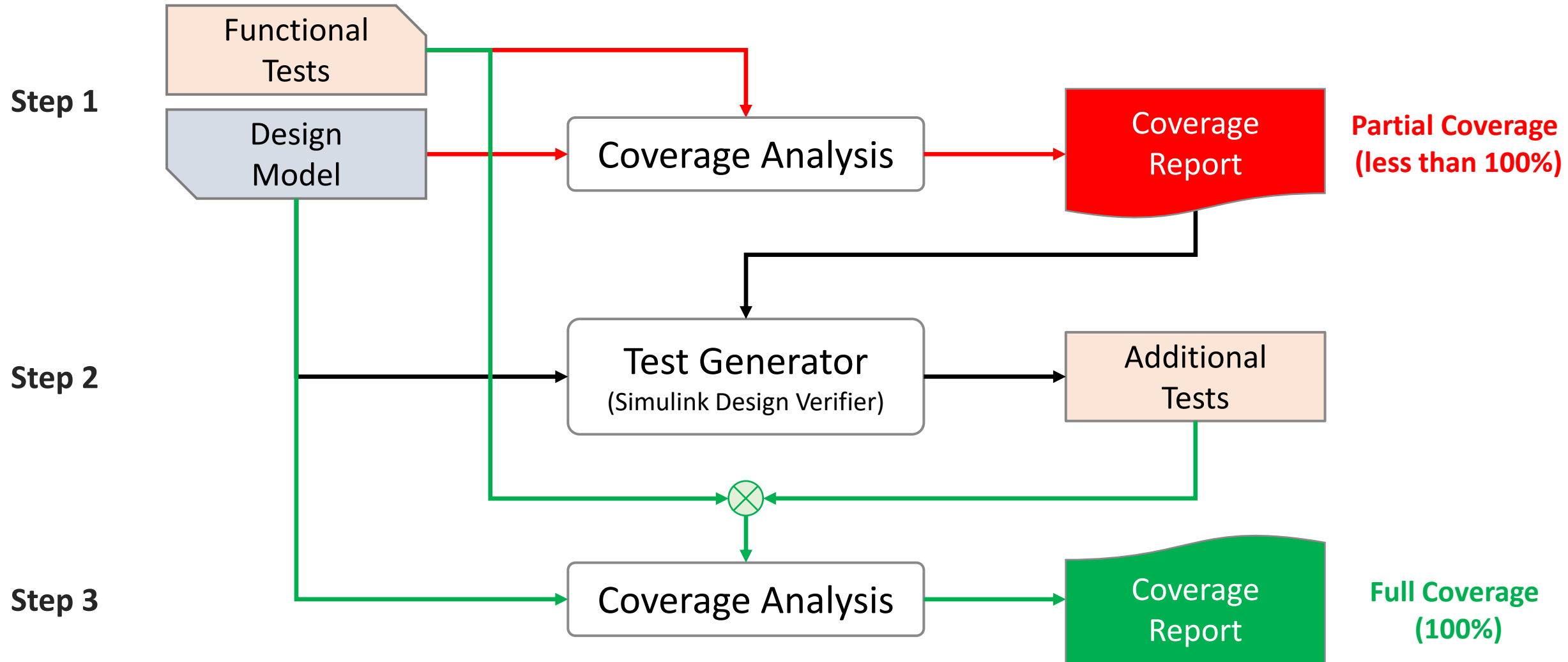
- Perform testing on compiled object code
- Run SIL and PIL simulations to demonstrate equivalence between model simulation and code execution
- Run Test Cases on object code in PIL to comply with objectives from DO-178C tables A-6 and A-7
 - „Executable Object Code complies with high-level requirements
 - Test coverage of low-level requirements is achieved”
- With Simulink Coverage collect coverage metrics from live code

DO Table	Description
A-6	Testing of Outputs of Integration Process
A-7	Verification of Verification Process Results

Automated Back to Back testing

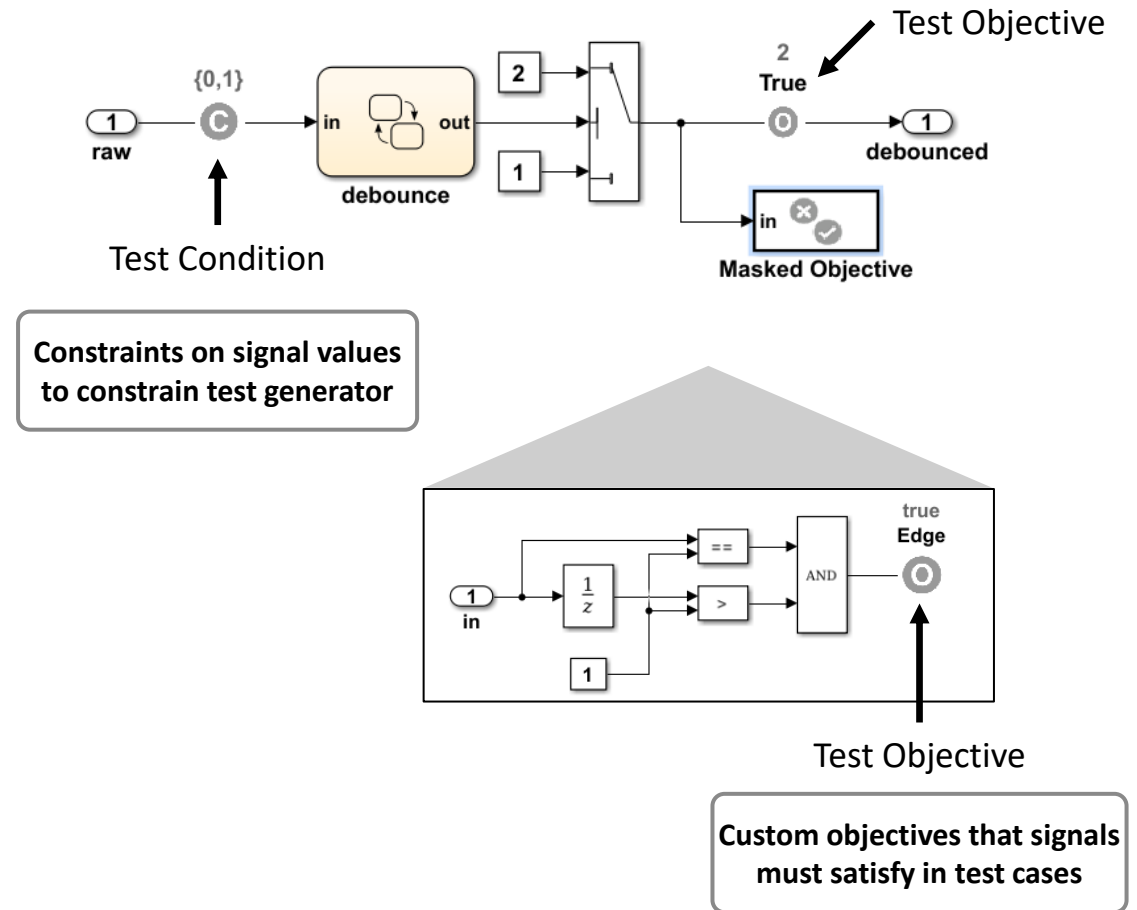


Low-level test generation



Addressing Missing Coverage

- Automated test case generation based on model structure
- Supports:
 - Simulink
 - Stateflow
 - MATLAB Function
 - C/C++ Code (S-Function, C Caller/Function)
- Variety of criteria:
 - Decision
 - Condition
 - MCDC
- Custom constrains for functional testing



Documentation and analysis

Chapter 2. Polyspace Run-Time Checks Statistics

Run-Time Checks Summary for PolyspaceProject - Flight_Control

Globally Proven: 99.6%

File	Proven	Green	Red	Gray	Orange
AHRS_Voter.c	100.0%	418	0	0	0
Flight_Control.c	100.0%	82	0	0	0
InnerLoop_Control.c	100.0%	79	0	0	0
OuterLoop_Control.c	100.0%	75	0	0	0
Actuator_Control.c	95.0%	57	0	0	3
Total	99.6%	711	0	0	3

Percentage of code checked for run-time errors

Result Set	% Checked
PolyspaceProject - Flight_Control	100%

Web Browser - Simulink Design Verifier Report

Simulink Design Verifier Report

Location: file:///D:/Presentations/Model_Based_Design/MBD_TCU_2019...

Analysis Constraints

Name	Analysis Constraint
Assumption	[0, 51]
Assumption1	[0, 51]

Design Min Max Constraints

Name	Design Min	Design Max	Constraint
throttle		[0, 100]	
speed		[0, 255]	

Rozdział 3. Proof Objectives Status

Spis treści

[Objectives Falsified with Counterexamples](#)

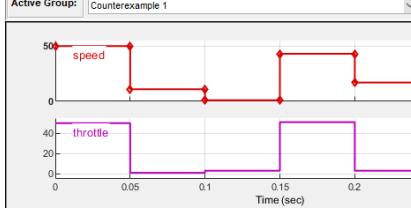
Objectives Falsified with Counterexamples

#	Type	Model Item	Description	Analysis Time (sec)
1	Proof Objective	Verification Subsystem ProofObjective	Objective: F	12

Rozdział 4. Properties

File Edit Group Signal Axes Help

Active Group: Counterexample 1



Name: speed T: [] Right Point: []

Index: 1 Y: [] Y: []

Click to select, Shift+click to add

Traceability Matrix

HOME

Filter Panel

Missing Expected Links

Missing Links

Models

Leaf Block

Stateflow Object

Subsystem

Simulink Tests

Test Case

Test File

Test Suite

Change Tracking

With Change Issues

Link

Type

Container

Functional

Cell

Implements

Verifies

Change Tracking

With Change Issues

Code Generation Report

Find: [] Match Case

Contents

Summary

[Subsystem Report](#)

[Traceability Report](#)

[Static Code Metrics Report](#)

[Code Replacements Report](#)

[Coder Assumptions](#)

Generated Code

[-] Model files

[AHRS_voter.c](#)

[AHRS_voter.h](#)

[+] Shared files (1)

[+] Other files (1)

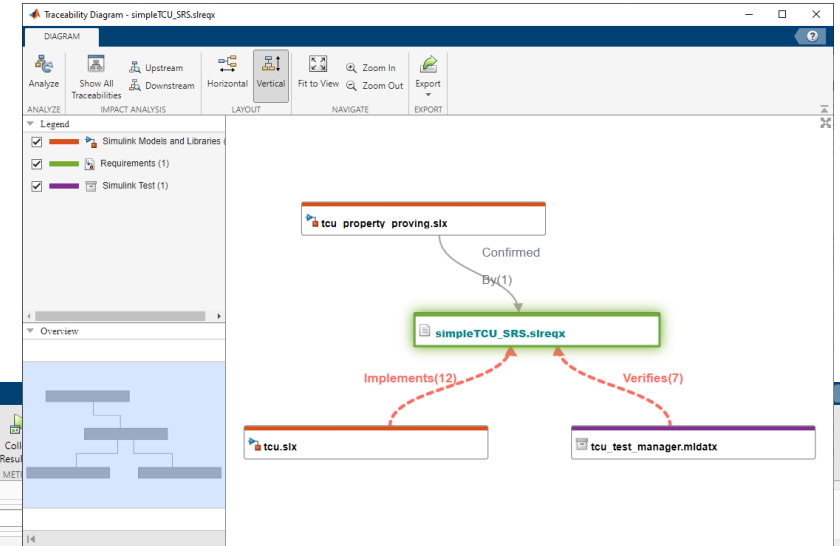
Traceable Simulink Blocks / Stateflow Objects / MAT

Root system: [AHRS_voter](#)

Object Name	Code Location
<Root>/Avg_Value	AHRS_voter.c:391, 394, 48
<Root>/Default	AHRS_voter.c:575, 582
<Root>/Merge	AHRS_voter.c:40, 50, 52, 5
	AHRS_voter.h:82, 83
<Root>/Mid_Value	AHRS_voter.c:96, 99, 387
<Root>/Single_Value	AHRS_voter.c:485, 488, 57
<Root>/Sum	AHRS_voter.c:84, 89, 92, 9
<Root>/Switch Case	AHRS_voter.c:82, 86, 92, 9

Subsystem: [AHRS_voter/Avg_Value](#)

Object Name	Code Location
<S1>/Action Port	AHRS_voter.c:392
<S1>/Constant	AHRS_voter.c:20, 398, 403, 414, 421, 42
	AHRS_voter.h:70, 71
<S1>/Gain	AHRS_voter.c:25, 466, 469, 475, 476, 48
	AHRS_voter.h:73, 74
<S1>/Sum	AHRS_voter.c:467, 472, 476, 477
<S1>/Switch	AHRS_voter.c:397, 400, 406, 407, 408, 409, 410, 411, 413, 414, 418



TEST CASE ANALYSIS

Requirements Linked to Tests: 42.9% (12 Unlinked)

Tests Linked to Requirements: 87.5% (1 Unlinked)

Test Case Breakdown

Type	Test Cases
Simulation	8
Equivalence	0
Baseline	0

Tests with Tag

Tag	Test Cases
1_Draft	1
2_Under_review	2
3_Reviewed	2
4_Released	3

TEST RESULT ANALYSIS

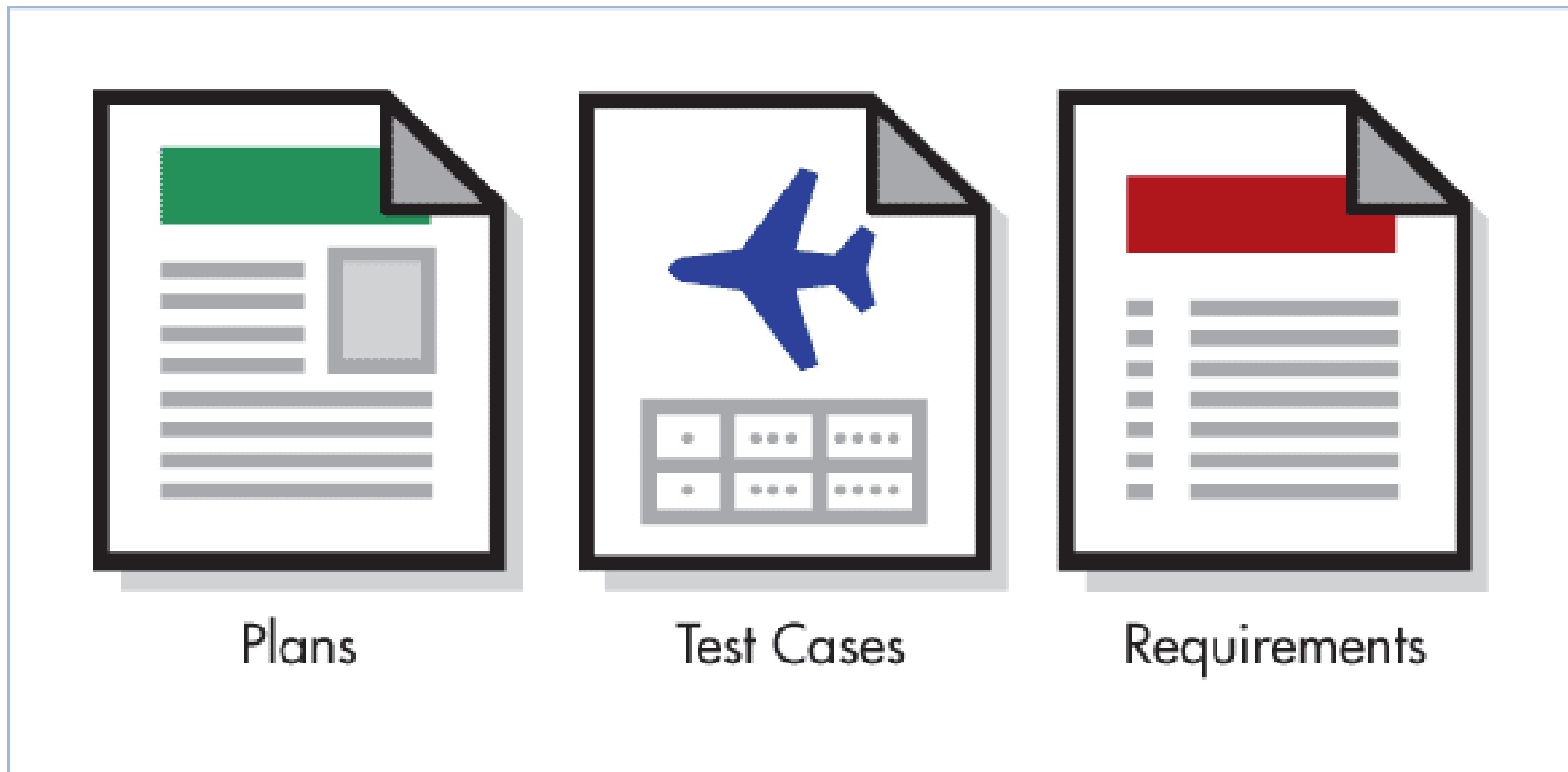
Model Test Status: 75% Passed (1 Failed, 0 Untested, 1 Disabled)

Model Coverage

Category	Achieved	Justified
Execution	100%	0%
Condition	100%	0%
Decision	100%	0%
MC/DC	0%	0%

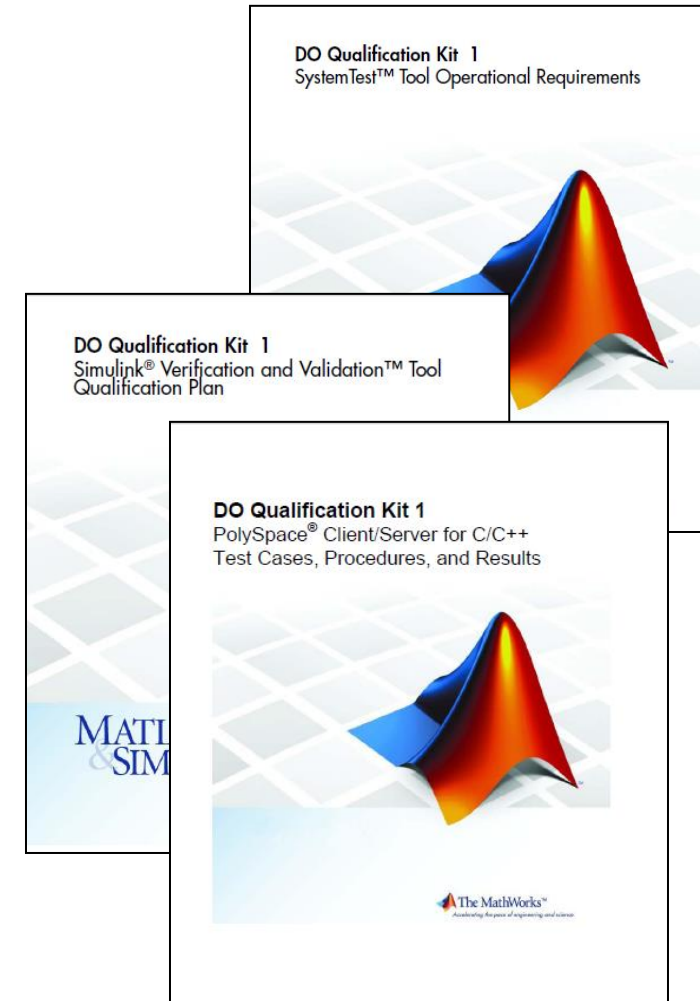
DO Qualification Kit

DO-330 Compliant Artifacts

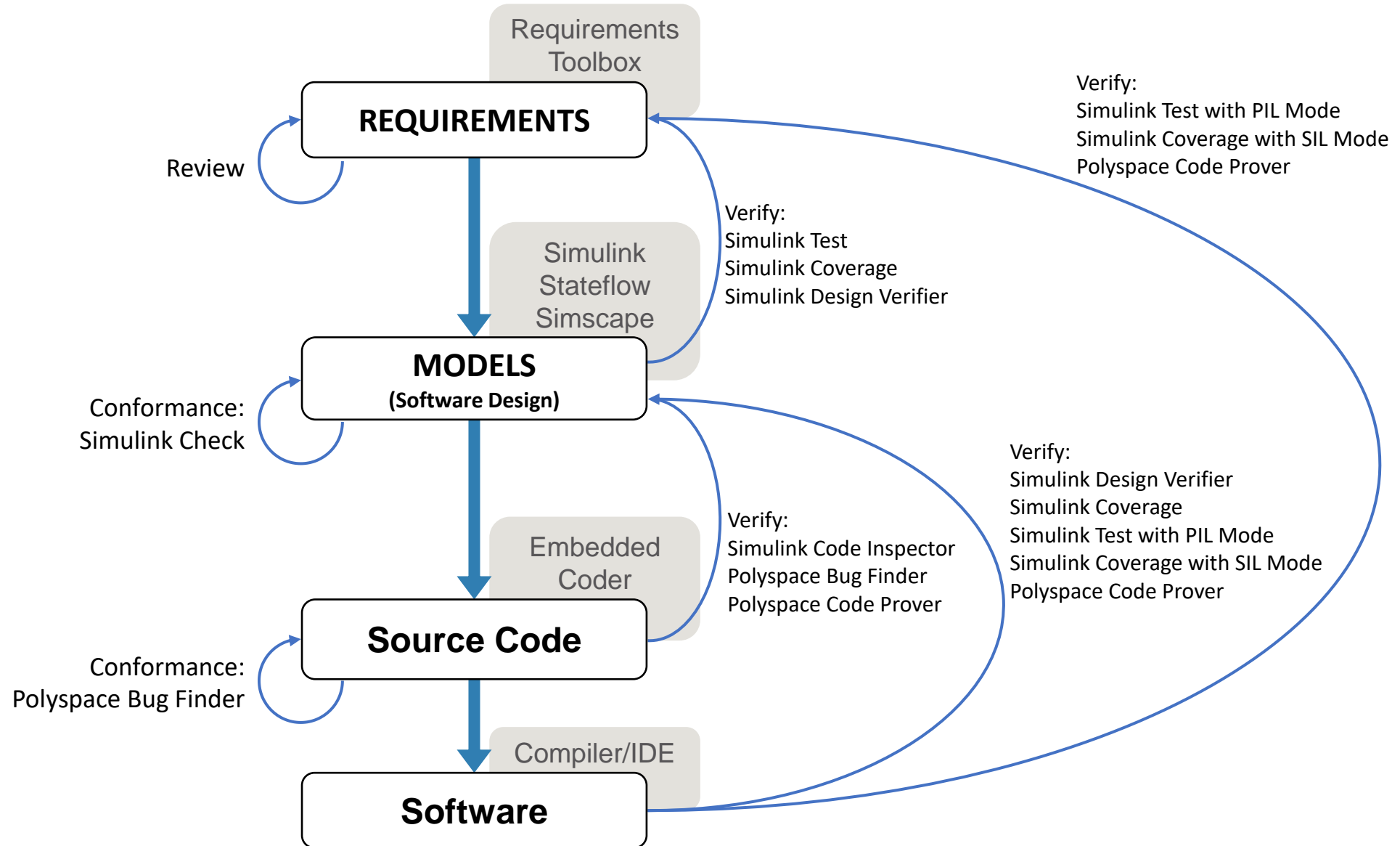


Qualifiable Products

- Requirements Toolbox (TQL-5)
- Simulink Report Generator (TQL-5)
- Simulink Check (TQL-5)
- Simulink Design Verifier (TQL-5)
- Simulink Test (TQL-5)
- Simulink Model Compare (TQL-5)
- Simulink Coverage (TQL-5)
- Simulink Code Inspector (TQL-5)
- Polyspace Bug Finder (TQL-5)
- Polyspace Code Prover (TQL-4)



DO-178C/DO-331 with MathWorks tools





Oprogramowanie
Naukowo-Techniczne
sp. z o.o.

www.ont.com.pl



matlab.pl



oprogramowanie-
naukowo-techniczne



ONT MATLAB



Konrad Kolski

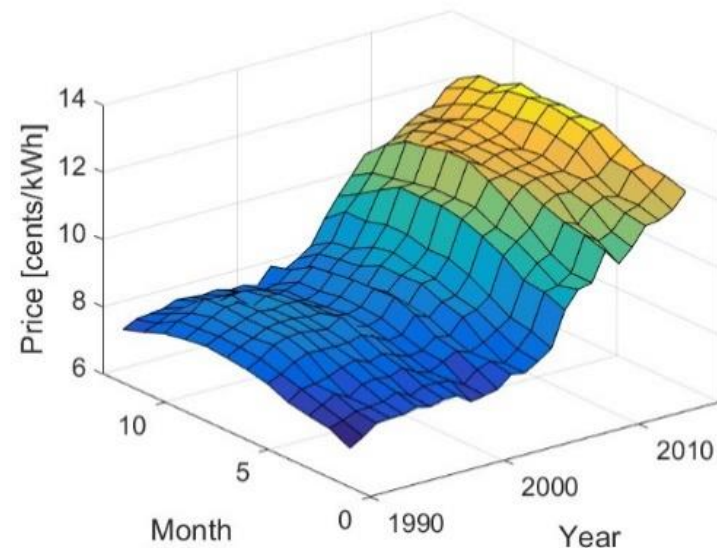
Application Engineer, ONT

konrad.kolski@ont.com.pl

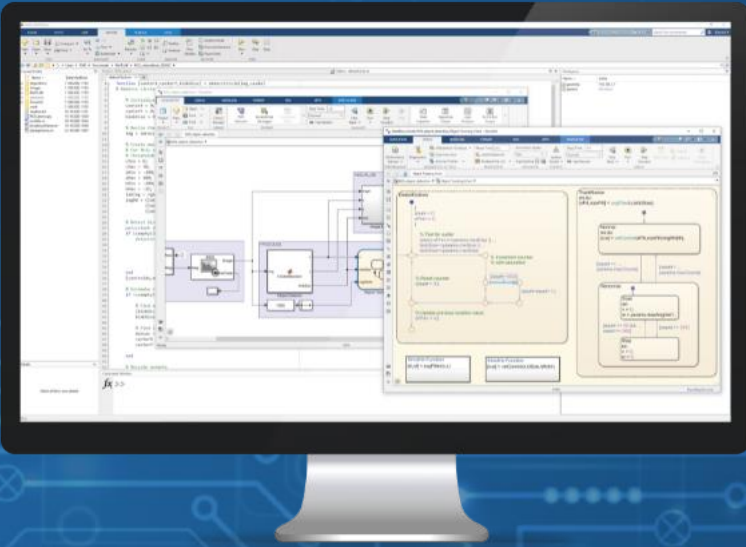
tel. +48 252 252 252

Trainings at ONT

- Over 50 courses from 13 areas of knowledge
- Courses for beginners and advanced users
- Training through practice – exercises for every course
- Courses organised by official MathWorks reseller and lead by specialists
- Small groups – up to 12 people
- [Trainings schedule](#)



Contact:
szkolenia@ont.com.pl
+48 12 630 49 55



APPLICATIONS

- ▶ Robotics and Automation
- ▶ Computational Finance
- ▶ Autonomous Vehicles
- ▶ Electronics
- ▶ Artificial Intelligence
- ▶ Biomedical Engineering
- ▶ Systems Engineering and certification
- ▶ Power Electronics and Systems
- ▶ Communications and Radar Systems

Let's stay in touch

Oprogramowanie Naukowo-Techniczne sp. z o.o.
MATLAB and Simulink authorised reseller for Poland
ul. Pod Fortem 19, 31-302 Kraków, Poland | www.ont.com.pl