

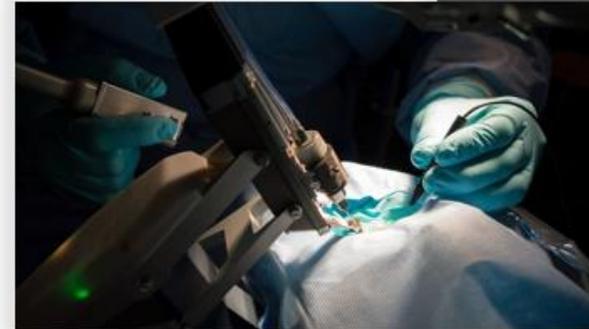
# MATLAB EXPO

인증 표준을 만족하기 위한 요구사항 기반 검증의 자동화 방안  
*류성연, MathWorks Korea*



# Challenge to Deliver Complex Systems and Meet Standards

- Need to meet industry or customer's standards
  - DO-178C (Aero), ISO 26262 (Auto), IEC 62304 (Medical), IEC 61508 (Industrial), MISRA, etc.
- Time and cost for safety critical projects estimated 20-30 times more costly\*
- Finding defects late increases cost and time



\*Source: [Certification Requirements for Safety-Critical Software](#)

# ISO 26262-6:2018 notes Simulink and Stateflow

**Table 5 — Notations for software unit design**

Notations		ASIL			
		A	B	C	D
1a	Natural language <sup>a</sup>	++	++	++	++
1b	Informal notations	++	++	+	+
1c	Semi-formal notations <sup>b</sup>	+	+	++	++
1d	Formal notations	+	+	+	+

<sup>a</sup> Natural language can complement the use of notations for example where some topics are more readily expressed in natural language or provide an explanation and rationale for decisions captured in the notations.  
EXAMPLE To avoid possible ambiguity of natural language when designing complex elements, a combination of an activity diagram with natural language can be used.

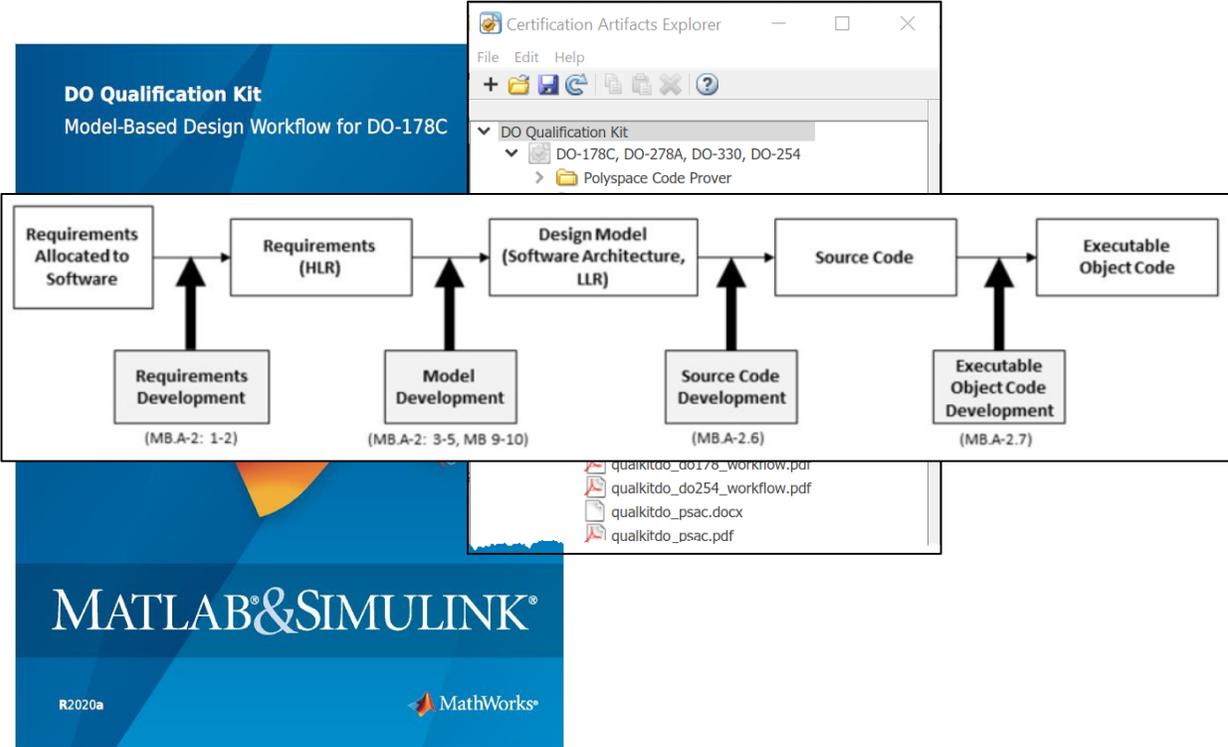
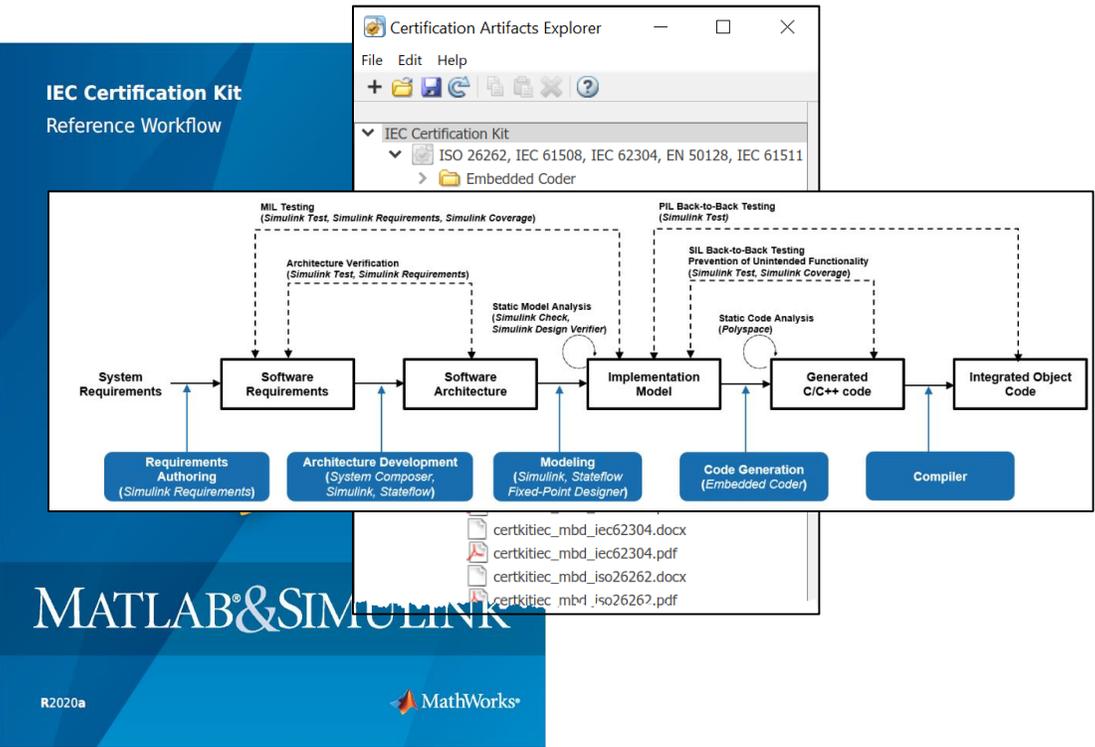
<sup>b</sup> Semi-formal notations can include pseudocode or modelling with UML®, SysML®, Simulink® or Stateflow®.

NOTE UML®, SysML®, Simulink® and Stateflow® are examples of suitable products available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of these products.

NOTE In the case of model-based development with automatic code generation, the methods for representing the software unit design are applied to the model which serves as the basis for the code generation.

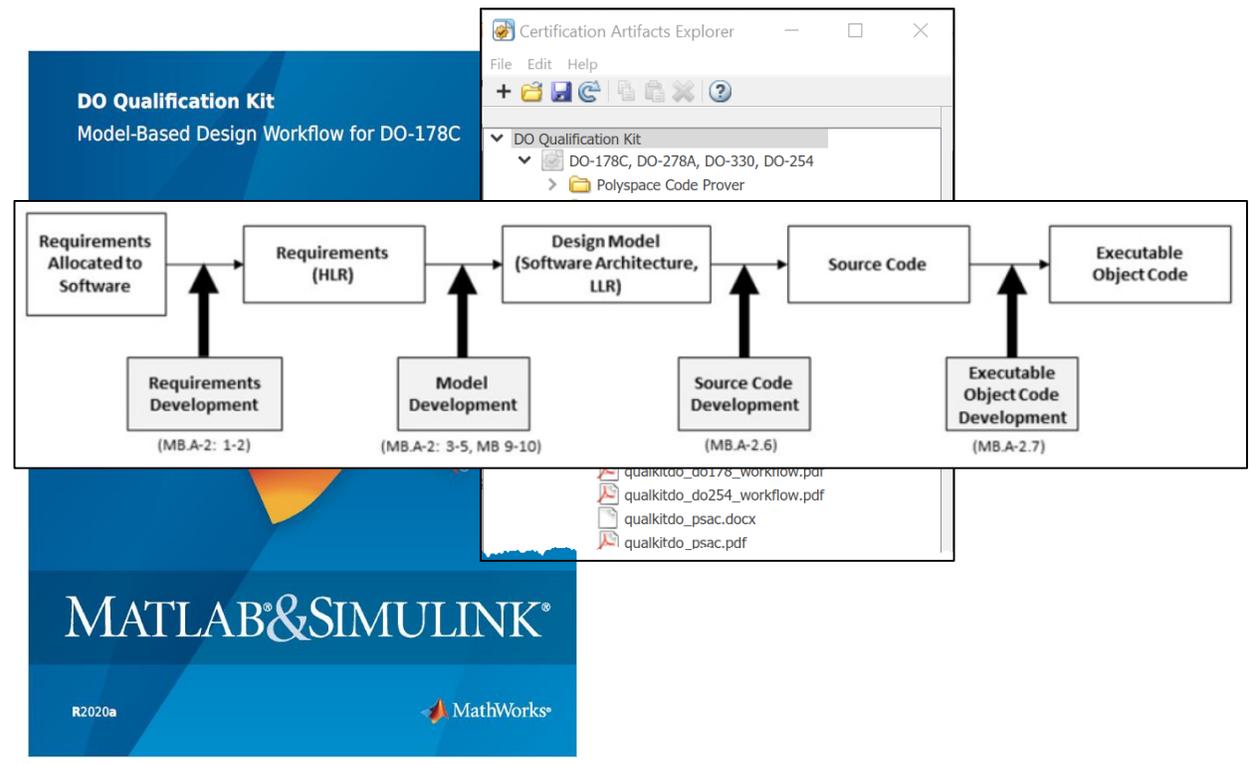
# Qualify tools with IEC Certification Kit and DO Qualification Kit

- Qualify code generation and verification products
- Includes documentation, test cases and procedures



# Qualify tools with IEC Certification Kit and DO Qualification Kit

- Qualify code generation and verification products
- Includes documentation, test cases and procedures



# Qualify tools with IEC Certification Kit and DO Qualification Kit

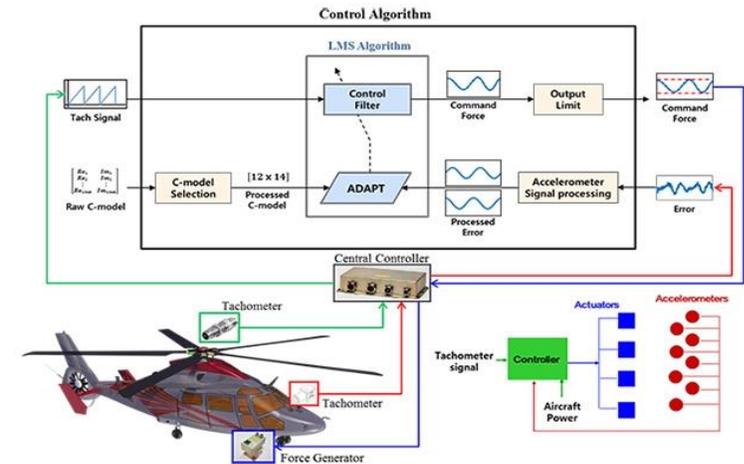
- Qualify code generation and verification products
- Includes documentation, test cases and procedures

KOSTAL Asia R&D Center Receives ISO 26262 ASIL D Certification for Automotive Software Developed with Model-Based Design



Kostal's electronic steering column lock module

Korea Aerospace Industries Develops Helicopter Active Vibration Control System Software to DO-178C Standards



The KAI active vibration control system

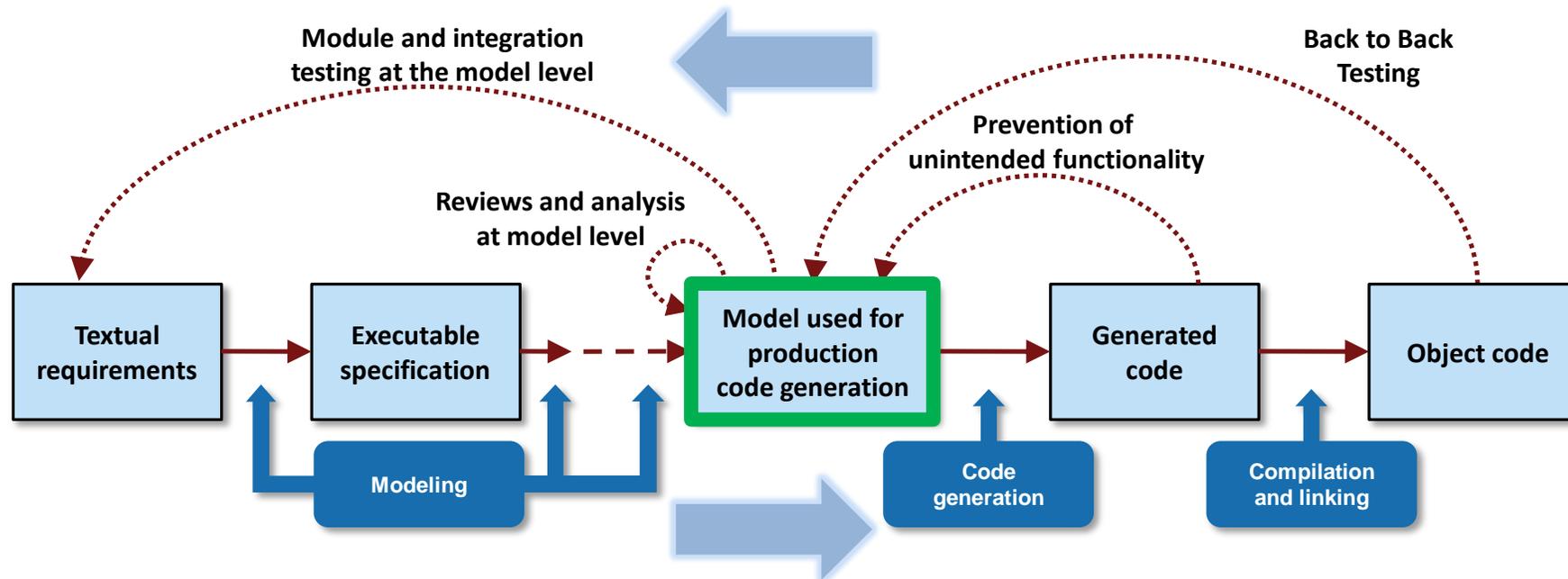
# Conform to Certification Standards with Reference Workflow

## Model Verification

*Discover design errors at design time*

## Code Verification

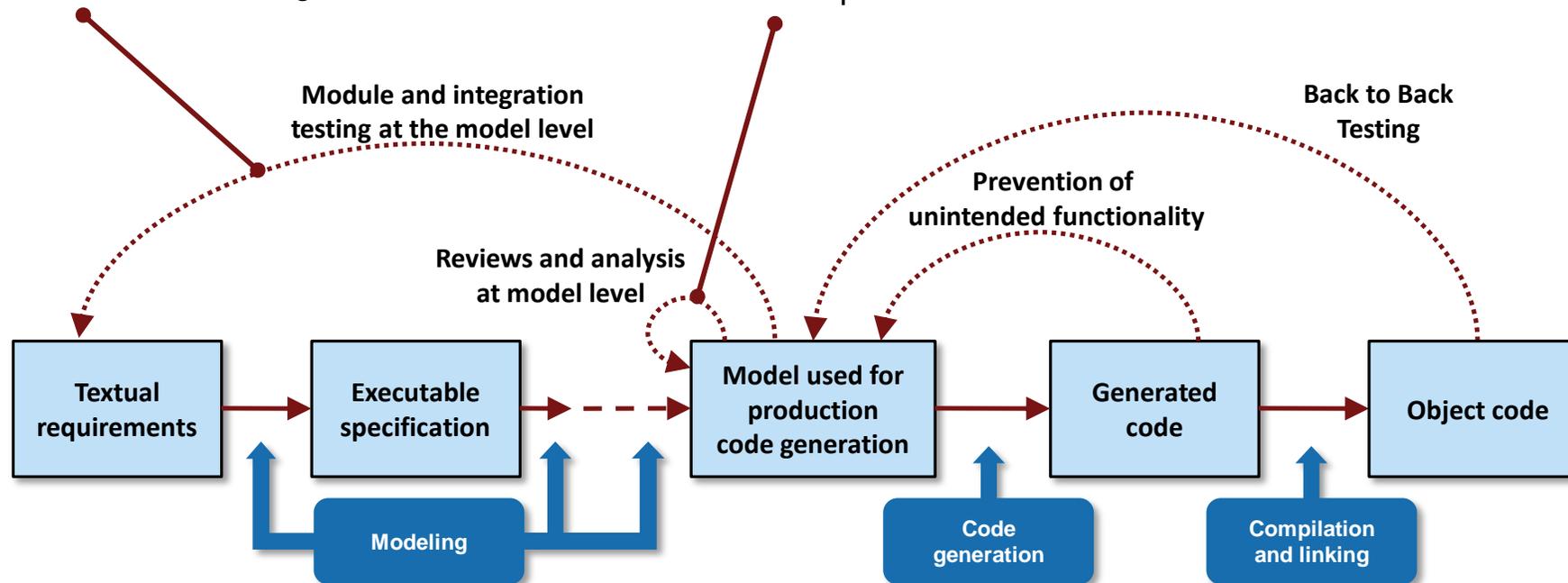
*Gain confidence in the generated code*



# Model Verification: Discover Design Errors at Design Time

## Model Verification

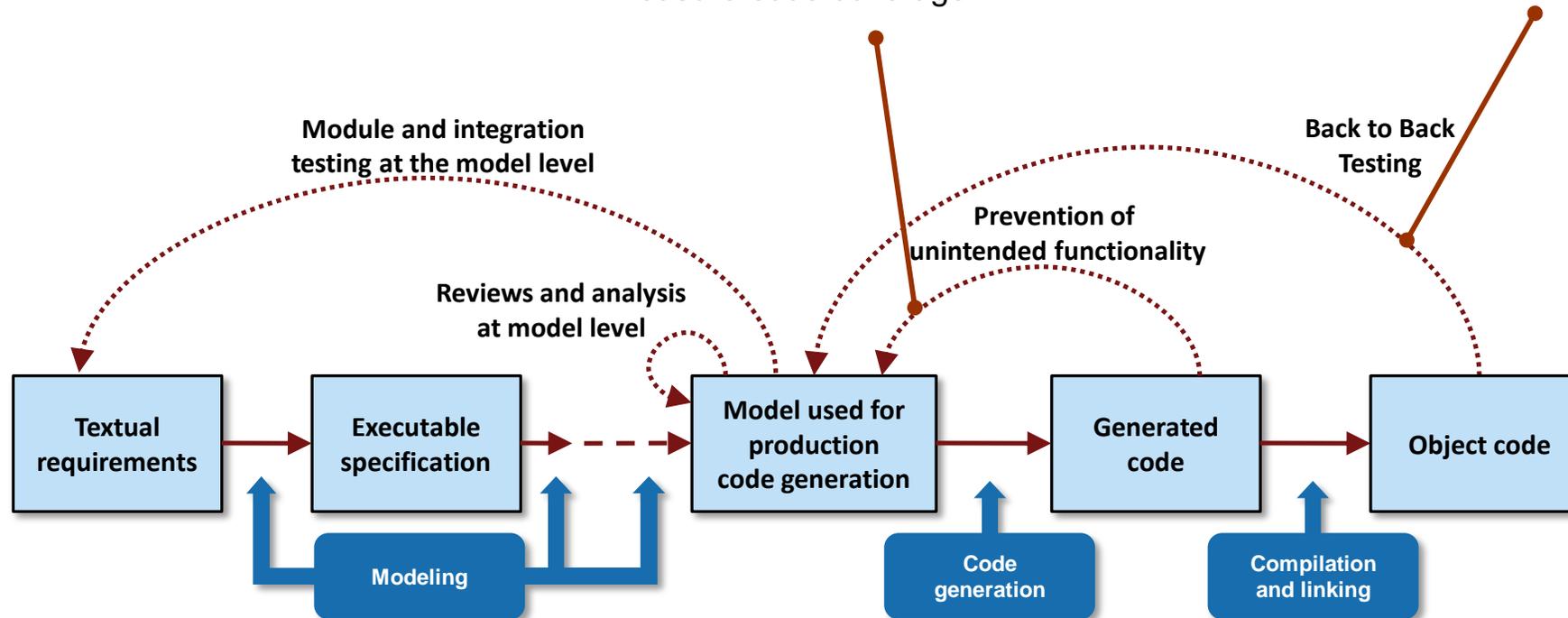
- Manage requirements
- Systematically test
- Measure model coverage
- Check standard compliance
- Detect design errors
- Prove model behavior compliance



# Code Verification: Gain Confidence in the Generated Code

## Code Verification

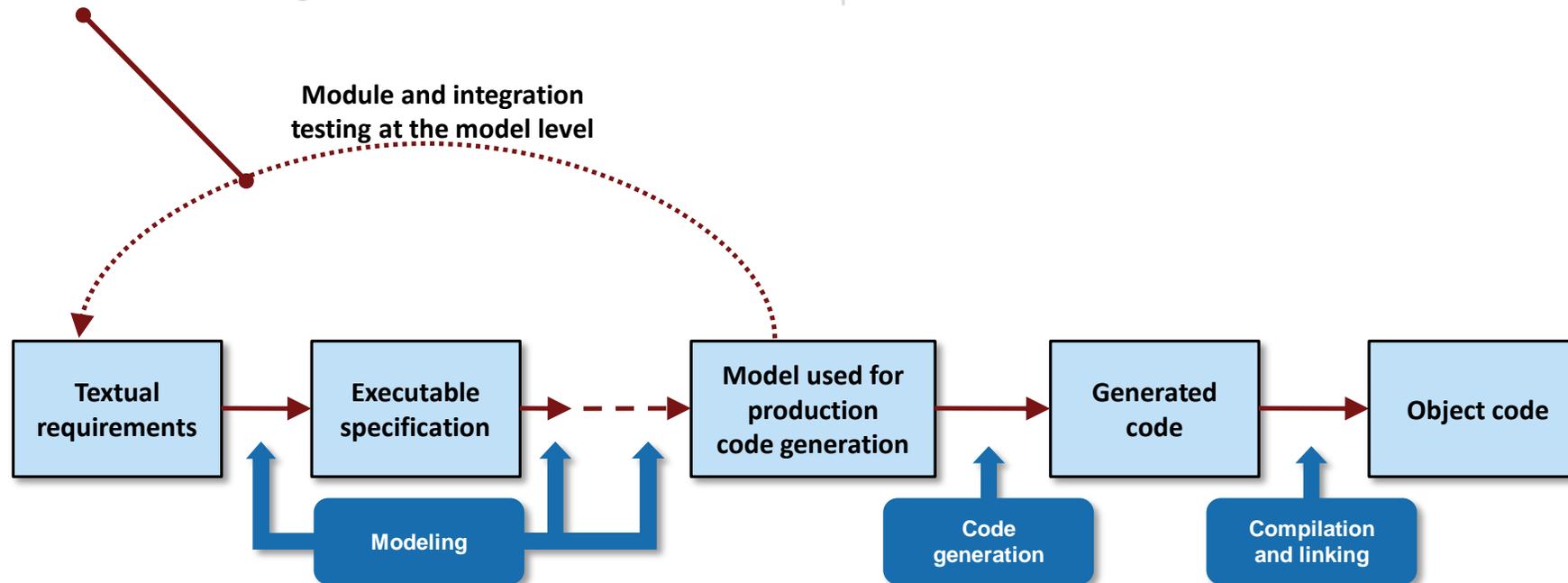
- Trace code to model and requirements
- Measure code coverage
- SIL/PIL equivalence testing
- Generate 100% coverage test vectors



# Manage Requirements

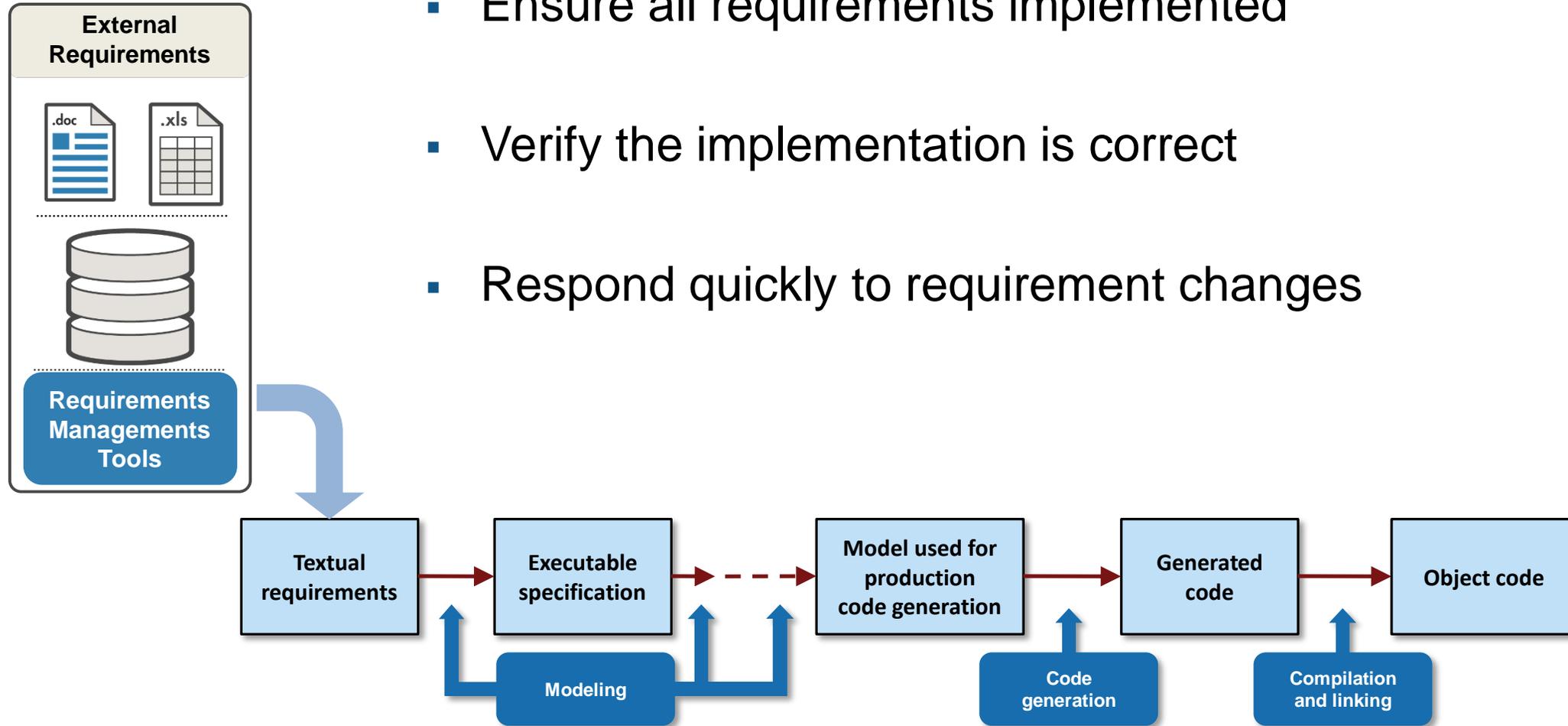
## Model Verification

- Manage requirements
- Systematically test
- Measure model coverage
- Check standard compliance
- Detect design errors
- Prove model behavior compliance

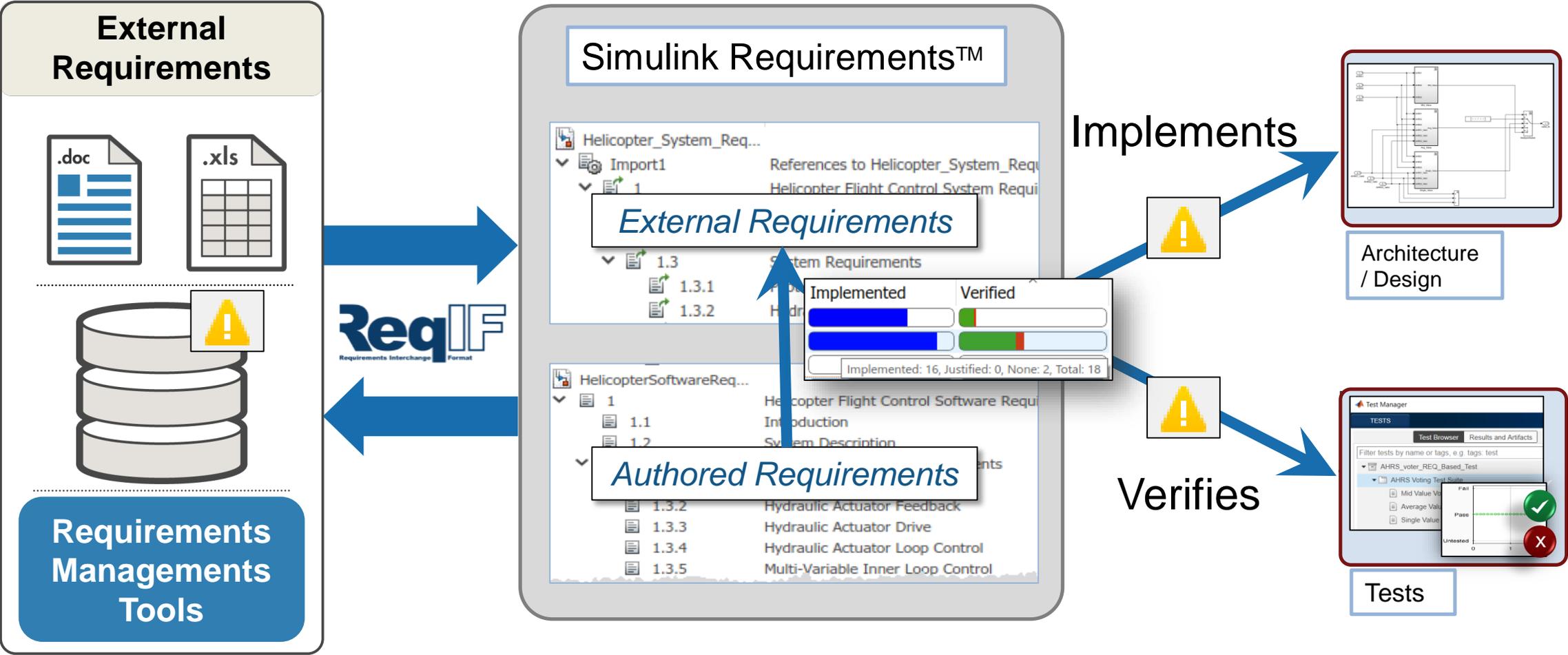


# Manage Requirements

- Ensure all requirements implemented
- Verify the implementation is correct
- Respond quickly to requirement changes



# Work with Requirements, Architecture and Design Together



# Demo: Requirements Perspective

The screenshot displays the Requirements Perspective in MATLAB Simulink. The top portion shows a Simulink model for a cruise control system. The model includes inputs for 'enable', 'brake', 'speed', 'set', 'inc', and 'dec'. It features logic blocks such as 'AND', 'OR', 'NO', and a '1/z' block. Key functional blocks are labeled 'Active Control' and 'Compute the target speed'. A 'PI Controller' block receives an 'error' signal and outputs 'throt'. A requirement 'ENABLE: Engage cruise control' is shown to be implemented by the 'Active Control' block.

The bottom portion shows the Requirements table for 'cruiseControlRBTcovExample'. The table has columns for Index, Summary, Implemented, and Verified. The 'ENABLE: Engage cruise control' requirement (Index 3) is highlighted.

Index	Summary	Implemented	Verified
1	Set target speed	████████████████████	████████████████████
2	Brake disengages cruise control	████████████████████	████████████████████
3	Engage cruise control	████████████████████	████████████████████
4	Increment set speed	████████████████████	████████████████████
5	Decrement set speed	████████████████████	████████████████████
6	Throttle to maintain set speed	████████████████████	████████████████████
SafetyReq			
1	Disable Throttle when Braking		

Property Inspector details for Requirement: ENABLE:

- Type: Functional
- Index: 3
- Custom ID: ENABLE
- Summary: Engage cruise control
- Description: The ENABLE button shall engage...

# Test and Requirements Traceability

The screenshot shows a Simulink model of a cruise control system. The model includes inputs for enable, brake, speed, set, inc, and dec. It features logic blocks for AND, OR, NOT, and PI Controller. Key components include 'Active Control', 'Determine if the control is active', 'Compute the target speed', and 'PI Controller'. The output is 'throt'. Below the model is a 'Requirements - cruiseControlRBTcovExample' window with a table showing the verification status of various requirements.

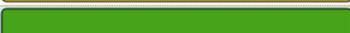
Index	Summary	Implemented	Verified
1	SET button locks set speed	Blue bar	Yellow bar
2	Applying BRAKE disengages cruise control	Blue bar	Yellow bar
3	ENABLE button engages cruise control	Blue bar	Yellow bar
4	INCREMENT button increases set speed	Blue bar	Green bar
5	DECREMENT button decreases set speed	Blue bar	Green bar
6	THROTTLE applied smoothly if speed differs from target	Blue bar	Red bar

The Test Manager interface shows a list of tests under the 'cruiseControlRBTcovTests' suite. The tests listed are: Set Speed Test, Brake Test, Enable Test, Increment Test, Decrement Test, and Throttle Test. A green arrow points from the 'Set Speed Test' entry in the Test Manager to the 'Set Speed Test' row in the Requirements table.

**Verification Status**

- Green square: Passed
- Red square: Failed
- Yellow square: Unexecuted
- White square: Missing

# Review and Analyze Traceability with Traceability Matrix

Summary	Implemented	Verified
		
SET button locks set speed		
Applying BRAKE disengages cruise control		
ENABLE button engages cruise control		
INCREMENT button increases set speed		
DECREMENT button decreases set speed		
THROTTLE applied smoothly if speed differs from target		

Requirement is missing  
link to Test Case

- Review links between different requirements, model, test
- Filter view to manage large sets of artifacts
- Highlight missing links
- Directly add links to address gaps

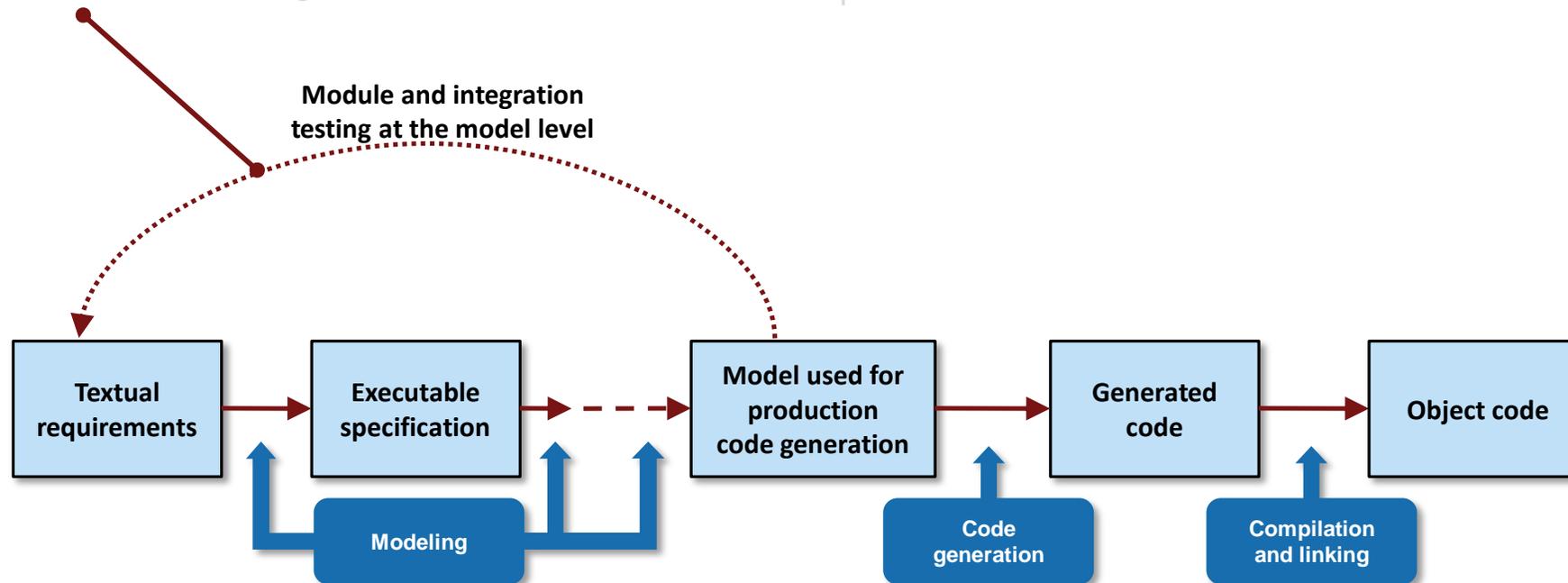
The screenshot shows the Traceability Matrix tool interface. The main window displays a matrix of links between requirements and tests. The requirements are listed on the left, and the tests are listed on the top. The matrix cells are color-coded: blue for implemented links, green for verified links, and yellow for missing links. A red circle highlights a missing link between the requirement 'Applying BRAKE disengages cruise control' and the test 'Brake Test'. A context menu is open over this cell, showing options for 'Left', 'Top', and 'Link', with 'Link' set to 'None (Create)'. A summary table is also visible, showing the status of each requirement.

Summary	Implemented	Verified
SET button locks set speed	Blue bar	Green bar with red segment
Applying BRAKE disengages cruise control	Blue bar	Green bar with yellow segment
ENABLE button engages cruise control	Blue bar	Green bar
INCREMENT button increases set speed	Blue bar	Green bar
DECREMENT button decreases set speed	Blue bar	Green bar
THROTTLE applied smoothly if speed differs from target	Blue bar	Red bar

# Systematic Functional Testing of Model

## Model Verification

- Manage requirements
- **Systematically test**
- Measure model coverage
- Check standard compliance
- Detect design errors
- Prove model behavior compliance



# Overview to Requirements-Based Test

**Goal:** Assess extent to which requirements-based tests exercise corresponding designs

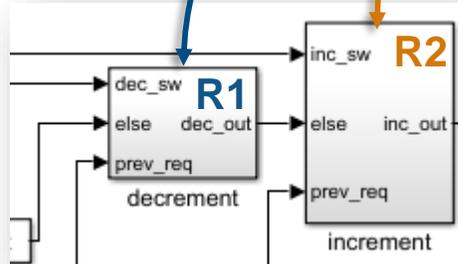
## Requirements

Index	ID	Summary
1	SET_SPEED	SET button locks set speed
2	BRAKE	BRAKE disengages cruise control
3	ACTIVATE	ACTIVATE cruise control when enabled, speed set, and no brake
4	INCREMENT	INCREMENT button increases set speed
5	DECREMENT	DECREMENT button decreases set speed
6	THROTTLE	THROTTLE applied smoothly if speed differs from target

Implemented by links

Verified by links

## Model



Test Browser Results and Artifacts

Filter tests by name or tags, e.g. tag

- cruiseControlRBTCovTests
  - Cruise Control Test Suite
    - Set Speed Test **R1**  
**R2**
    - Brake Test
    - Activate Test
    - Increment Test
    - Decrement Test
    - Throttle Test

Set Speed Test

cruiseControlRBTCovTests » Cruise Control Test Suite » Set Speed Test

Simulation Test

Select releases for simulation: Current

Create Test Case from External File

TAGS

DESCRIPTION\*

REQUIREMENTS\*

[SET\\_SPEED: SET button locks set speed \(cruiseControlRBTCovReqs#1\)](#)

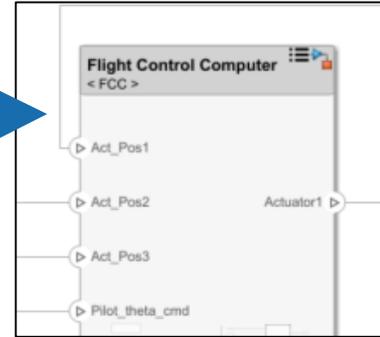
DO-178C  
6.4.4.2

... coverage information collected during requirements-based testing to confirm that ...

# Requirements Based Verification with Simulink Test

**FUNCTIONAL REQUIREMENTS**  
The flight control system shall ...

Implemented by

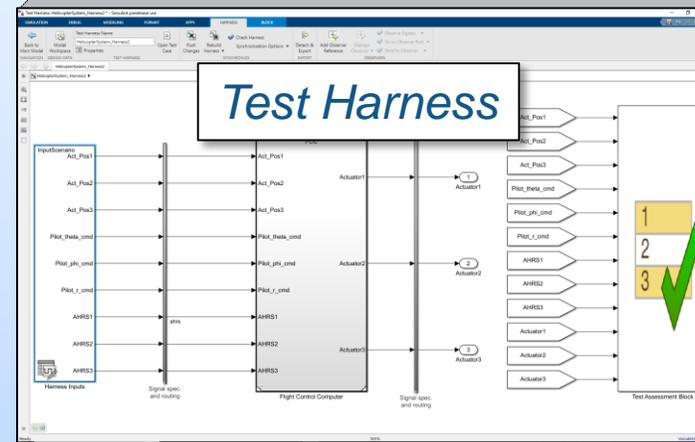
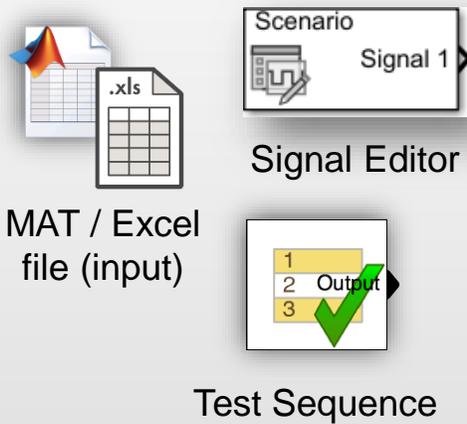


*System Composer /  
Simulink / Stateflow*

Verified by

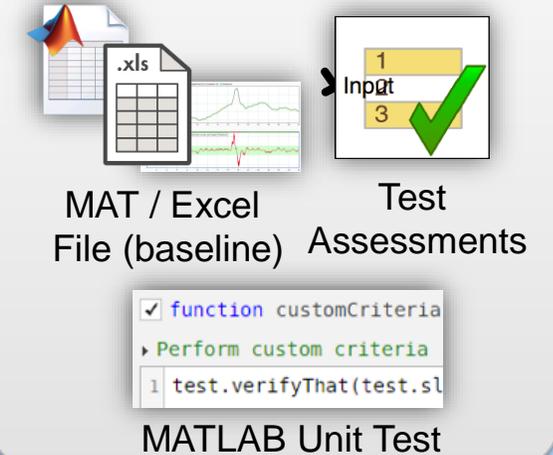
## Test Case

### Inputs



Simulink Test™

## Assessments



# Automate Functional Testing with Simulink Test Test Manager

- ✓ Create test cases
- ✓ Group into suites and test files
- ✓ Execute individual or batch
- ✓ View result summary
- ✓ Analyze results
- ✓ Archive, export, report

The screenshot displays the Simulink Test Manager interface with several key components highlighted in red:

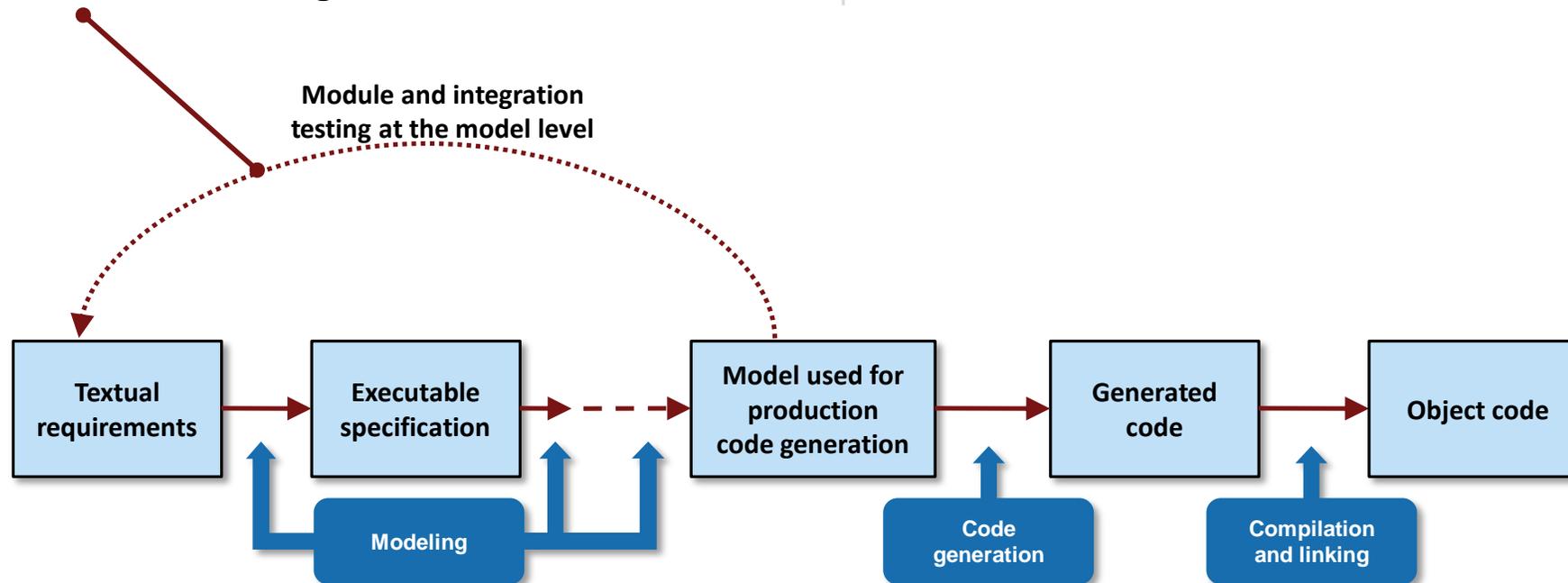
- Test Manager Main Window:** Shows a test suite named "Slow Accel" under "Signal Builder Baseline examples". The "Run" and "Report" buttons in the top toolbar are highlighted.
- Test Browser:** A tree view on the left shows the test hierarchy, including "ComponentTesting", "General Performance Test", "Functional and Regression tests", "Signal Builder Baseline examples", "Slow Accel", "Fast Accel", "Decel", "ExcelDrivenExamples", "Software-in-the-loop Testing", "SystemTesting", and "ExampleBaselineTesting".
- Test Results Summary:** A table shows the results of the test run:
 

NAME	STATUS
Results : 2015-Jan-12 17:35:31	2 <span style="color: green;">✓</span> 1 <span style="color: red;">✗</span>
Signal Builder Baseline examples	2 <span style="color: green;">✓</span> 1 <span style="color: red;">✗</span>
Fast Accel	<span style="color: red;">✗</span>
Baseline Criteria Result	<span style="color: red;">✗</span>
gear	<span style="color: orange;">⚠</span>
throttle	<span style="color: red;">✗</span>
vehicle speed	<span style="color: red;">✗</span>
Decel	<span style="color: green;">✓</span>
- Test Specification Report:** A dialog box titled "Create a Test Specification Report" allows users to configure report details:
  - Title: Test Specification Report
  - Author: Author name
  - Include in Report: Test Details, Logged Signals, Callback Scripts, Coverage Settings, System Under Test, Custom Criteria, Test File Options, Iterations, External Inputs, Parameter Overrides, Logical and Temporal Assessments, Baseline Criteria, Equivalence Criteria, Configuration Settings.
  - Output Options: File Format (PDF), File Name (C:\Users\Desktop\newReport\_3.pdf).
  - Customization Templates: Test Suite Reporter, Test Case Reporter.
- Report Generated by Test Manager:** A PDF report showing:
  - Title: LandingGearControl-Regression Tests
  - Author: Jessica Johnson
  - Date: 20-Feb-2016 16:28:22
  - Test Environment: Platform: PCWIN64, MATLAB: (R2015a)
- Test Results Analysis:** A detailed view of the "Activate\_Pump" test case showing:
  - Expected Behavior: A plot showing the trigger condition becoming true and staying true for at least 2 seconds.
  - Actual Result: A plot showing the actual test results, with a failure indicated by a red vertical line.
  - Explanation: Assessment 'Activate\_Pump' - Trigger condition 'threshold' is true at [time]. Expected response is with a delay of at most 2 seconds, pumpCmd must stay true for at least 2 seconds.
  - Plots: A summary plot showing the test status (Fail, Pass, Untested) over time.

# Measure completeness of testing

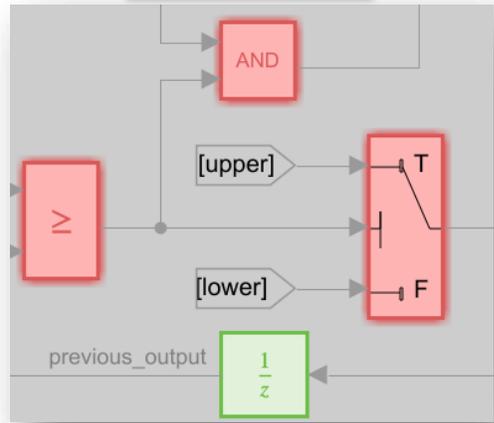
## Model Verification

- Manage requirements
- Systematically test
- **Measure model coverage**
- Check standard compliance
- Detect design errors
- Prove model behavior compliance



# Coverage Analysis to Measure Testing

Simulink®



Stateflow™



- Identify testing gaps
- Missing requirements
- Unintended functionality
- Design errors

Code

Coverage annotation

Links to model element

```
Code
rtwdemo_sil_topmodel.c
/* Output and update for enable system: '<Root>/CounterTypeB' */
100 static void CounterTypeB(void)
101 {
102     /* Outputs for Enabled SubSystem: '<Root>/CounterTypeB' incorporates:
103      * EnablePort: '<S2>/Enable'
104      */
105     if (enable) {
106         /* Outputs for '<Root>/CounterTypeB' incorporates:
107          * Inport: '<Root>/reset'
108          * Inport: '<Root>/ticks_to_count'
109          * Output: '<Root>/count_b'
110          * Sum: '<S2>/Add'
111          */
112     }
113     if (rtu.reset) {
114         rtv.count_b = 0;

```

Tooltip with code coverage results

Coverage Reports

Model Hierarchy/Complexity	Test 1									
	Decision	Condition	MCDC	Execution	Relational Boundary	Saturation on integer overflow				
1. <a href="#">sidemo_fuelsys</a>	80	34%	34%	7%	90%	10%	50%			
2. <a href="#">Engine Gas Dynamics</a>	13	71%	NA	NA	100%	50%	50%			
3. <a href="#">Mixing &amp; Combustion</a>	3	67%	NA	NA	100%	NA	50%			
4. <a href="#">EGO Sensor</a>	2	100%	NA	NA	100%	NA	NA			
5. <a href="#">System Lag</a>	NA	NA	NA	NA	100%	NA	NA			
6. <a href="#">Throttle &amp; Manifold</a>	10	73%	NA	NA	100%	50%	50%			
7. <a href="#">Intake Manifold</a>	2	100%	NA	NA	100%	NA	50%			
8. <a href="#">MATLAB Function</a>	2	100%	NA	NA	NA	NA	NA			
9. <a href="#">Throttle</a>	6	83%	NA	NA	100%	100%	50%			

# Test and Requirements Traceability in Coverage Results

NAME	STATUS
Results: 2020-Mar-02 22:14:00	6 <span style="color: green;">✔</span>
cruiseControlRBTcovTests	6 <span style="color: green;">✔</span>
Cruise Control Test Suite	6 <span style="color: green;">✔</span>
Brake Test	<span style="color: green;">✔</span>
Decrement Test	<span style="color: green;">✔</span>
Enable Test	<span style="color: green;">✔</span>
Increment Test	<span style="color: green;">✔</span>
Set Speed Test	<span style="color: green;">✔</span>
Throttle Test	<span style="color: green;">✔</span>

AGGREGATED COVERAGE RESULTS

Create a coverage report from coverage results to justify or exclude missing coverage. The filters and updated coverage values will be displayed with this result.

ANALYZED MODEL	REPORT	COMPLEXI...	DECISION	CONDITION	EXECUTION
cruiseControlRBTcovExample		8	100% <span style="color: blue;">█</span>	100% <span style="color: blue;">█</span>	100% <span style="color: blue;">█</span>

Coverage Details

2. Subsystem block "Controller"

[Justify or Exclude](#)

Parent: [cruiseControlRBTcovExample](#)  
Child Systems: [PI Controller](#)

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	7
Condition	NA	100% (12/12) condition outcomes
Decision	NA	100% (12/12) decision outcomes
Execution	NA	100% (17/17) objective outcomes

Logic block "Logical Operator"

[Justify or Exclude](#)

Requirement Testing Details

Implemented Requirements	Verified by Tests	Associated Runs
<a href="#">Brake disengages cruise control</a>	<a href="#">Brake Test</a>	T2
<a href="#">Engage cruise control</a>	<a href="#">Enable Test</a>	T1

Parent: [cruiseControlRBTcovExample.Controller](#)

Metric	Coverage
Cyclomatic Complexity	0
Condition	100% (6/6) condition outcomes
Execution	100% (1/1) objective outcomes

Conditions analyzed

\* All tests passed and shows 100% coverage

Requirements - cruiseControlRBTcovExample

View: Requirements

Index	ID	Summary	Implemented	Verified
1	SET_SPEED	Set speed	<span style="color: blue;">█</span>	<span style="color: green;">█</span>
2	BRAKE	Brake disengages cruise control	<span style="color: blue;">█</span>	<span style="color: green;">█</span>
3	ENABLE	Engage cruise control	<span style="color: blue;">█</span>	<span style="color: green;">█</span>
4	INCREMENT	Increment set speed	<span style="color: blue;">█</span>	<span style="color: green;">█</span>
5	DECREMENT	Decrement set speed	<span style="color: blue;">█</span>	<span style="color: green;">█</span>
6	THROTTLE	Throttle to maintain set speed	<span style="color: blue;">█</span>	<span style="color: green;">█</span>

The screenshot shows the MATLAB coverage tool interface. On the left, a tree view displays test results for 'Results: 2020-Mar-02 22:14:00' with a status of 6 green checkmarks. Below this, the 'cruiseControlRBTcovTests' suite is expanded, showing 'Cruise Control Test Suite' with 6 green checkmarks, and several individual tests: Brake Test, Decrement Test, Enable Test, Increment Test, Set Speed Test, and Throttle Test, all with green checkmarks.

The main window displays 'AGGREGATED COVERAGE RESULTS' for the analyzed model 'cruiseControlRBTcovExample'. A table shows the following coverage data:

ANALYZED MODEL	REPORT	COMPLEXI...	DECISION	CONDITION	EXECUTION
cruiseControlRBTcovExample	8		92%	100%	76%

At the bottom of the window, the checkbox 'Scope coverage results to linked requirements' is checked and circled in red. Other buttons include '+ Add Tests for Missing Coverage' and 'Export'.

- Checking coverage of tests with the linked requirements
  - 1) Testing(covered) with missing requirements
  - 2) Testing(covered) with requirements but not by linked tests
  - 3) Not testing(not covered)

NAME	STATUS
Results: 2020-Mar-02 22:14:00	6 ✓
cruiseControlRBTCovTests	6 ✓
Cruise Control Test Suite	6 ✓
Brake Test	✓
Decrement Test	✓
Enable Test	✓
Increment Test	✓
Set Speed Test	✓
Throttle Test	✓

AGGREGATED COVERAGE RESULTS

Create a coverage report from coverage results to justify or exclude missing coverage. The filters and updated coverage values will be displayed with this result.

ANALYZED MODEL: **cruiseControlRBTCovExample** | 8 | 92% | 100% | 76%

1) Missing requirement links identified

Coverage Details

Constant block "Constant1"

Justify or Exclude

Parent: [cruiseControlRBTCovExample/Controller](#)

Uncovered Links:

Metric	Coverage
Cyclomatic Complexity	0
Execution	0% (0/1) objective outcomes

Execution analyzed

Block executed	Coverage
	0%
	11

Constant block "Constant3"

Justify or Exclude

Parent: [cruiseControlRBTCovExample/Controller](#)

Uncovered Links:

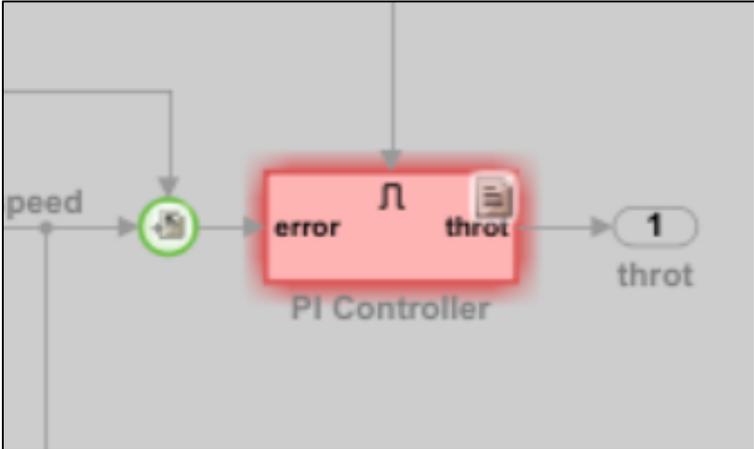
Requirements - cruiseControlRBTCovExample

View: Requirements

Index	ID	Summary	Implemented
6	THRITTLE	Throttle to maintain set speed	

\* Each test only contributes coverage for the corresponding models that implement the requirements

# Test and Requirements Traceability in Coverage Results

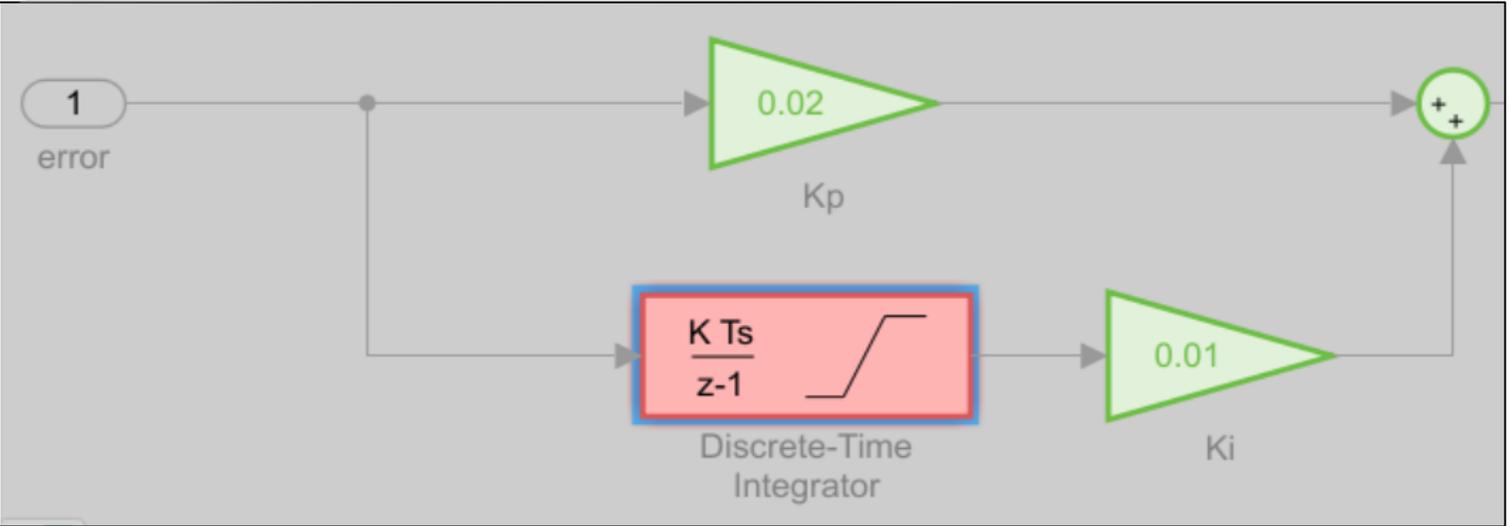


**3. SubSystem block "PI Controller"**

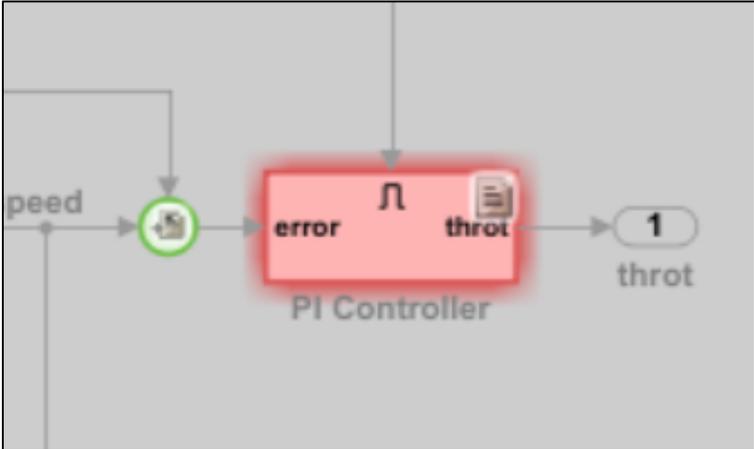
[Justify or Exclude](#)

Requirement Testing Details

Implemented Requirements	Verified by Tests	Associated Runs
<a href="#">Throttle to maintain set speed</a>	<a href="#">Throttle Test</a>	<a href="#">T6</a>



# Test and Requirements Traceability in Coverage Results



### 3. SubSystem block "PI Controller"

[Justify or Exclude](#)

#### Requirement Testing Details

Implemented Requirements	Verified by Tests	Associated Runs
<a href="#">Throttle to maintain set speed</a>	<a href="#">Throttle Test</a>	<a href="#">T6</a>

### DiscreteIntegrator block "Discrete-Time Integrator"

[Justify or Exclude](#)

Parent: [cruiseControlRBTCovExample/Controller/PI Controller](#)

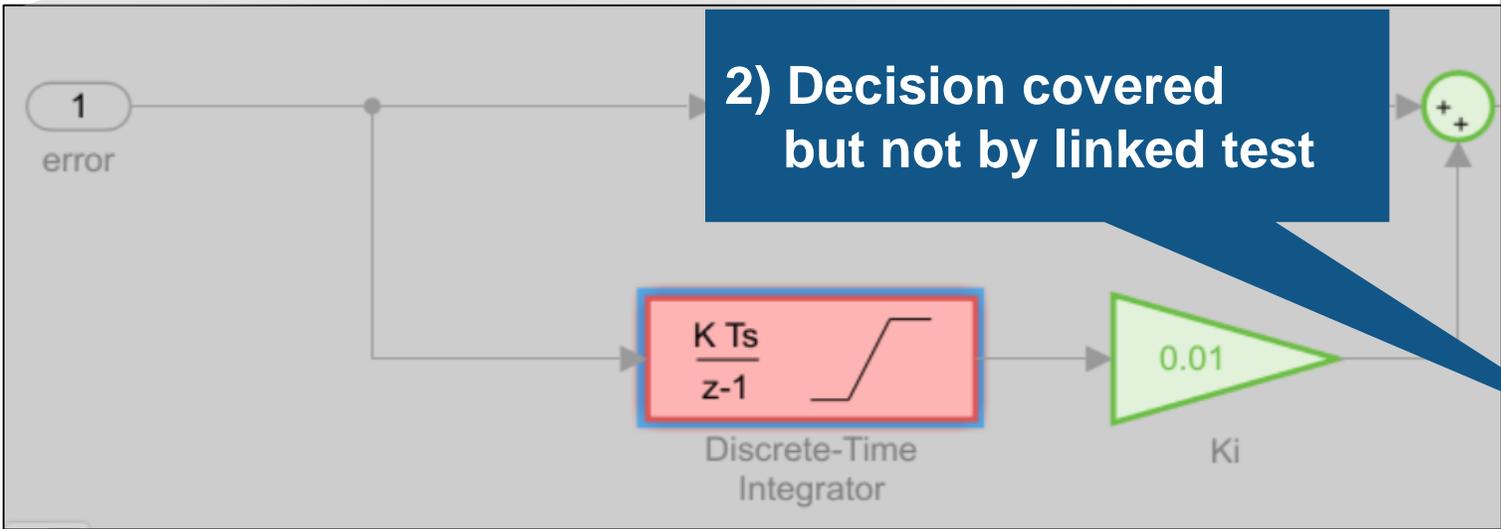
Uncovered Links: ◀

Metric	Coverage
Cyclomatic Complexity	2
Decision	75% (3/4) decision outcomes
Execution	100% (1/1) objective outcomes

#### Decisions analyzed

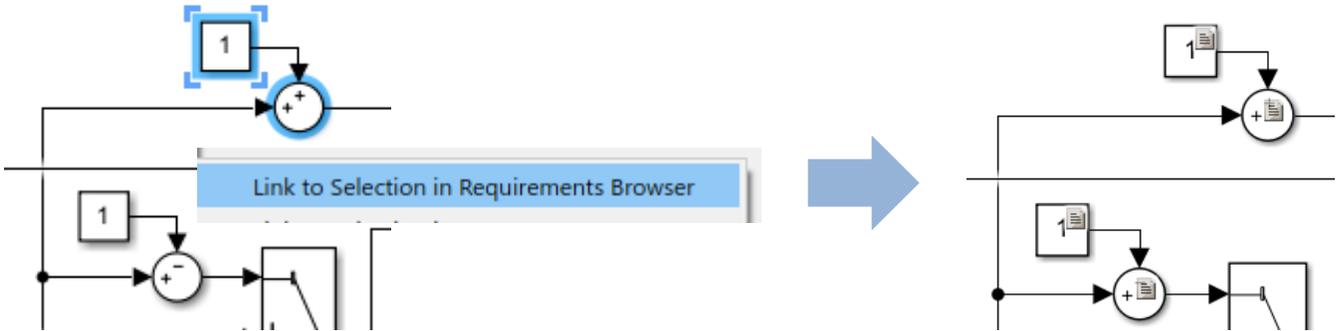
integration result <= lower limit	100%
false	391/801 <a href="#">T6</a>
true	410/801 <a href="#">T6</a>
integration result >= upper limit	50%
false	801/801 <a href="#">T6</a>
true	0/801 <a href="#">T4</a>

**2) Decision covered but not by linked test**



# Address missing Requirements-Based Test Coverage

1) Add missing implementation links to requirements



2) Update test to increase target speed to be covered by the linked test

▼ INPUTS\* ?

- Include input data in test result
- Stop simulation at last time point

EXTERNAL INPUTS

NAME	FILE	SHEET	STATUS
<input type="checkbox"/> td_throttle.mat	C:\Demos\examples\R2020a\sir		<a href="#">Mapped</a>
<input checked="" type="checkbox"/> td_throttle_updated.mat	C:\Demos\examples\R2020a\sir		<a href="#">Mapped</a>

# 100% Coverage but Testing Identified Error in Implementation

Results: 2020-Mar-02 23:59:38	5	1
cruiseControlRBTCovTests	5	1
Cruise Control Test Suite	5	1
Brake Test		
Decrement Test		
Enable Test		
Increment Test		
Set Speed Test		
Throttle Test		

AGGREGATED COVERAGE RESULTS

Create a coverage report from coverage results to justify or exclude missing coverage. The filters and updated coverage values will be displayed with this result.

ANALYZED MODEL	REPORT	COMPLEXI...	DECISION	CONDITION	EXECUTION
cruiseControlRBTCovExample		8	100%	100%	100%

Scope coverage results to linked requirements

DiscreteIntegrator block "[Discrete-Time Integrator](#)"

[Justify or Exclude](#)

Parent: [cruiseControlRBTCovExample/Controller/PI Controller](#)

Metric	Coverage
Cyclomatic Complexity	2
Decision	100% (4/4) decision outcomes
Execution	100% (1/1) objective outcomes

**Decisions analyzed**

integration result <= lower limit	100%
false	394/950 <a href="#">T6</a>
true	556/950 <a href="#">T6</a>
integration result >= upper limit	100%
false	950/1001 <a href="#">T6</a>
true	51/1001 <a href="#">T6</a>

# Additional Testing Identified Error in Implementation

Results: 2020-Mar-02 23:59:38	5	1
cruiseControlRBTcovTests	5	1
Cruise Control Test Suite	5	1
Brake Test		
Decrement Test		
Enable Test		
Increment Test		
Set Speed Test		
Throttle Test		

**✖ Throttle changes within limits**

ASSESSMENT

► At any point of time, throttle\_deriv must be greater than -1 and less than 1

SYMBOLS

► throttle\_deriv

Error 1 of 1

**Expected Behavior**

**Actual Result**

**Explanation**

Assessment 'Throttle changes within limits' failed from 10.5 s to 12 s.

- Expected 'throttle\_deriv' to be greater than '-1' and less than '1'.
- At 11.67 s, expected value to be greater than -1 and less than 1, actual value is **2.309499999999999**.

[https://kr.mathworks.com/help/sitest/ug/test-coverage-for-requirements-based-testing.html?s\\_tid=srchtitle](https://kr.mathworks.com/help/sitest/ug/test-coverage-for-requirements-based-testing.html?s_tid=srchtitle)

# Scoped Model Coverage to Requirements-Based Tests

The screenshot shows the Test Manager window with the 'Results and Artifacts' tab selected. The left pane shows a tree view of test results for '2019-Oct-02 19:02:58', including 'dTestReqLinkBasic\_Tests' and 'MyTestSuite'. The right pane shows 'AGGREGATED COVERAGE RESULTS' for 'mTestReqLinkBasic' with a complexity of 5, 33% decision coverage, and 25% execution coverage. A red box highlights the checkbox 'Scope coverage results to linked requirements' which is checked.

**DO-178C 6.4.4.2** ... coverage information collected during requirements-based testing to confirm that ...

MultiPortSwitch block "MPSwitch1"

Requirement Testing Details

Implemented Requirements	Verified by Tests	Associated Runs
<a href="#">Requirement 1</a>	<a href="#">Testcase 1</a>	<a href="#">T1</a>

Metric Coverage

Cyclomatic Complexity 2

Decision 33% (1/3) decision outcomes

Execution 100% (1/1) objective outcomes

Decisions analyzed

truncated input value	33%
= 1 (output is from input port 1)	51/51 <a href="#">T1</a>
= 2 (output is from input port 2)	0/51 <a href="#">T2</a>
= *,3 (output is from input port 3)	0/51 <a href="#">T2</a>

Hit by linked RBT -- Satisfied

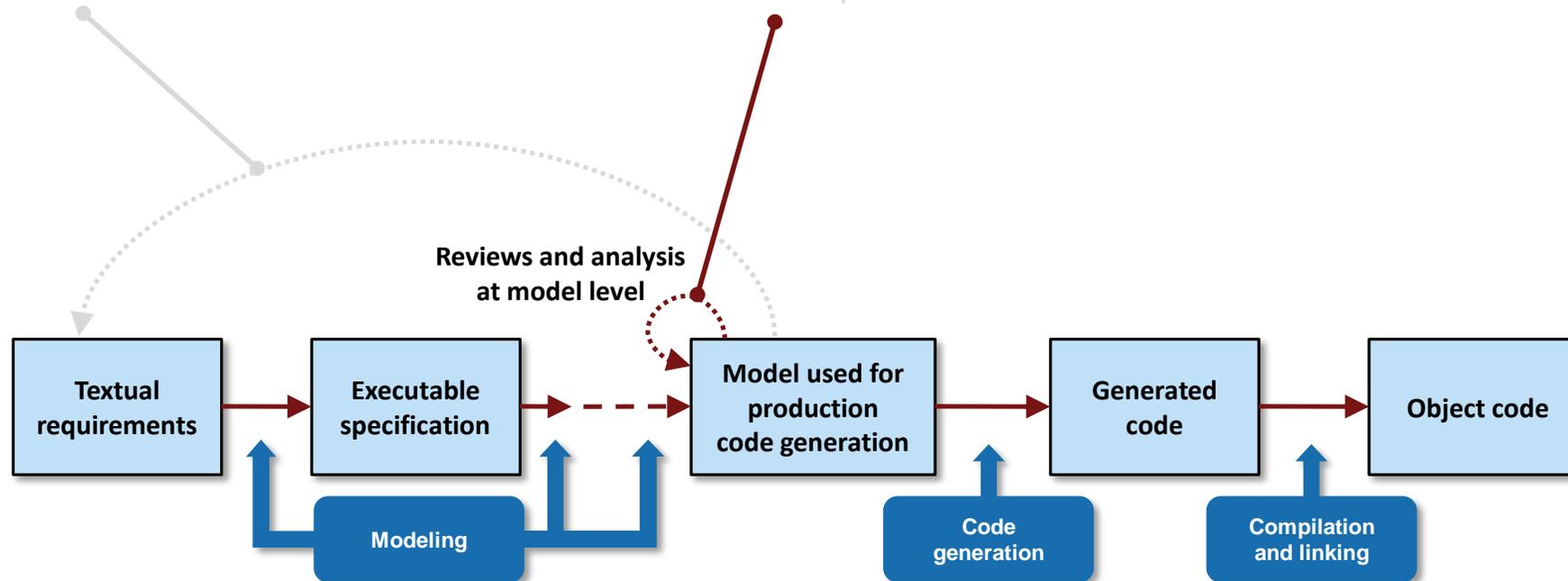
Hit, but not by linked RBT -- Unsatisfied

RBT (Requirements-Based Tests)

# Check Standard Compliance

## Model Verification

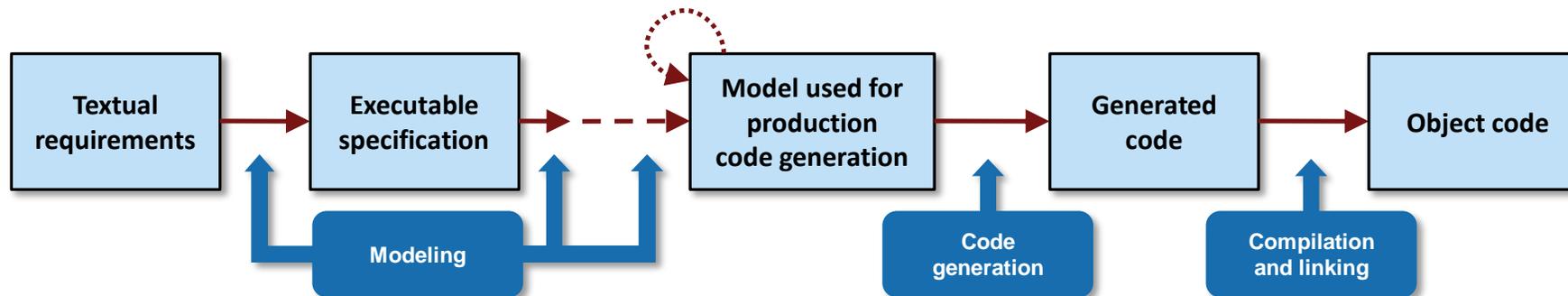
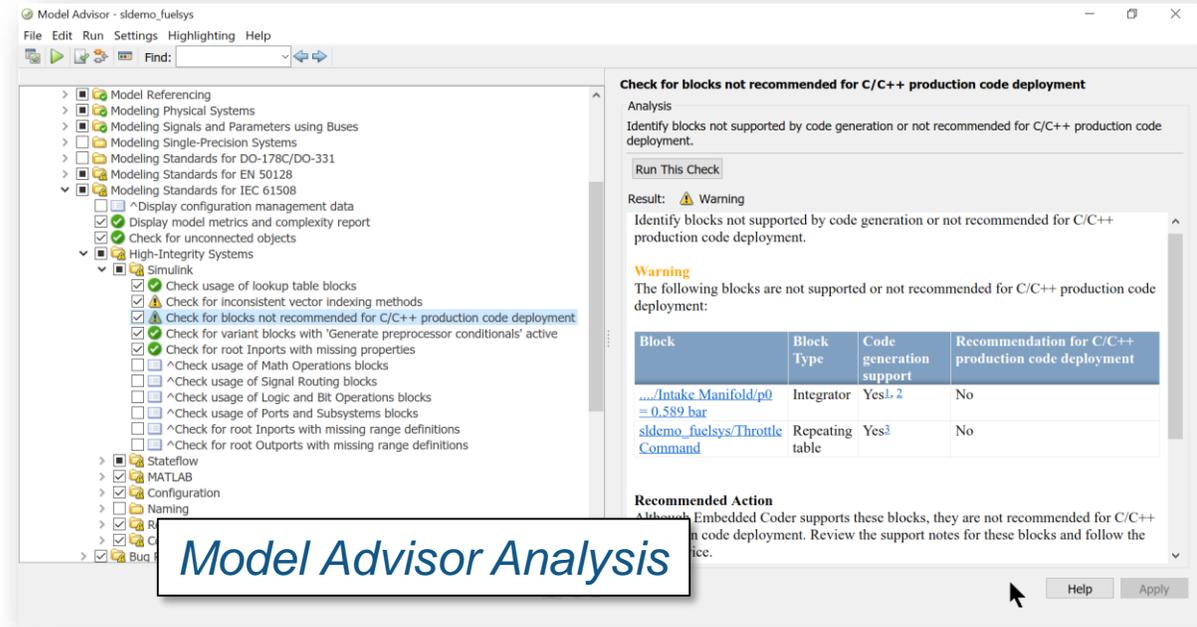
- Manage requirements
- Systematically test
- Measure model coverage
- **Check standard compliance**
- Detect design errors
- Prove model behavior compliance



# Verify Design to Guidelines and Standards

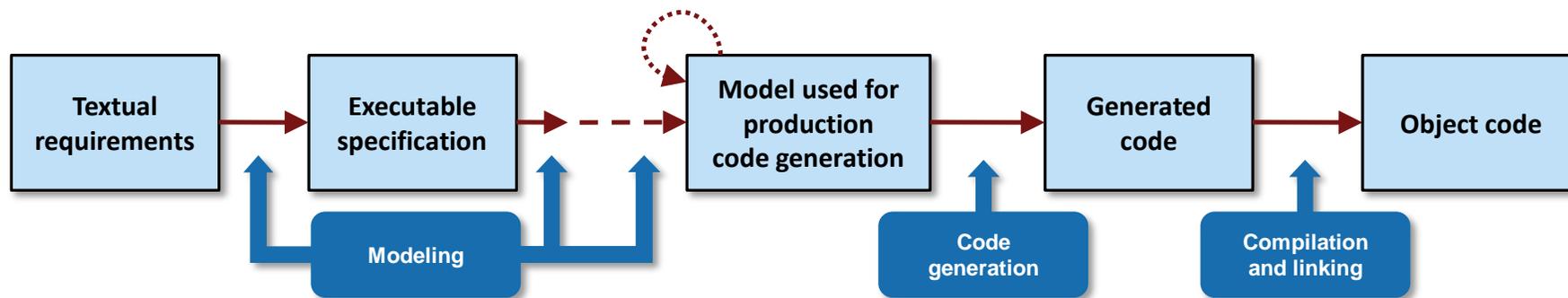
Check for:

- Readability and Semantics
- Performance and Efficiency
- Clones (for optimization)
- And more.....



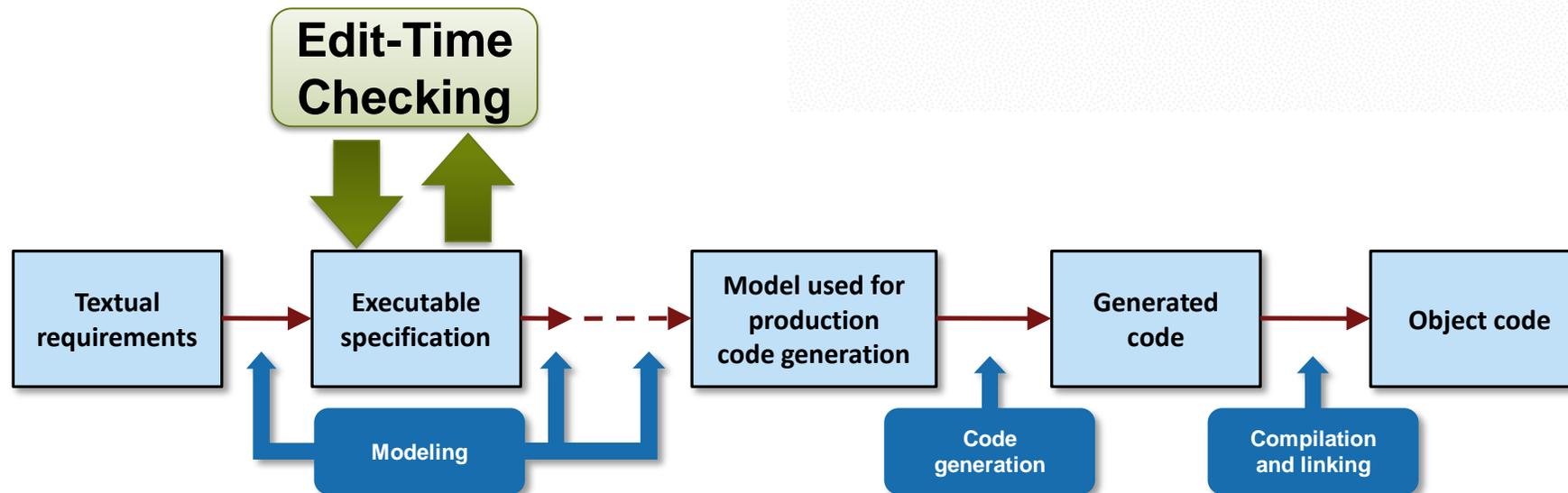
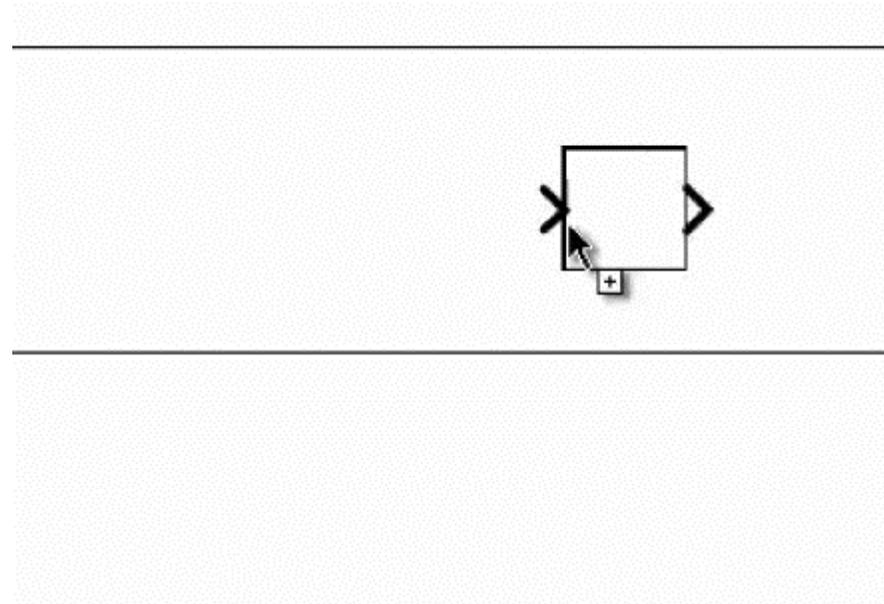
# Built in Checks for Industry Standards and Guidelines

- DO-178/DO-331
- MISRA C:2012
- ISO 26262
- CERT C, CWE, ISO/IEC TS 17961
- IEC 61508
- MAB (MathWorks Advisory Board)
- IEC 62304
- JMAAB (Japan MATLAB Automotive Advisory Board)
- EN 50128



# Shift Verification Earlier With Edit-Time Checking

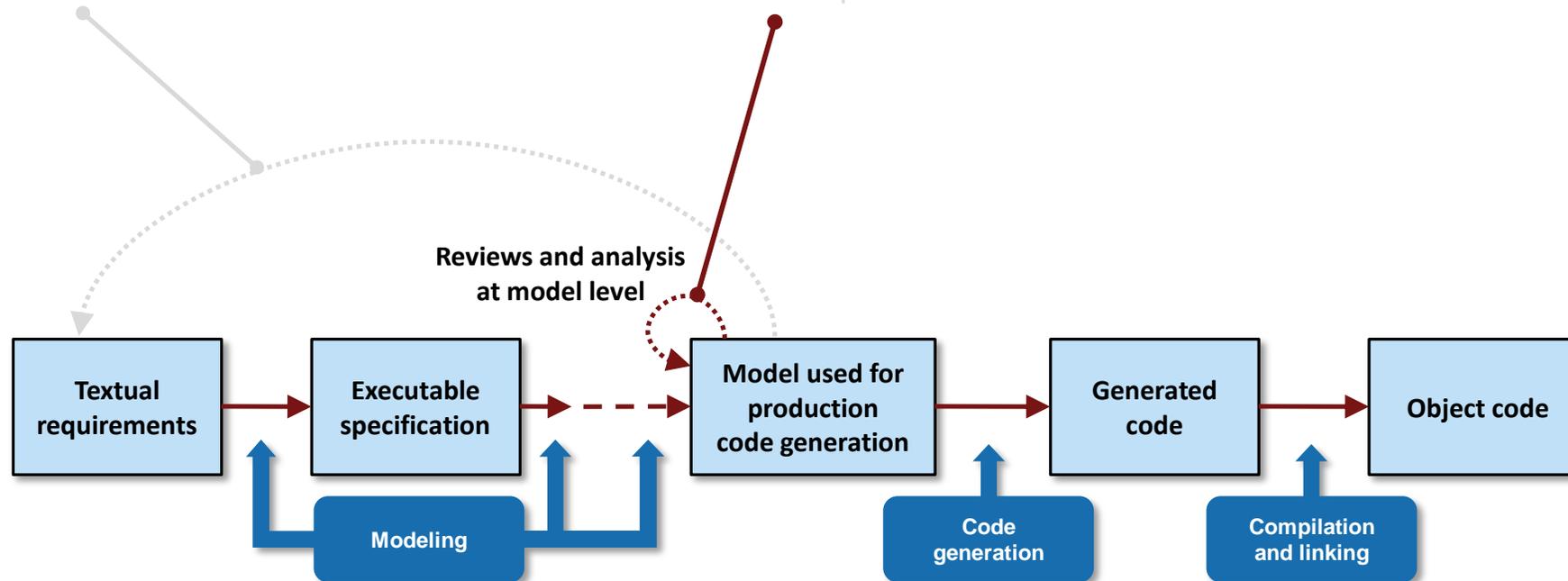
- Highlight violations as you edit
- Fix issues earlier
- Avoid rework



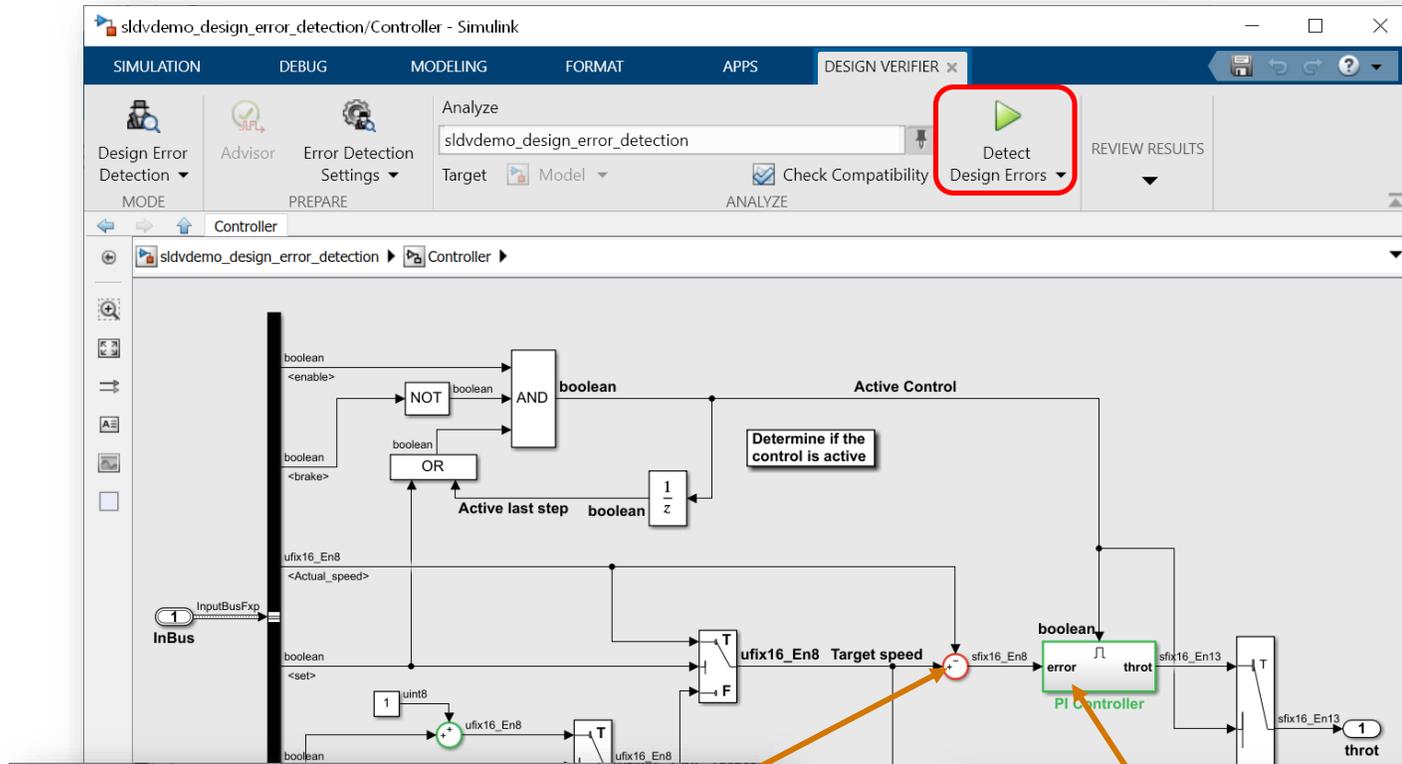
# Detect Design Errors with Formal Methods

## Model Verification

- Manage requirements
- Systematically test
- Measure model coverage
- Check standard compliance
- **Detect design errors**
- Prove model behavior compliance



# Detect Design Errors Using Formal Methods



- Find design errors
  - Integer overflow
  - Dead Logic
  - Division by zero
  - Array out-of-bounds
  - Range violations

- Generate counter example to reproduce error

Results: sldvdemo\_design\_error\_detection

[Back to summary](#)

sldvdemo\_design\_error\_detection/Controller/Sum1

Integer overflow Objectives

Overflow **Error - needs simulation** [- View test case](#) [Justify](#)

Derived Ranges:

Output 1: [-128..127.99609375]

Results: sldvdemo\_design\_error\_detection

[Back to summary](#)

sldvdemo\_design\_error\_detection/Controller/PI Controller/Kp

Integer overflow Objectives

Overflow **Valid**

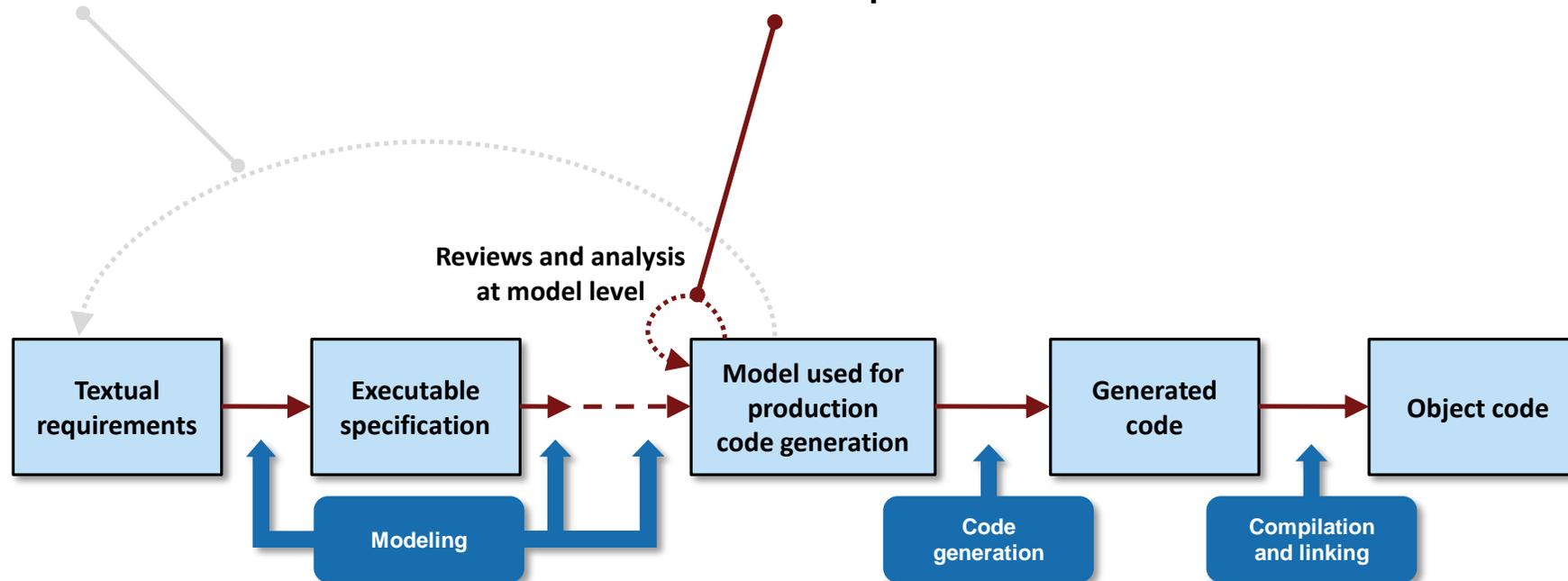
Derived Ranges:

Output 1: [-2.56005859375..2.559814453125]

# Prove Model Behavior Compliance

## Model Verification

- Manage requirements
- Systematically test
- Measure model coverage
- Check standard compliance
- Detect design errors
- **Prove model behavior compliance**

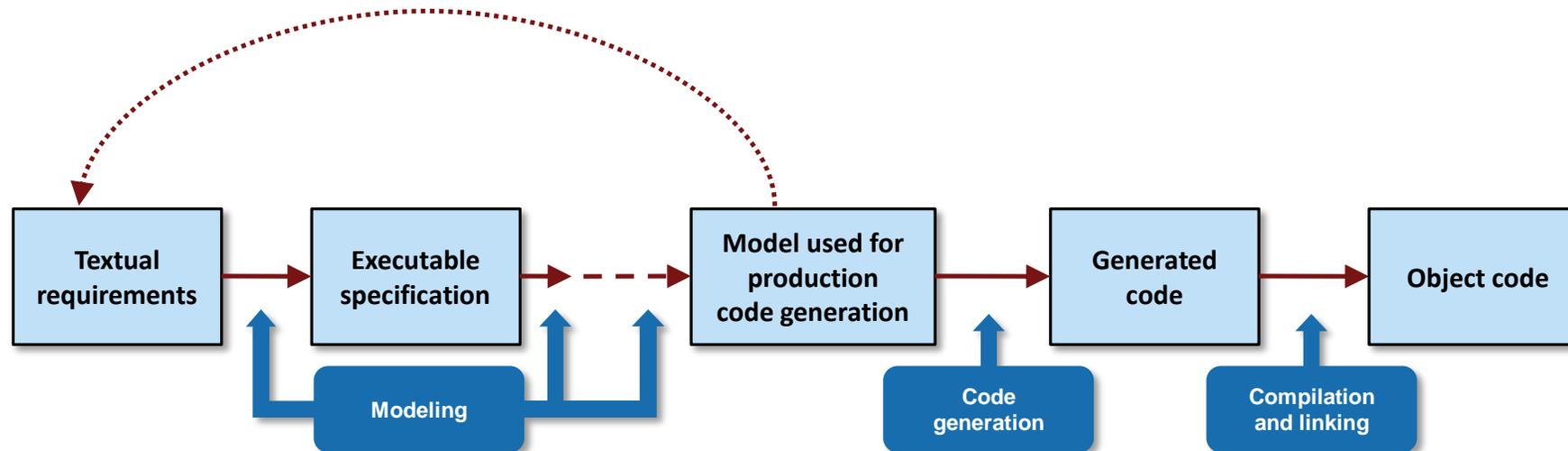


# Proving Model Meets Requirements

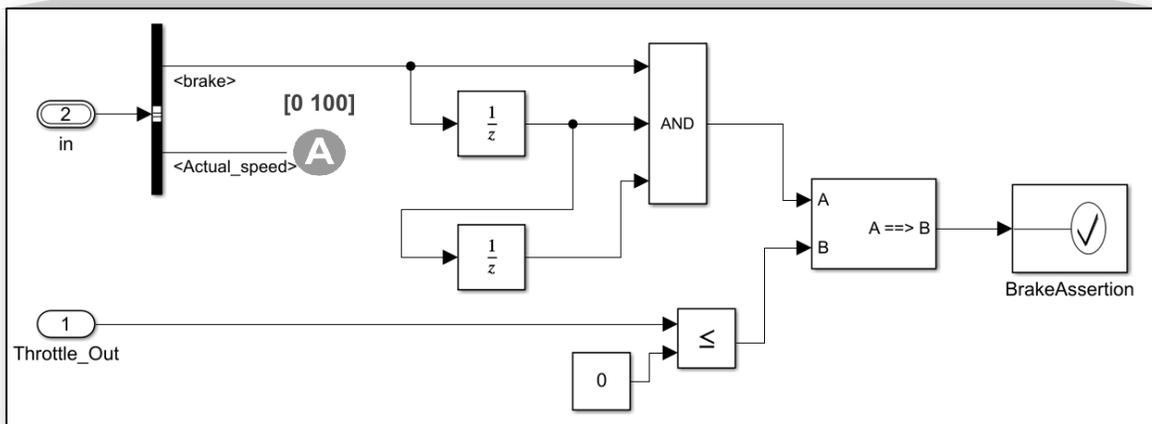
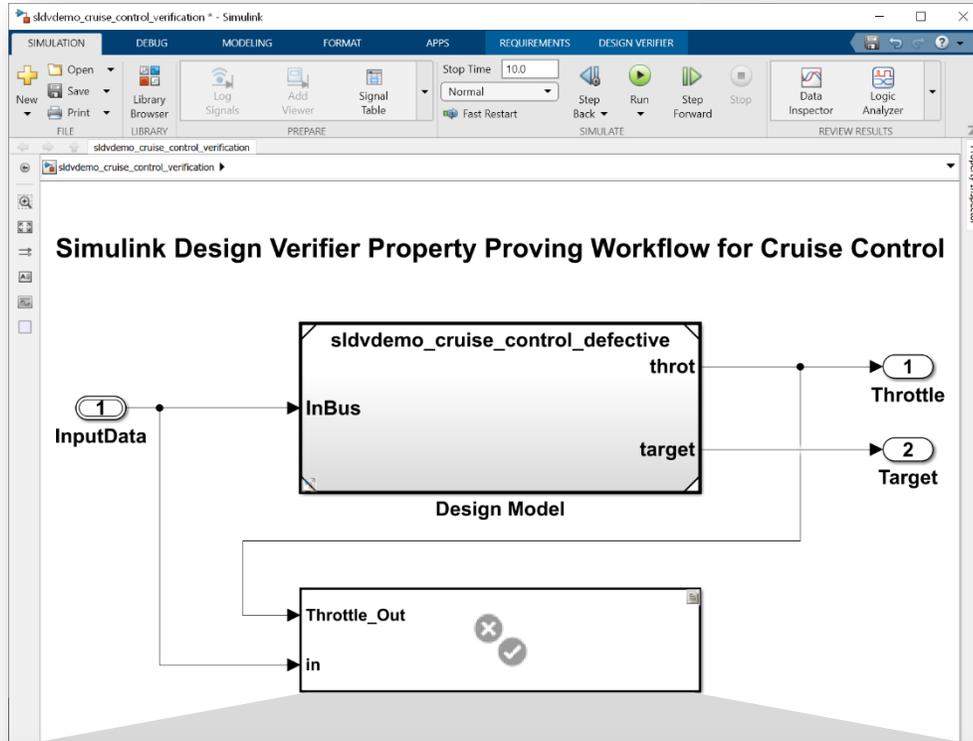
- Need to ensure the design performs correctly

## Safety Requirement:

When the brake is applied for three consecutive steps, the throttle shall go to zero.

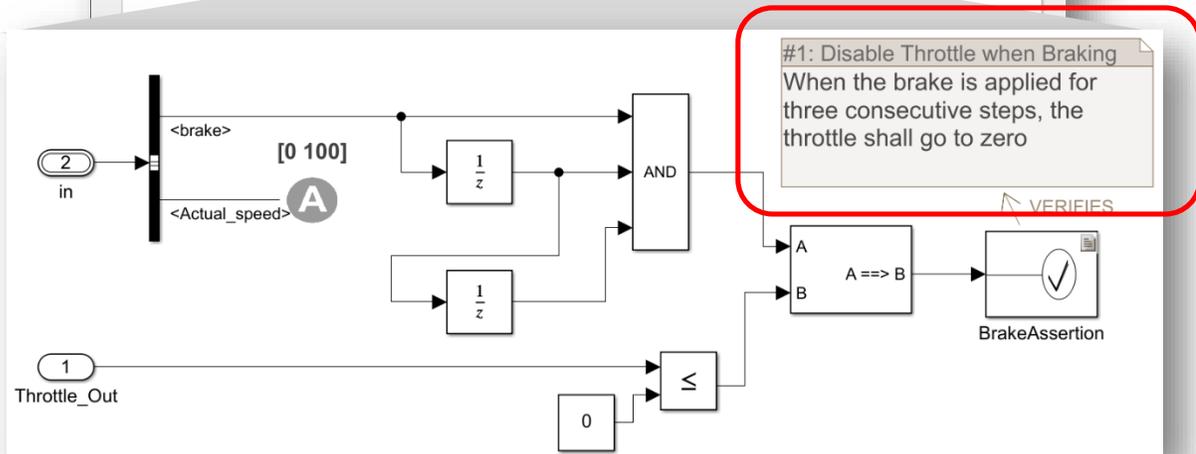
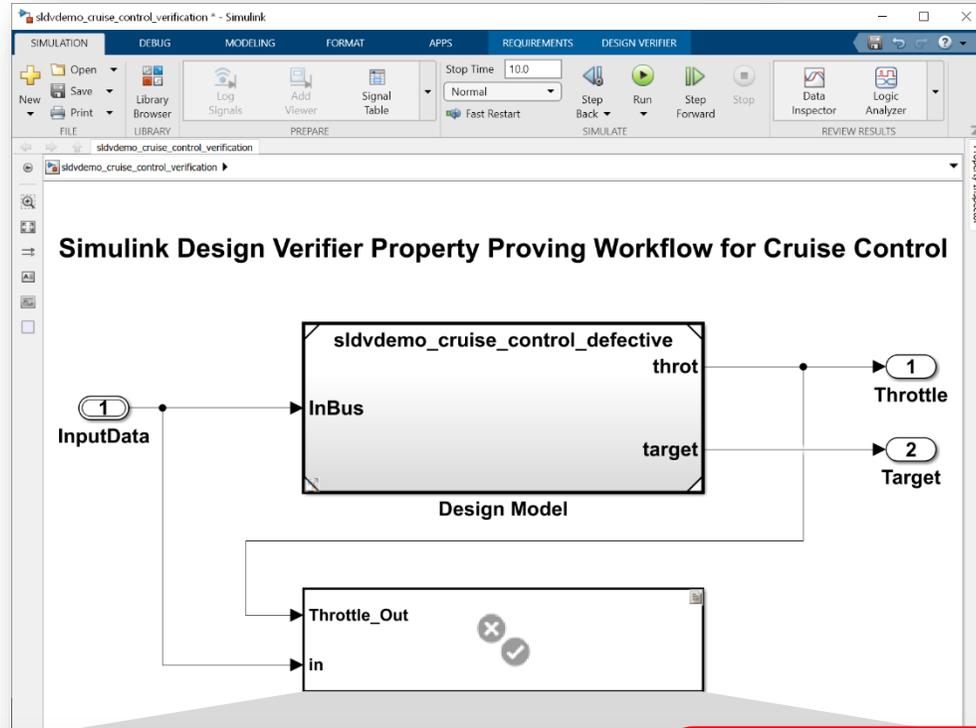


# Model Functional and Safety Requirements

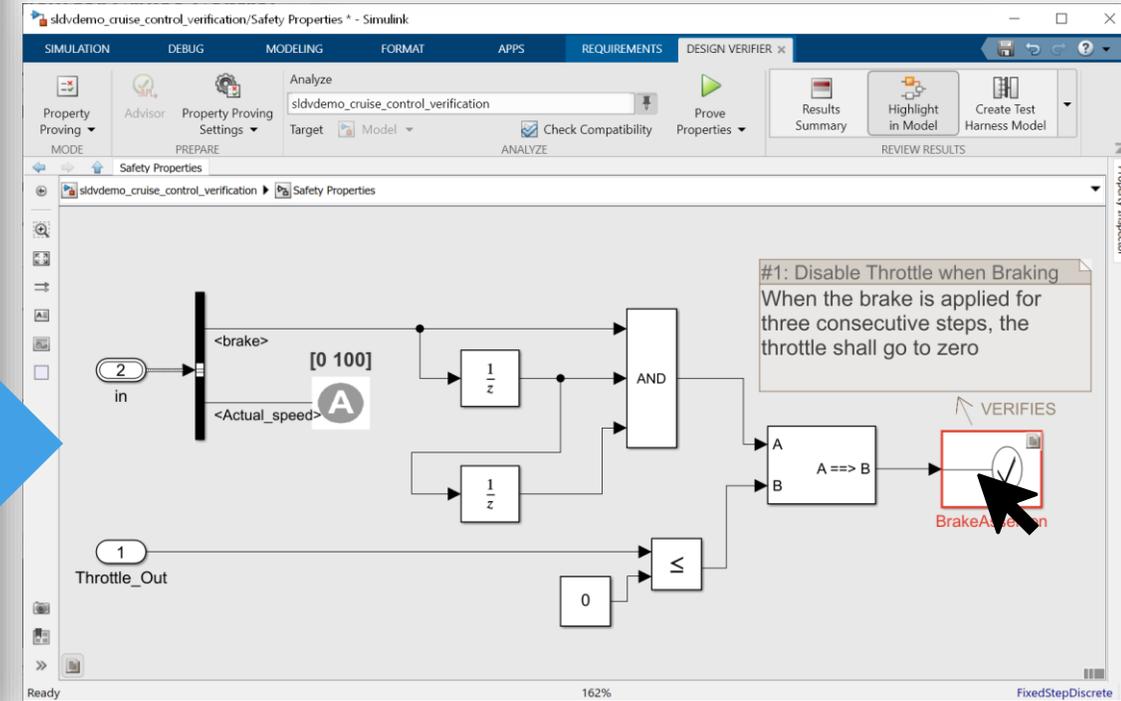
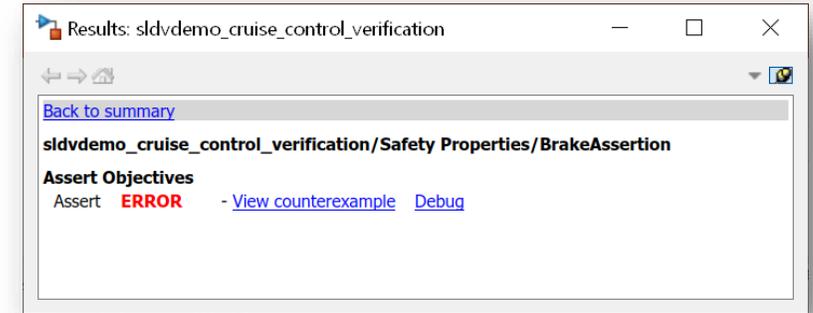
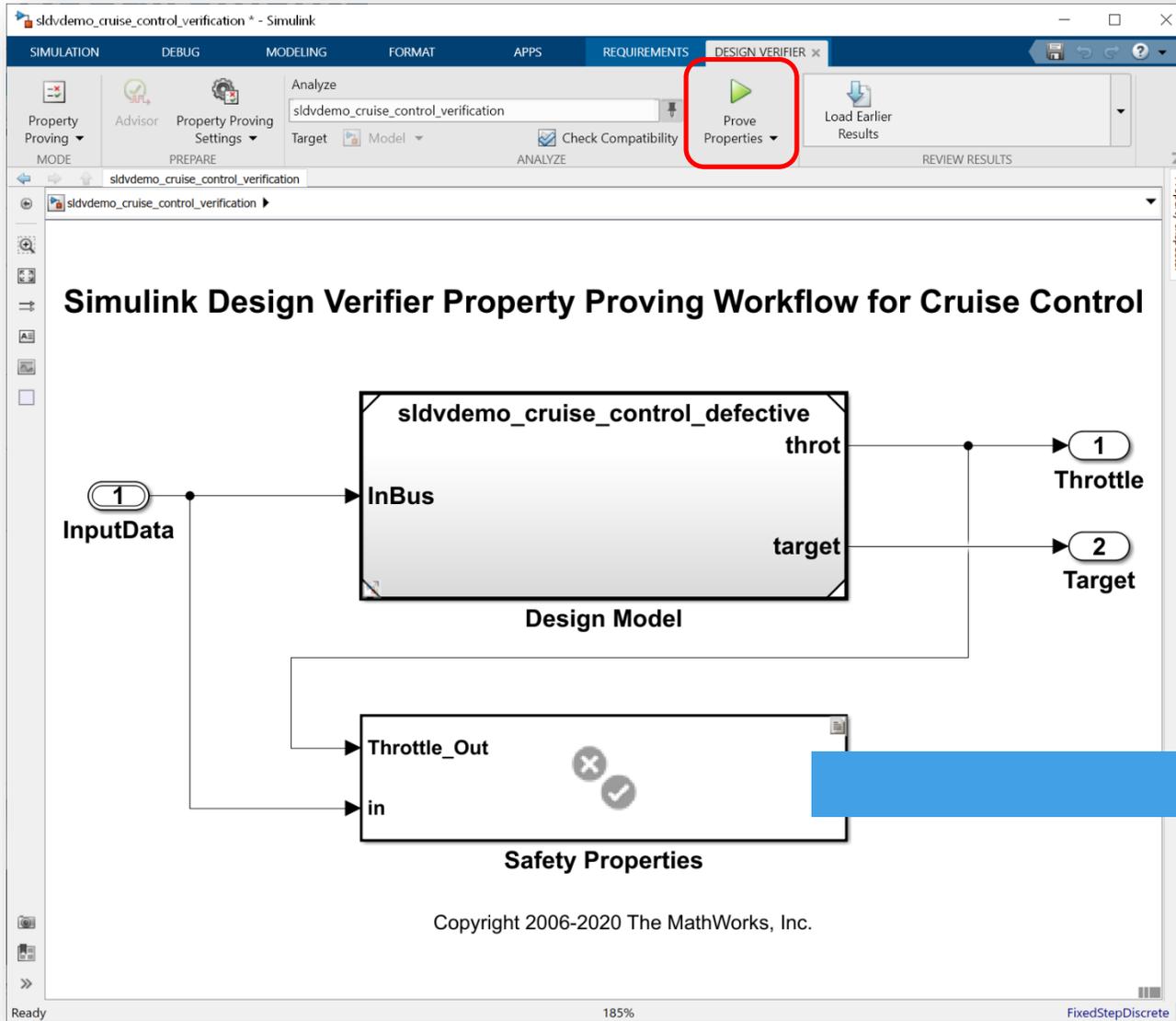


<https://kr.mathworks.com/help/sldv/ug/property-proving-workflow-for-cruise-control.html>

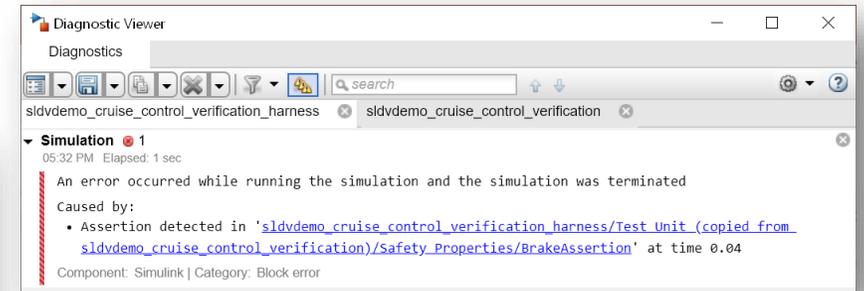
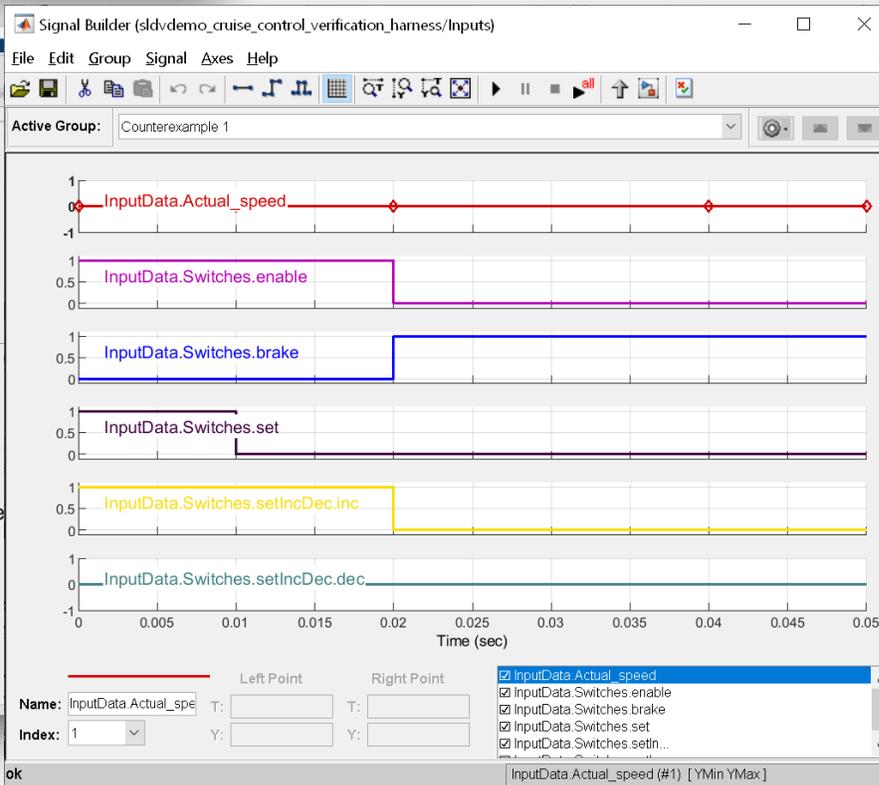
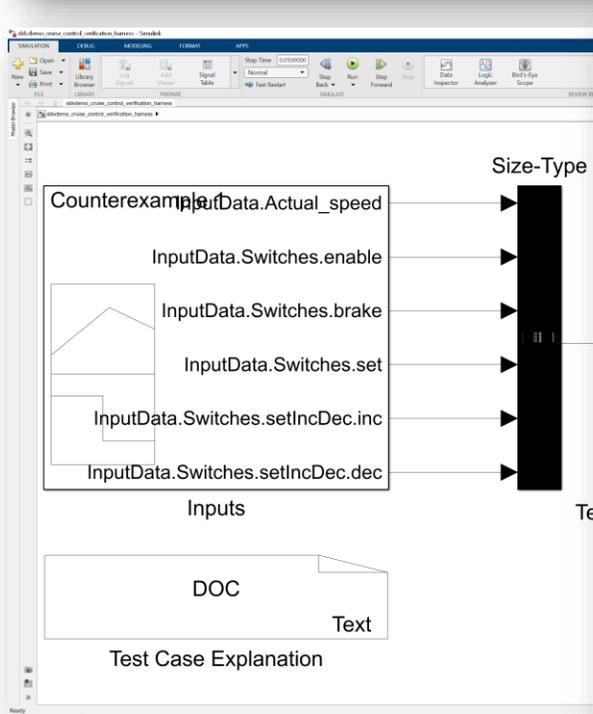
# Link Requirements to Properties



# Prove That Design Meets Requirements



# Prove That Design Meets Requirements



# Debugging Property Proving Violations

The screenshot shows the Simulink Model Slicer interface. The main workspace displays a block diagram with red annotations. A yellow banner at the top of the workspace reads: "Model Slicer setup is now complete. Use the Step Back/Step Forward buttons to debug." The diagram includes an input block 'in' with a value of 2, a gain block '1/z', an AND gate, a block 'A ==> B', and a BrakeAssertion block. The BrakeAssertion block is highlighted in red and contains a checkmark icon. The right-hand pane, titled "Model Slicer", shows the configuration for the selected slice. It includes a "Slice configuration list" with the following details:

- Name: sldvdemo\_cruise\_control\_verification/Safety Properties/BrakeAssertion : Assert
- Description: This slice configuration is autogenerated for debugging the falsified property 'sldvdemo\_cruise\_control\_verification/Safety Properties/BrakeAssertion : Assert'.
- Signal propagation: upstream
- Starting Points: BrakeAssertion

At the bottom of the right-hand pane, there are checkboxes for "Simulation time window (Enabled)", "Display port value labels", and "Display control dependencies". The "Time window" section shows "Start 0" and "Stop 0.04". The status bar at the bottom indicates "Paused after final step", "View diagnostics", "158%", "T=0.040", "Paused", and "FixedStepDiscrete".

The screenshot shows the "Results: sldvdemo\_cruise\_control\_verification" window. It contains a "Back to summary" link and the following text:

**sldvdemo\_cruise\_control\_verification/Safety Properties/BrakeAssertion**

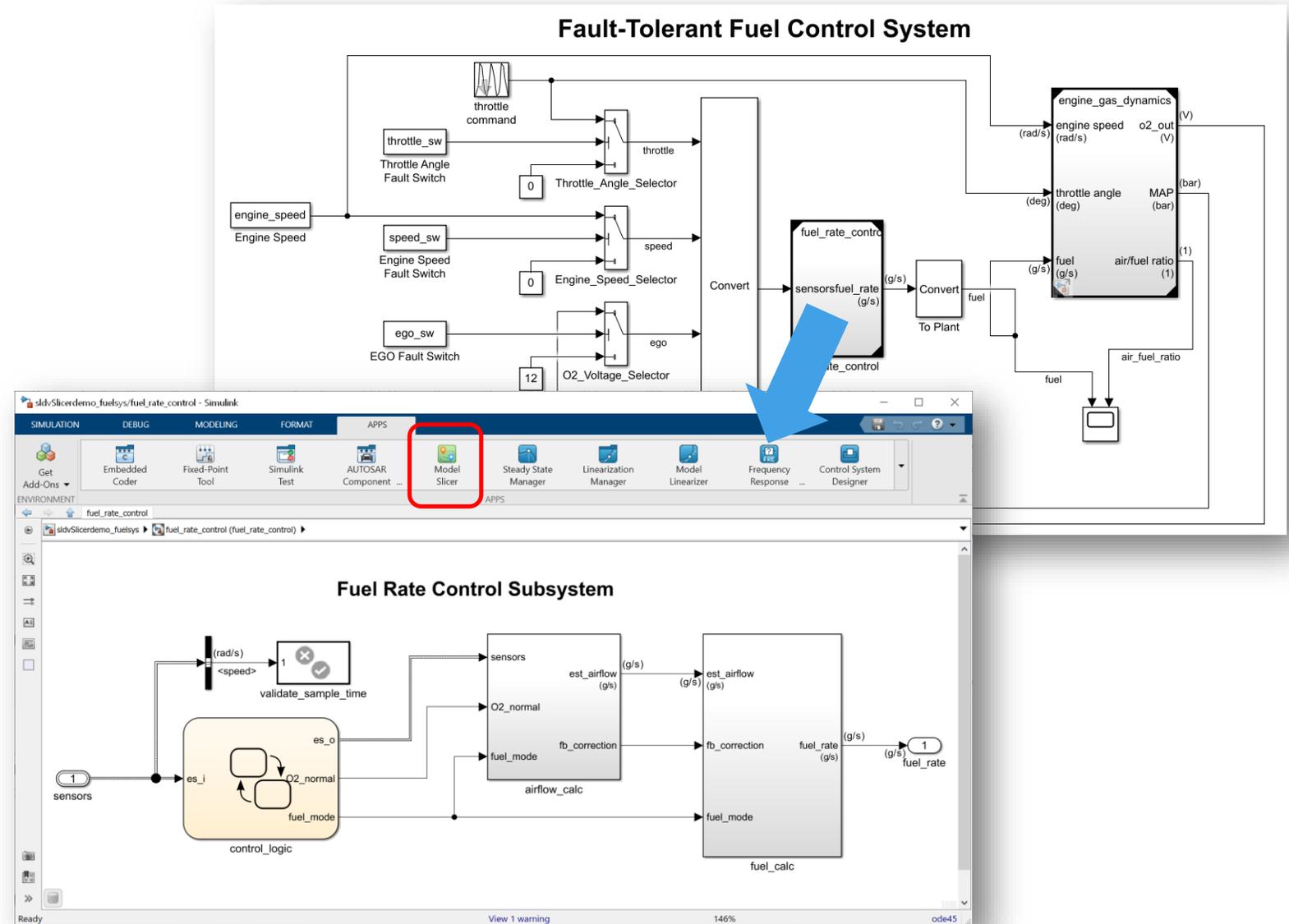
**Assert Objectives**  
Assert **ERROR** - [View counterexample](#) **Debug**

A red box highlights the "Debug" button. Below the window is a callout box with a bug icon and the text "Debug Using Slicer".

# Resolve Unexpected Behavior in a Model with Model Slicer

## Isolate

Find the area of the model responsible for unexpected behavior



# Resolve Unexpected Behavior in a Model with Model Slicer

## Isolate

Find the area of the model responsible for unexpected behavior



## Analyze dependencies

Understand data & control dependencies in large or complex models

The screenshot displays the Simulink Model Slicer interface for a 'fuel\_rate\_control' model. A dialog box titled 'Record simulation time window: fuel\_rate\_control' is open, prompting the user to specify a stop time (set to 20) and offering options for logging and saving the slice. The main workspace shows a block diagram of the 'Fuel Rate Control Subsystem' with components like 'sensors', 'control\_logic', 'fuel\_mode', 'airflow\_calc', and 'fuel\_calc'. The 'Model Slicer' panel on the right shows a 'Starting Points' list with 'fuel\_calc.1' and 'control\_logic.3' highlighted in red. The 'Run simulation' button in the 'Simulation time window' section is also highlighted in red.

# Resolve Unexpected Behavior in a Model with Model Slicer

## Isolate

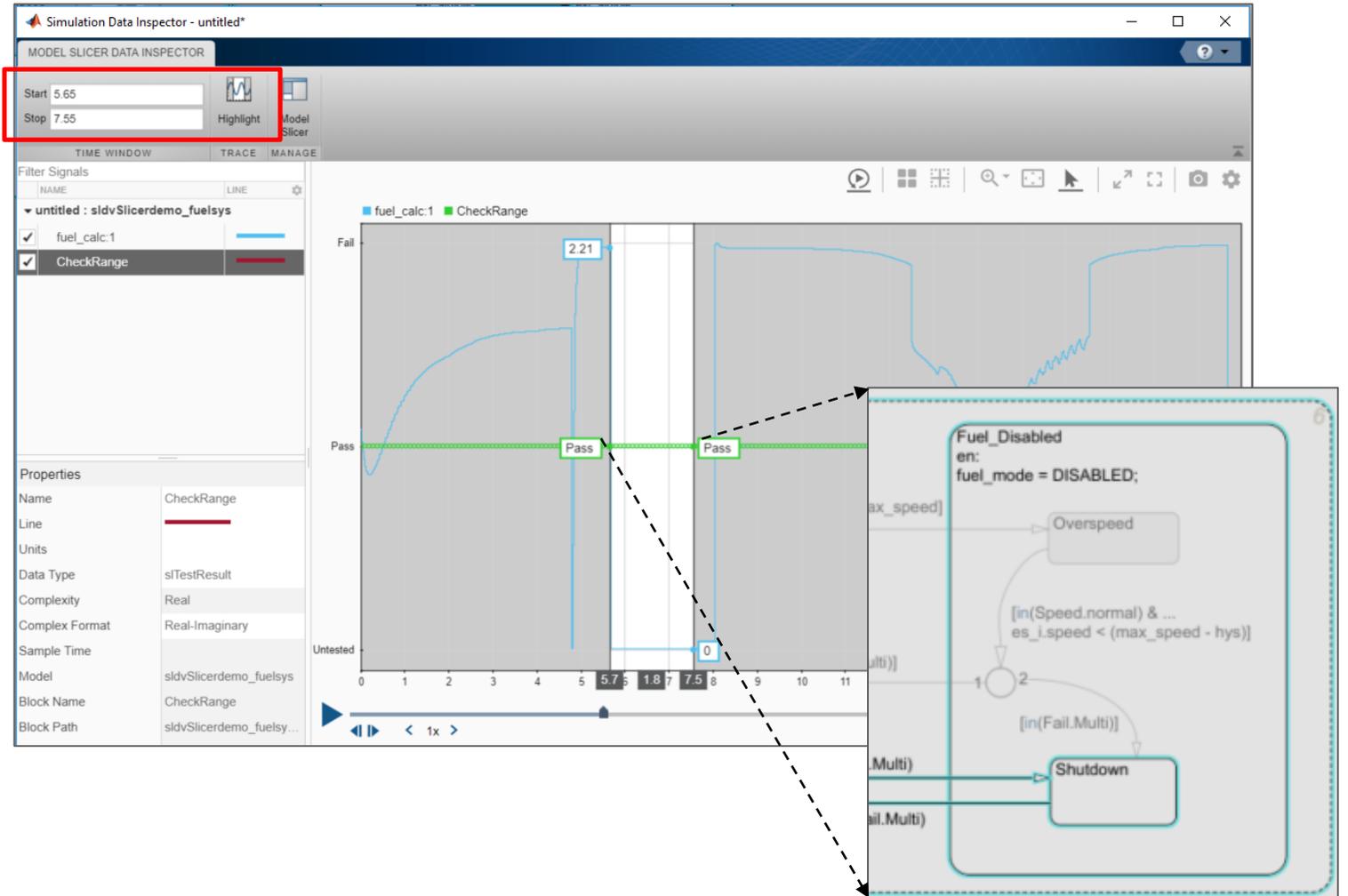
Find the area of the model responsible for unexpected behavior

## Analyze dependencies

Understand data & control dependencies in large or complex models

## Inspect slice regions

Highlight model slices for time windows or failure states & transitions for state flow.



# Resolve Unexpected Behavior in a Model with Model Slicer

## Isolate

Find the area of the model responsible for unexpected behavior

## Analyze dependencies

Understand data & control dependencies in large or complex models

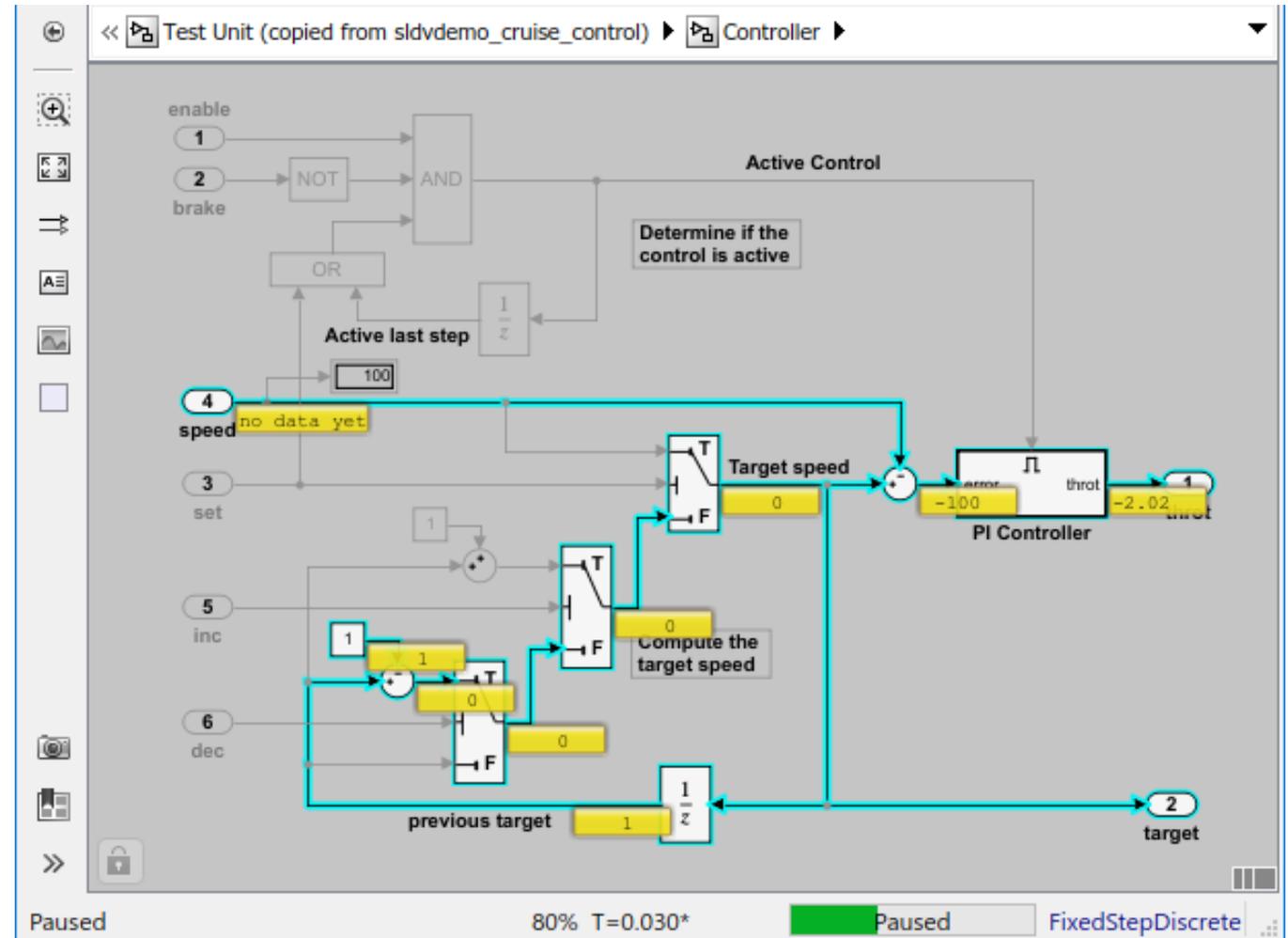
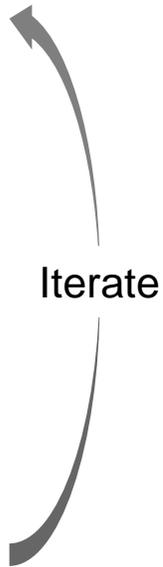
## Inspect slice regions

Highlight model slices for time windows or failure states & transitions for state flow.

## Debug simulation behavior

Step through precompiled slices to understand signal and port value propagation

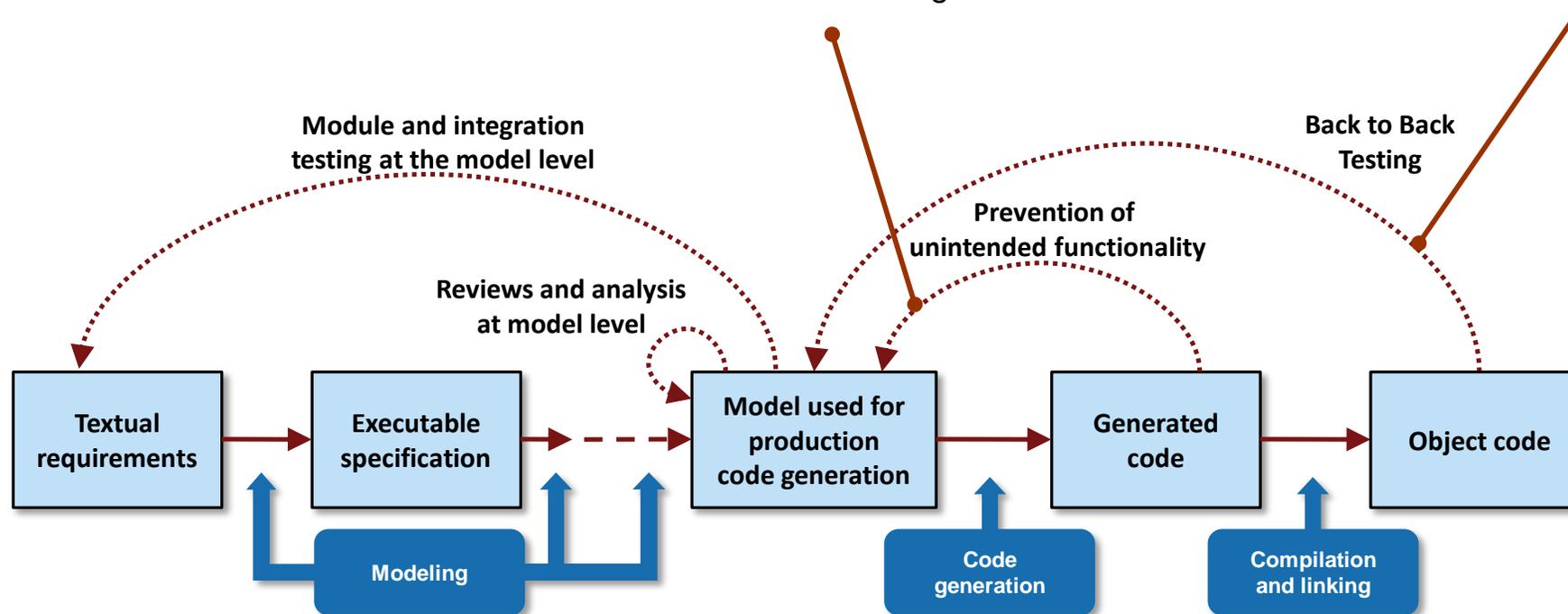
Correct Model



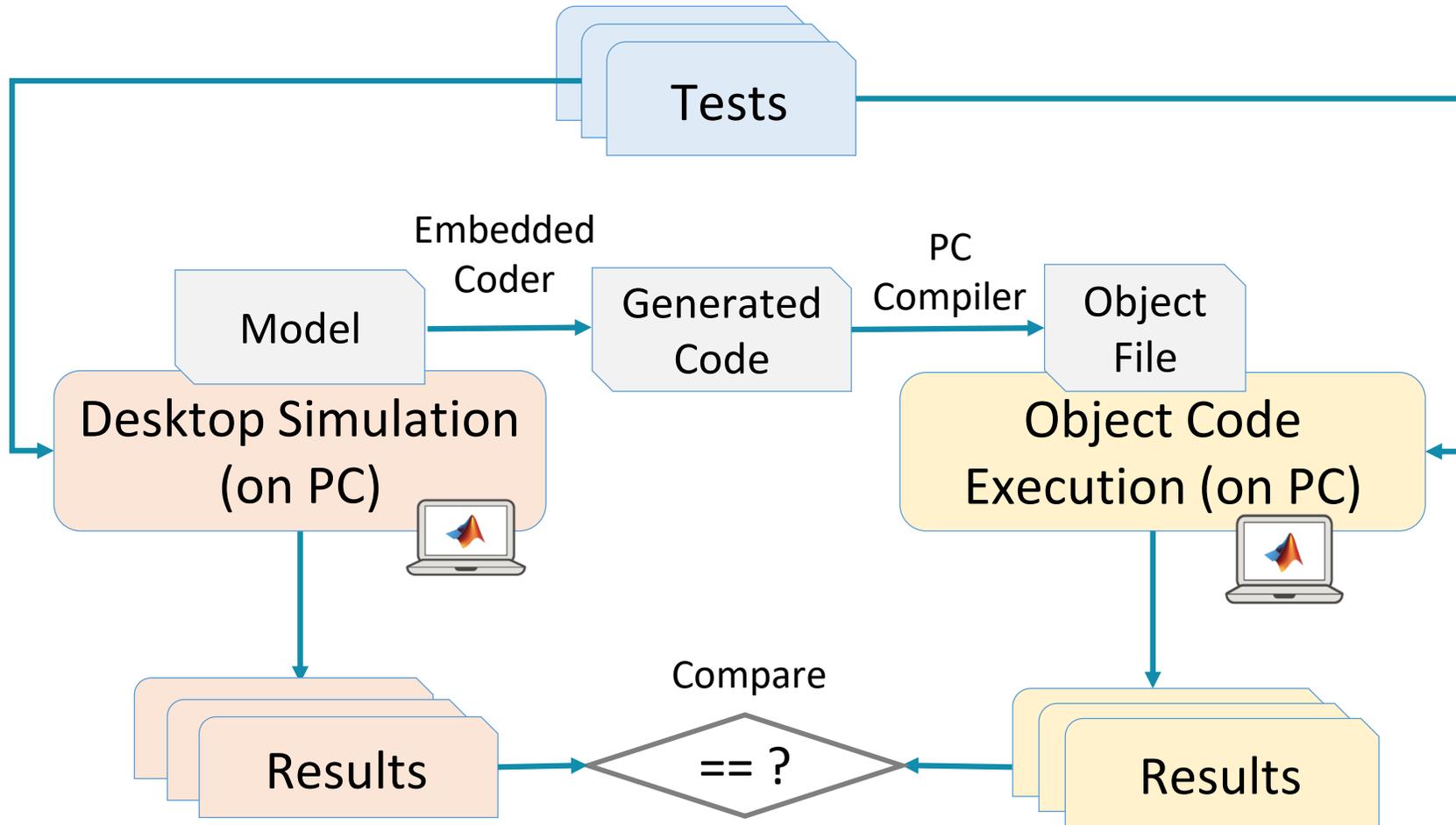
# Code Verification: Gain Confidence in the Generated Code

## Code Verification

- Trace code to model and requirements
- Measure code coverage
- SIL/PIL equivalence testing
- Generate 100% coverage test vectors

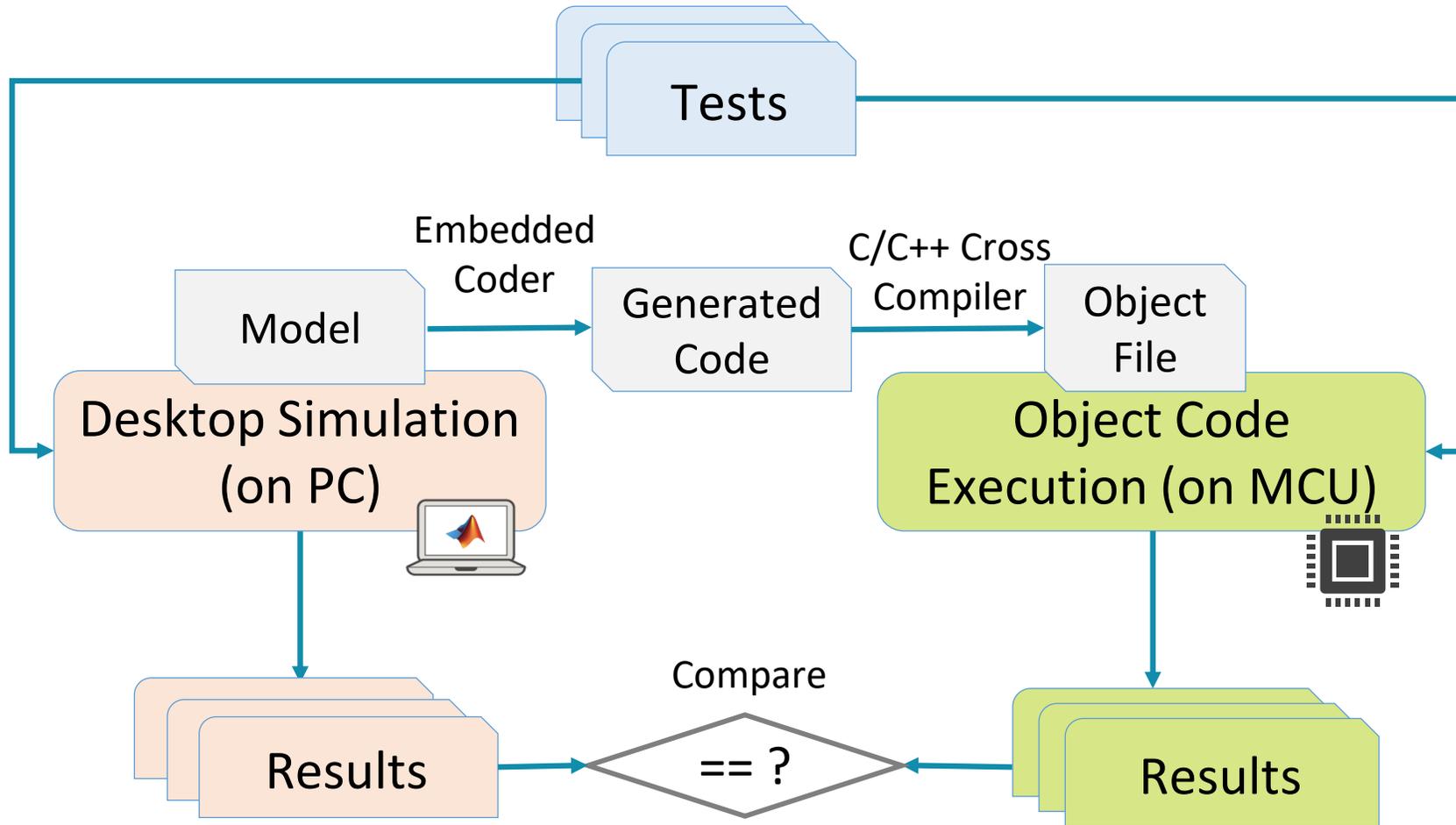


# Back-to-Back Testing: Simulation-In-the-Loop



- Automate SIL testing using Simulink Test
- Testing across releases

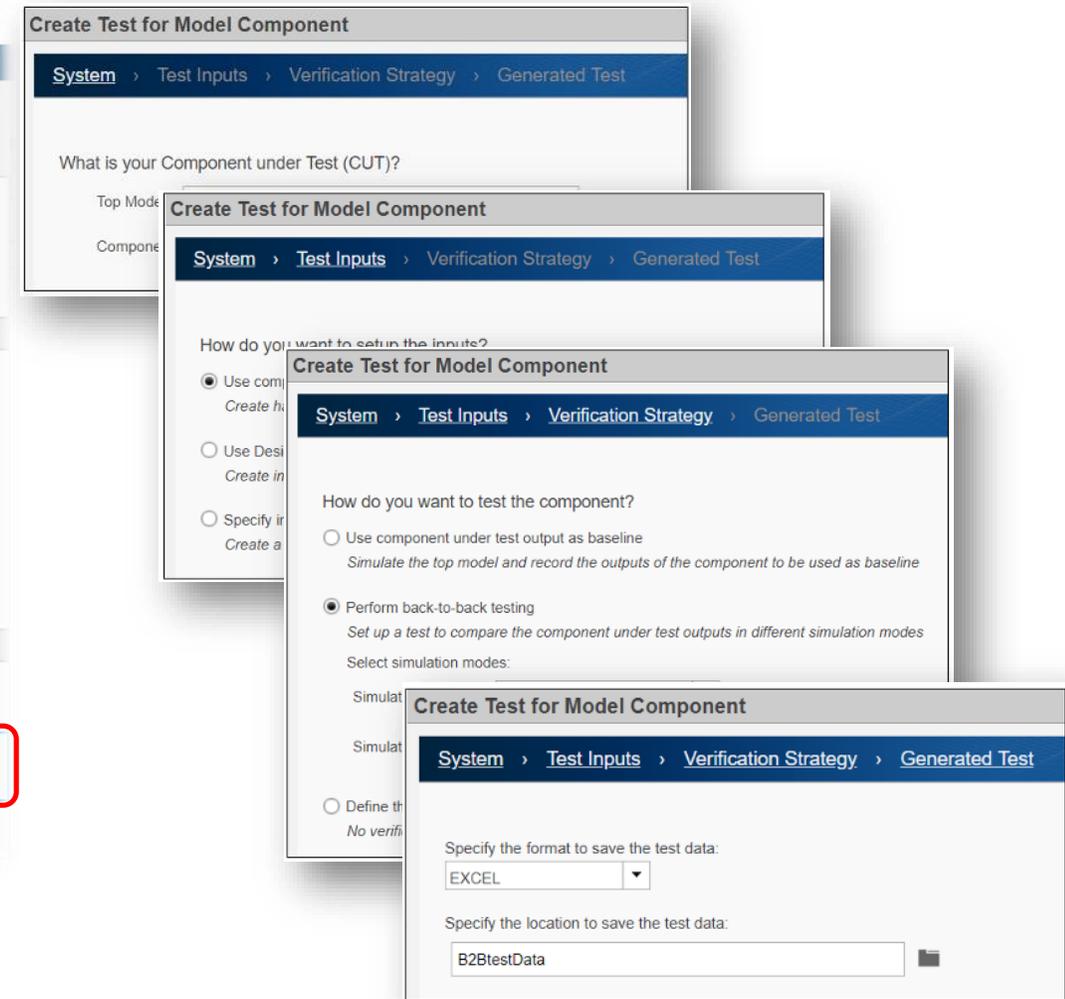
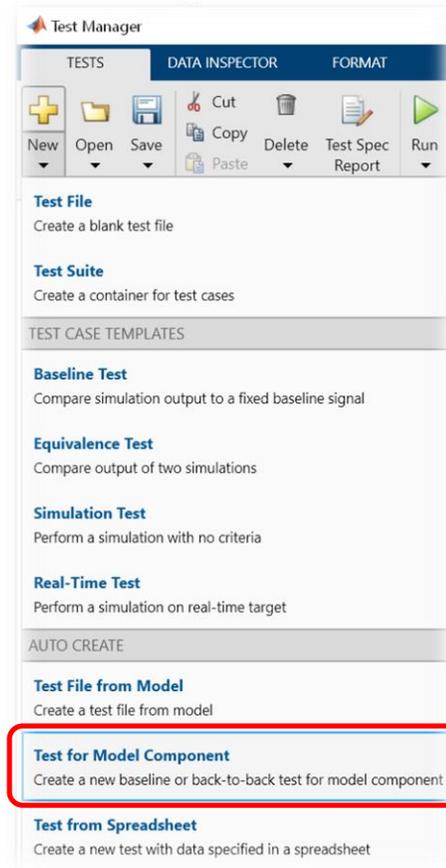
# Back-to-Back Testing: Processor-In-the-Loop



- Automate PIL testing using Simulink Test
- Testing across releases
- [Hardware support package](#)

# Automate Test Creation using Test Manager Wizard

- Guided steps to define component to test, inputs, type of test and format for output
- Wizard generates required test harness
- Auto generate tests using Simulink Design Verifier

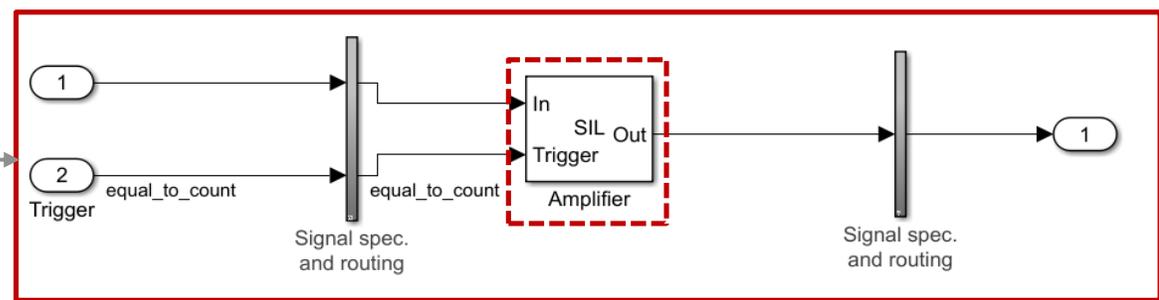
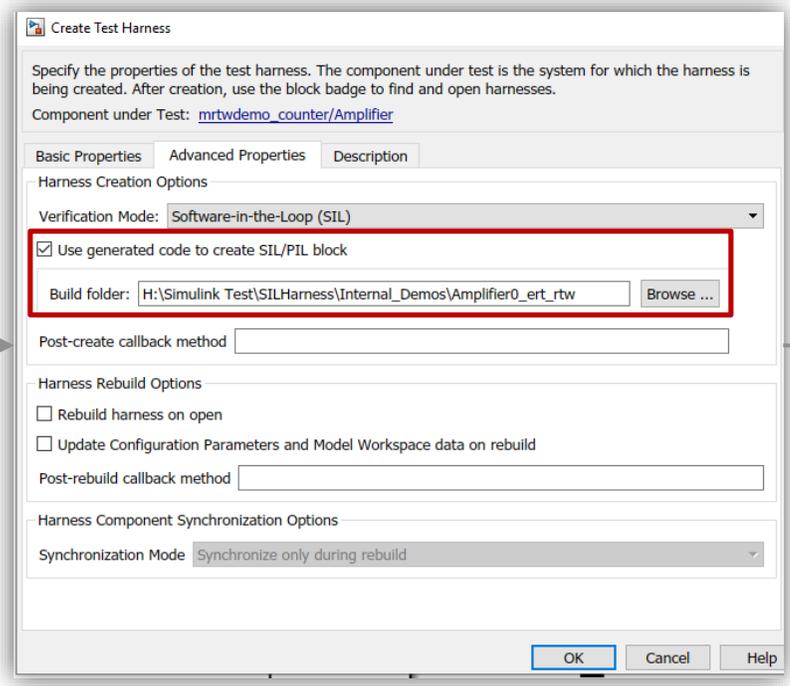
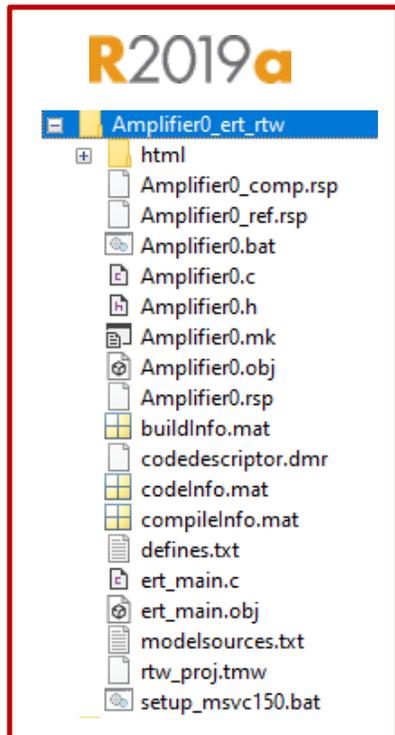


# Automate Test Creation using Test Manager Wizard

- Guided steps to define component to test, inputs, type of test and format for output
- Wizard generates required test harness
- Auto generate tests using Simulink Design Verifier

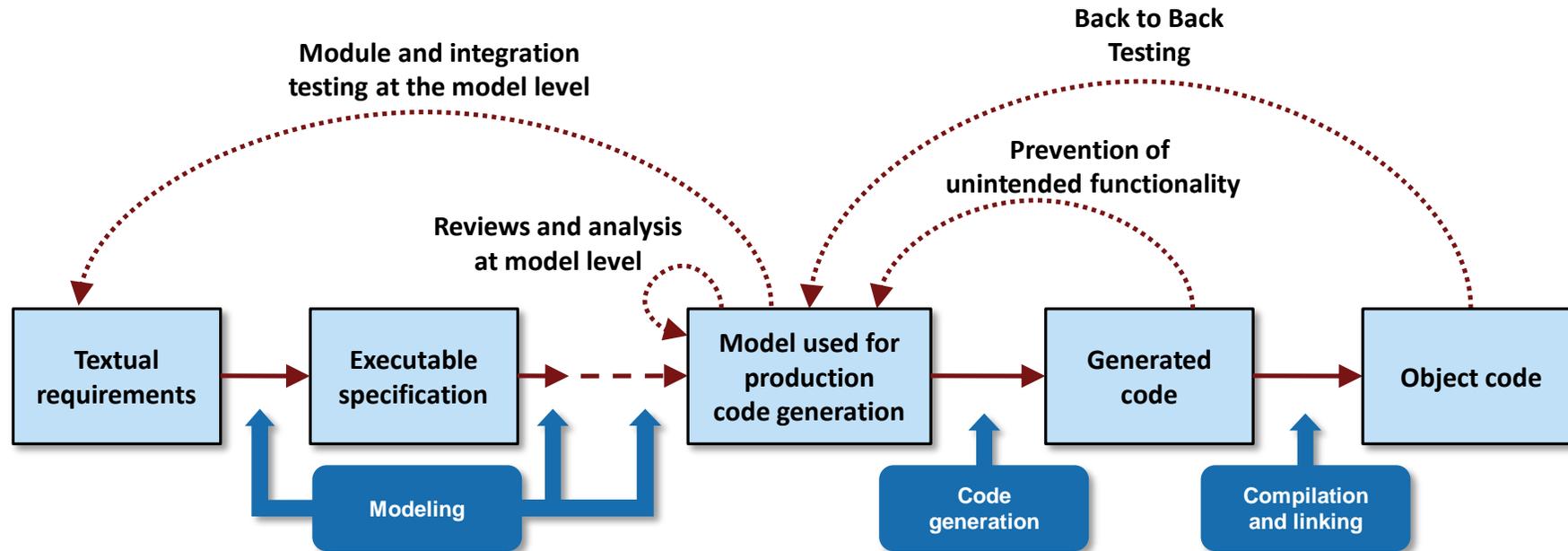
The screenshot displays the Test Manager application interface. On the left, the 'TEST CASE TEMPLATES' section lists several options: 'Test File', 'Test Suite', 'Baseline Test', 'Equivalence Test', 'Simulation Test', 'Real-Time Test', 'Test File from Model', 'Test for Model Component' (highlighted with a red box), and 'Test from Spreadsheet'. The main area shows the configuration for a test named 'rtwdemo\_sil\_block\_Harness1'. It is an 'Equivalence Test' for the subsystem 'rtwdemo\_sil\_block/Controller'. The configuration includes two simulation settings. Simulation 1 is set to 'Normal' mode, while Simulation 2 is set to 'Software-in-the-Loop (SIL)' mode. Both simulations use the 'rtwdemo\_sil\_block' model and a corresponding harness.

- Create a SIL/PIL test harness using code that was generated in a previous release
- Modify existing SIL/PIL test harnesses to store the build folder path information which can be used for rebuild



# Use Reference Workflow to Conform to Standards

- Shift verification earlier
- Automate manual verification tasks (coding, compiling, back-to-back)
- Measure completeness of Requirements-Based Testing



# Customer References and Applications



Airbus Helicopters Accelerates Development of DO-178B Certified Software with Model-Based Design

Software testing time cut by two-thirds



LS Automotive Reduces Development Time for Automotive Component Software with Model-Based Design

Specification errors detected early



Continental Develops Electronically Controlled Air Suspension for Heavy-Duty Trucks

Verification time cut by up to 50 percent

More User Stories: [https://kr.mathworks.com/company/user\\_stories.html](https://kr.mathworks.com/company/user_stories.html)

Verification, Validation 및 Test를 위한  
MATLAB 및 Simulink  
모델 기반 설계를 이용하여 임베디드 시스템을 검증합니다.

- [Verification, Validation, and Test Solution Page](#)
  - [Requirements-Based Testing Workflow Example](#)
  - [Verifying Models and Code for High-Integrity Systems](#)
  - [Getting Started with Model Verification and Validation](#)
- 요구사항을 아키텍처, 설계, 테스트, 코드까지 추적합니다. MathWorks 특은 엄격하고 자동화 가능한  
을 설계하고 고품질의 C, C++, HDL 코드를 생성합니다. MathWorks 특은 엄격하고 자동화 가능한  
을 설계하고 고품질의 C, C++, HDL 코드를 생성합니다. MathWorks 특은 엄격하고 자동화 가능한  
을 설계하고 고품질의 C, C++, HDL 코드를 생성합니다. MathWorks 특은 엄격하고 자동화 가능한
- "수작업으로 코딩하던 과거의 경험과 비교하면, 저희는 모델 기반 설계를 통해 인건비 30%, 테스트 코드 20%를 줄이고 생산성을 30% 이상 높였습니다. 저희는 자체 소프트웨어 개발 팀을 구축하는 한편, 일부 MCU 개발을 완료했습니다."
- Daming Li, Weichai Power
- 요구사항을 아키텍처, 설계, 테스트, 코드까지 추적합니다.
  - 모델과 코드의 표준 적합성을 점검하고 품질을 측정합니다.
  - 자동으로 테스트 케이스를 생성하여 테스트 커버리지를 확대합니다.
  - 보고서와 아티팩트를 생성하고 표준 적합성을 인증합니다(DO-178 및 ISO 26262 등).

**Thank You !!**