MATLAB EXPO

Accelerating Production of Industry-Compliant Embedded Software Using Model-Based Design

Vamshi Kumbham, MathWorks *Vaishnavi Hanumapalli Rajasimha, MathWorks*

MathWorks®

The Economist

Log in | Register | Subscribe | Digital & mobile

World politics | Business & finance | Economics | Science & technology | Culture | Blogs | De

When code can kill or cure

Medical technology: Applying the “open source” model to the design of medical devices promises to increase safety and spur innovation

Jun 2nd 2012 | From the print edition

Like 328 Tweet 236

Andrew Baker

Recall: BMW 7-Series may roll away when parked

Automaker blames a software problem that causes certain 2005-2008 models to remain in neutral.

By Clifford Atiyeh Oct 29, 2012 6:07AM

Share 83 Tweet 6 Share 14

BMW is again recalling the previous-generation 7-Series for a software problem, this time to stop the transmission from selecting neutral when the car is shut off, according to filings with the National Highway Traffic Safety Administration.

On 2005-2008 models with the Comfort Access keyless start option, the transmission may select neutral instead of park when the driver presses the start/stop button. Like other BMW models, the 7-Series is designed to engage park automatically upon shutoff, and the "P" button does not need to be pressed. However, several instances -- unknown to the driver -- can prevent this from occurring.

United Airlines experiences yet another major computer glitch

Problem with dispatch system software leads to hundreds of delays, some cancellations, call for 'heads to roll'

November 15, 2012

By Gregory Karp, Chicago Tribune

United Airlines, just a week before the year's busiest travel period,

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

SEAS Computer Scientists Find Vulnerabilities in Cisco VoIP Phones

HYBRID VEHICLES

Toyota: Software to blame for Prius brake problems

Home » Technology » Tech News

THE GLOBE AND MAIL

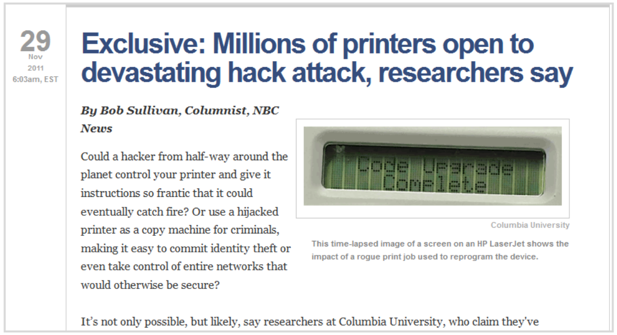
Hacker attack on your car’s computer could be lethal: experts

JIM FINKLE

Boston — Reuters

Published Monday, Aug. 20 2012, 8:41 AM EDT

Last updated Monday, Aug. 20 2012, 8:51 AM EDT



29 Nov 2011 6:03am, EST

Exclusive: Millions of printers open to devastating hack attack, researchers say

By Bob Sullivan, Columnist, NBC News

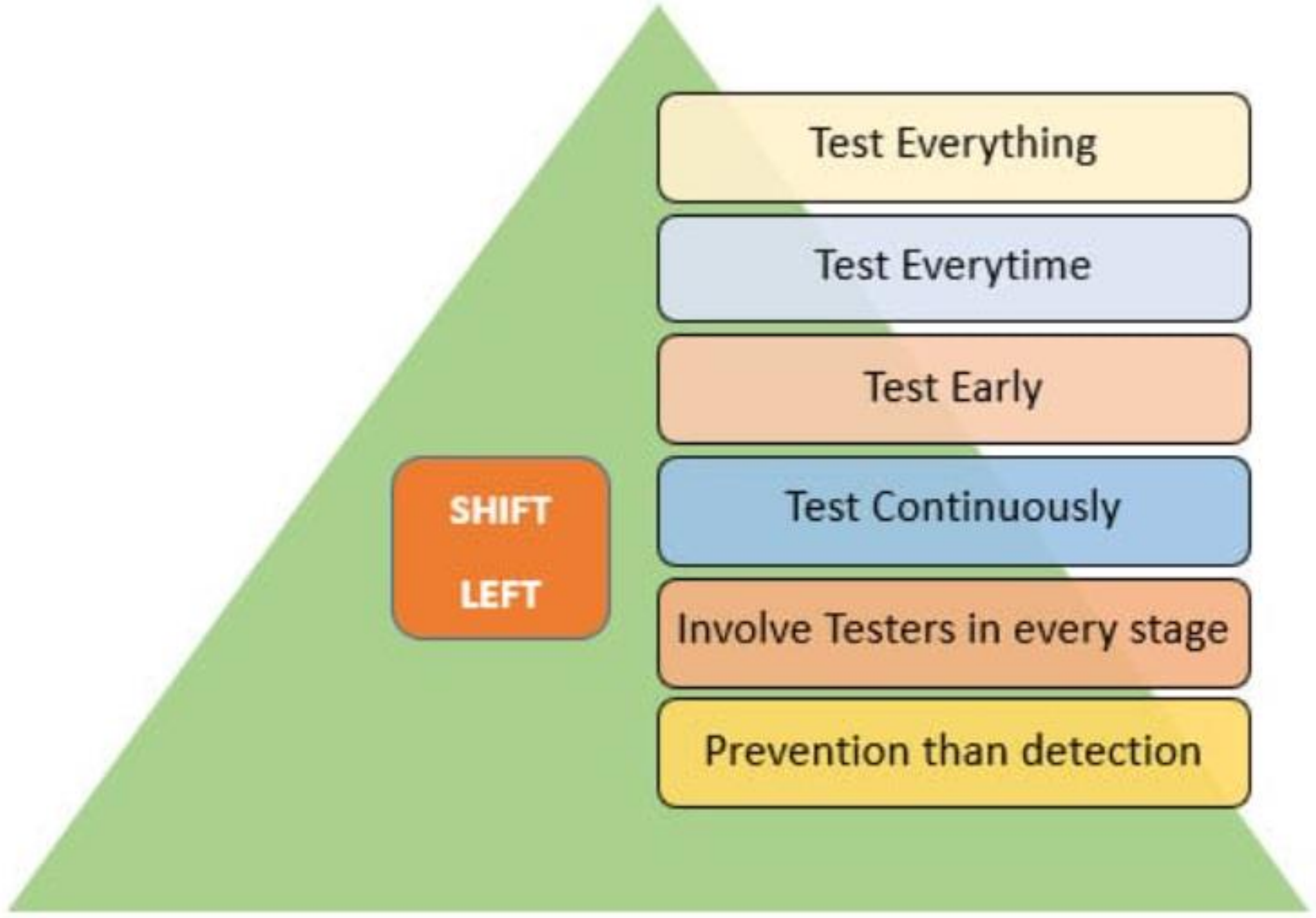
Could a hacker from half-way around the planet control your printer and give it instructions so frantic that it could eventually catch fire? Or use a hijacked printer as a copy machine for criminals, making it easy to commit identity theft or even take control of entire networks that would otherwise be secure?

Columbia University

This time-lapsed image of a screen on an HP LaserJet shows the impact of a rogue print job used to reprogram the device.

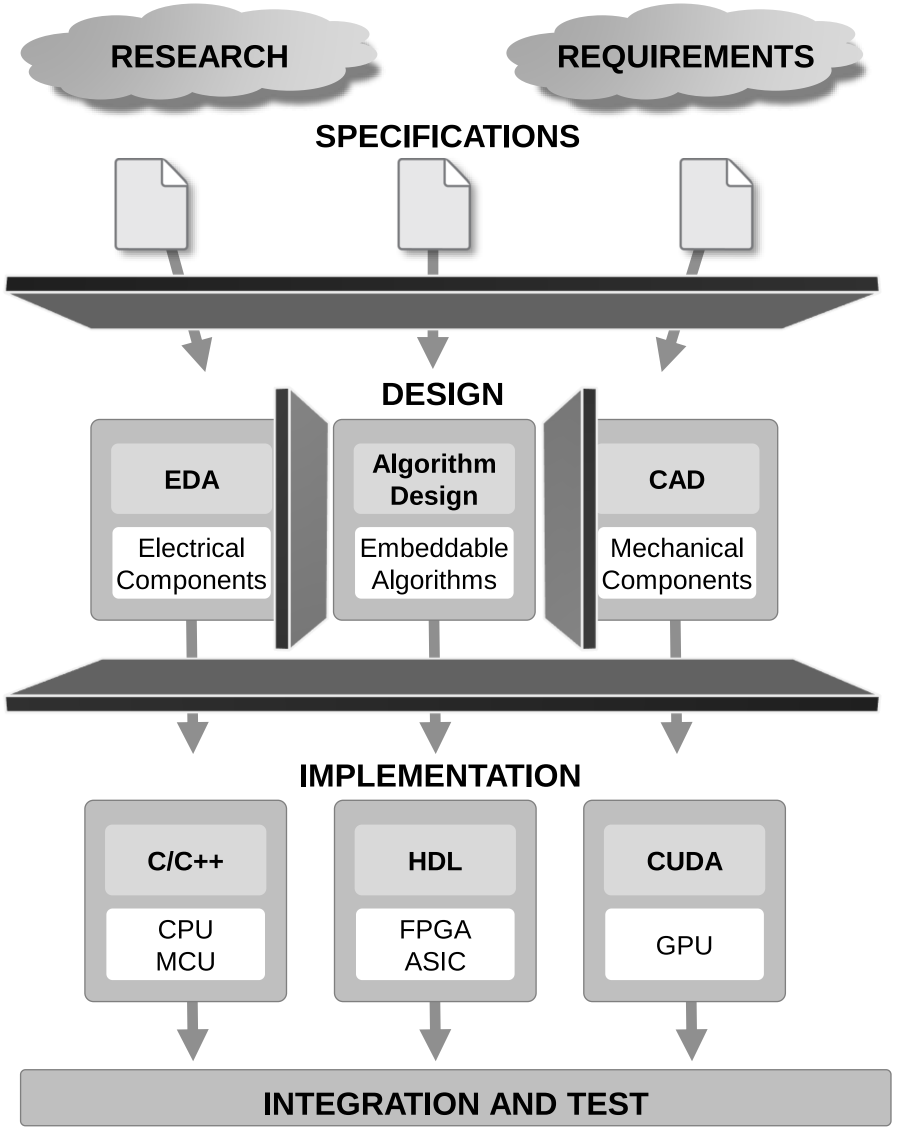
It’s not only possible, but likely, say researchers at Columbia University, who claim they've

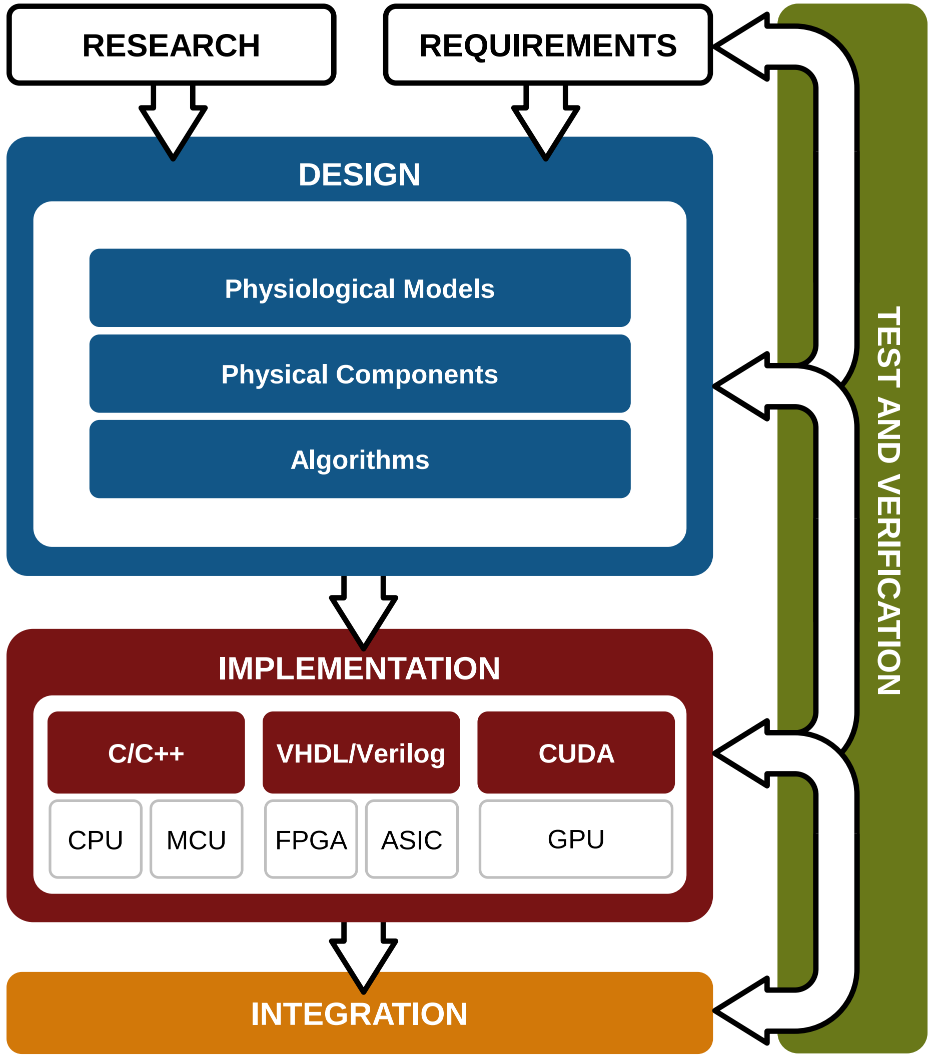
Shift-Left Verification



MATLAB EXPO

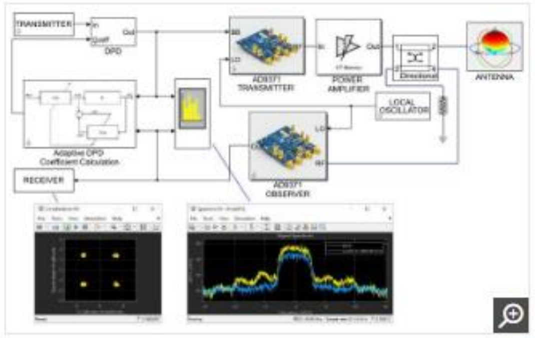
Traditional software development vs. Model-Based Design

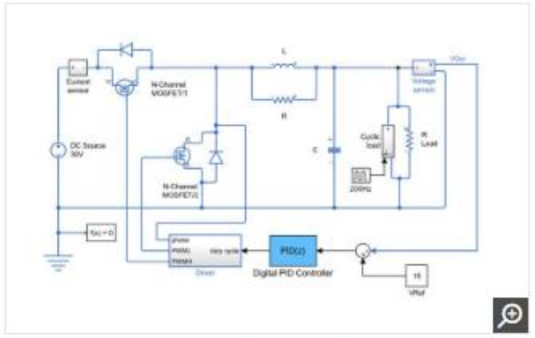


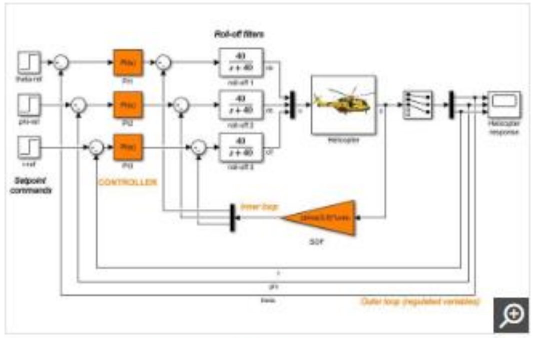


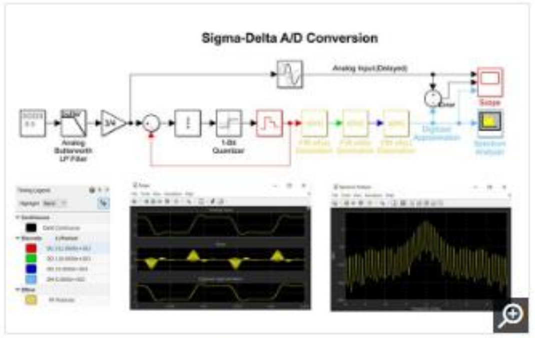
Source: Simulation and Model-Based Design

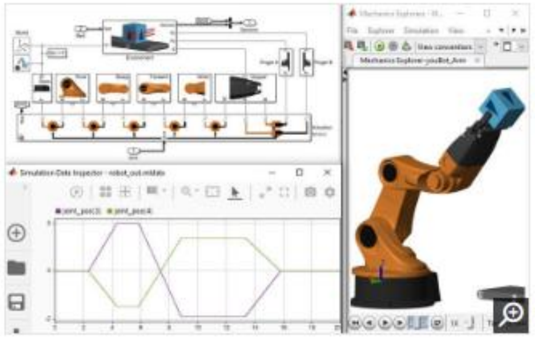
Simulink is for Simulation of Every Project:

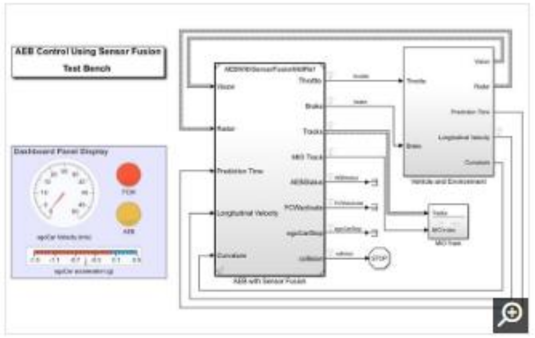

Wireless Communications

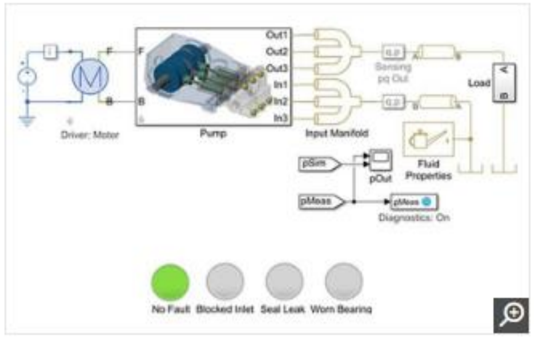

Electrification

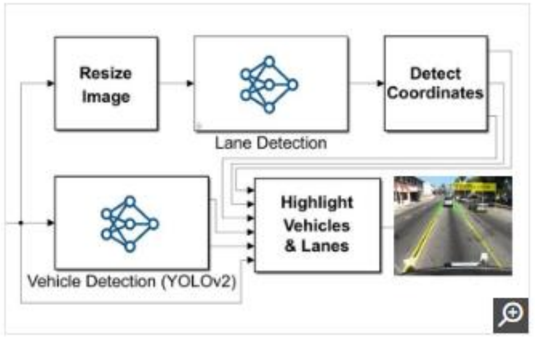

Control Systems


Signal Processing


Autonomous Systems and Robotics


Advanced Driver Assistance Systems


Digital Twins



Artificial Intelligence

Source: Simulation and Model-Based Design

Adoption of Model-Based Design across Industries

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

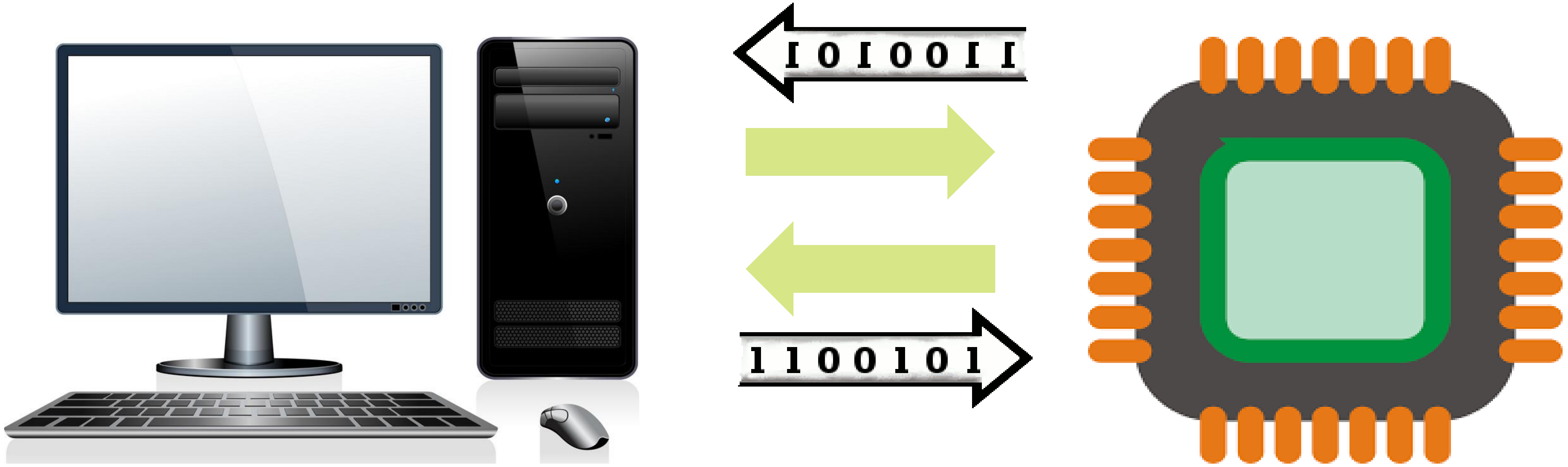
Specification errors detected early

EVLO Energy Storage Accelerates Development of Energy Management Systems with Model-Based Design

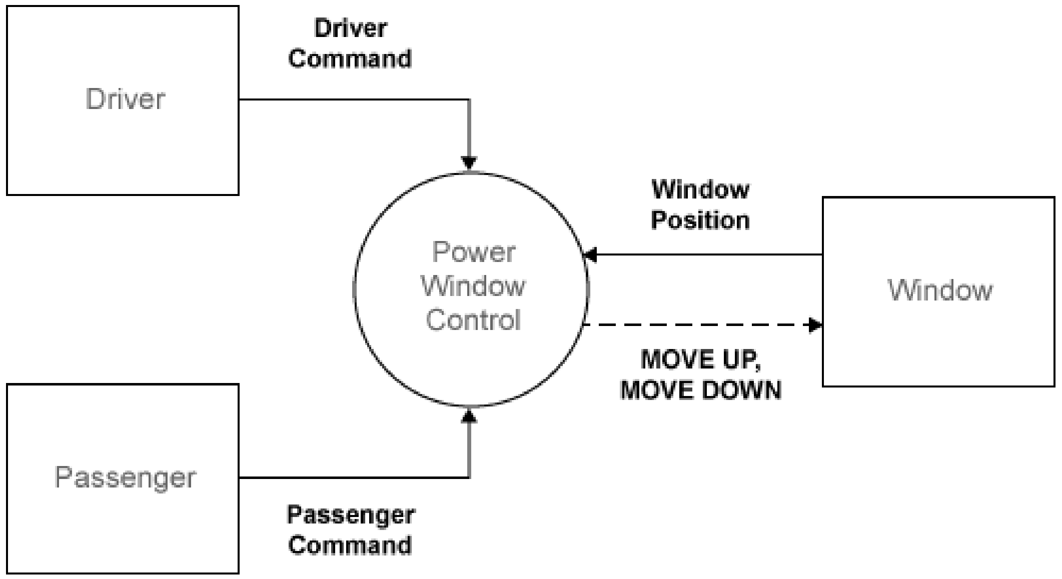
Continuously improve software quality

More User Stories: www.mathworks.com/company/user_stories.html

Concept to Deployment:



Developing Control Software for Power Window

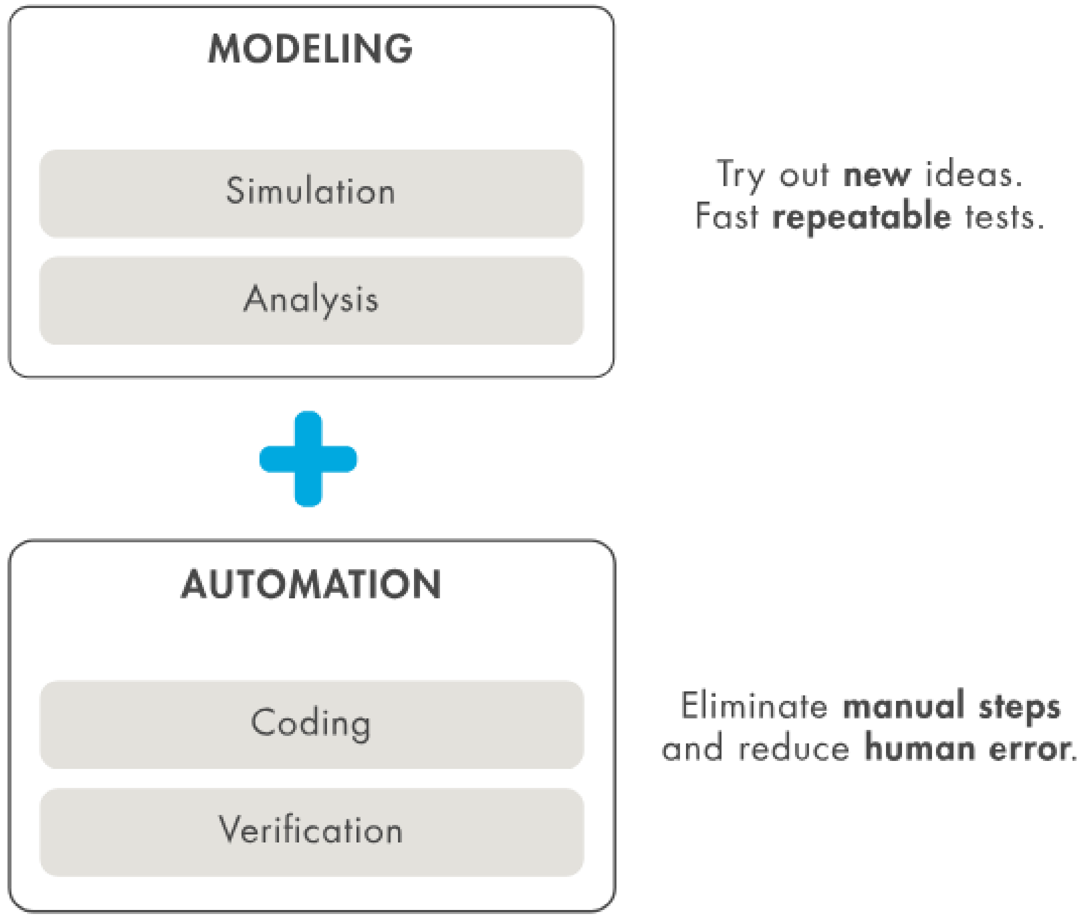


MATLAB EXPO

What we learn today

Accelerating Production of Embedded Software

- Simulate and test your system early and often
- Validate your design with physical models
- Generate and deploy directly to your embedded system
- Verify the generated code for any Run-Time issues and comply to Coding Standards
- Maintain a digital thread with traceability throughout


MODELING
Simulation
Analysis
Try out **new** ideas.
Fast **repeatable** tests.
AUTOMATION
Coding
Verification
Eliminate **manual steps**
and reduce **human error**.

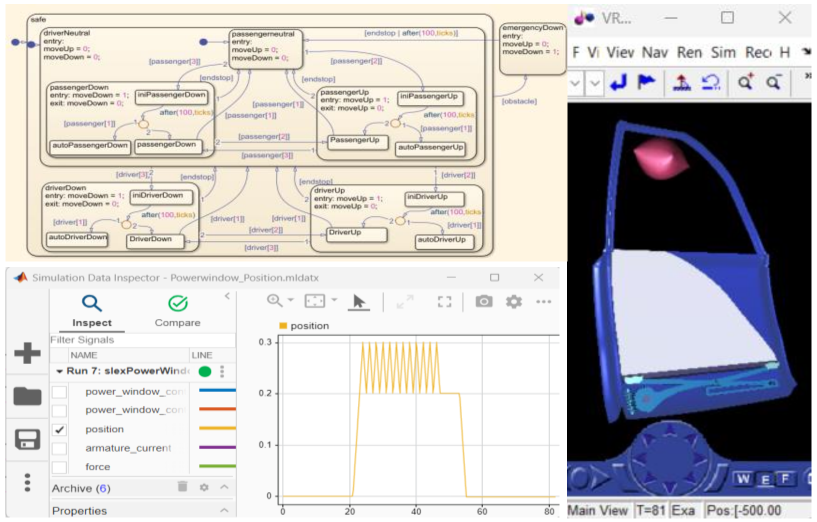
MATLAB EXPO

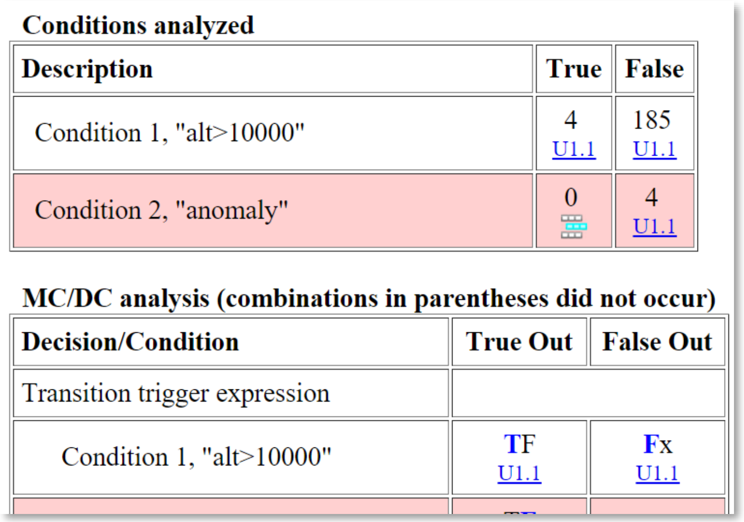
There are three key pieces to Model-Based Design

Modeling & Simulation

Testing & Validation

Code Generation & Code Verification

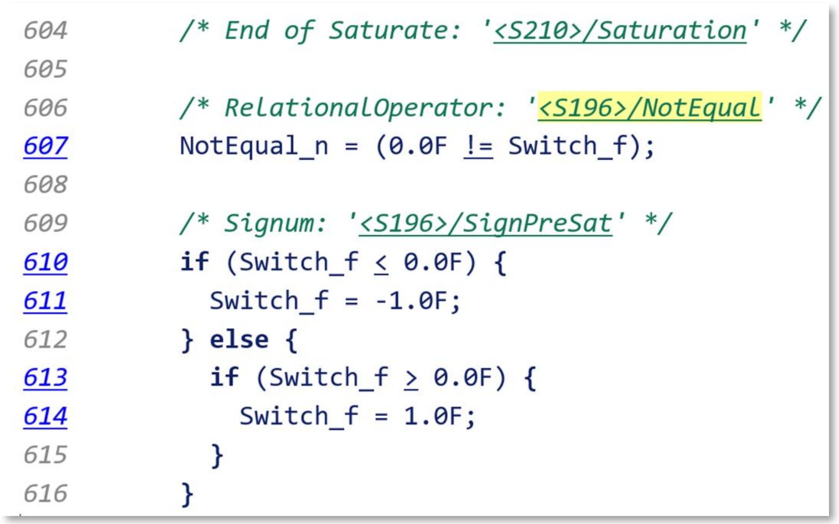



Conditions analyzed

Description	True	False
Condition 1, "alt>10000"	4 U1.1	185 U1.1
Condition 2, "anomaly"	0	4 U1.1

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition	True Out	False Out
Transition trigger expression		
Condition 1, "alt>10000"	TF U1.1	Fx U1.1

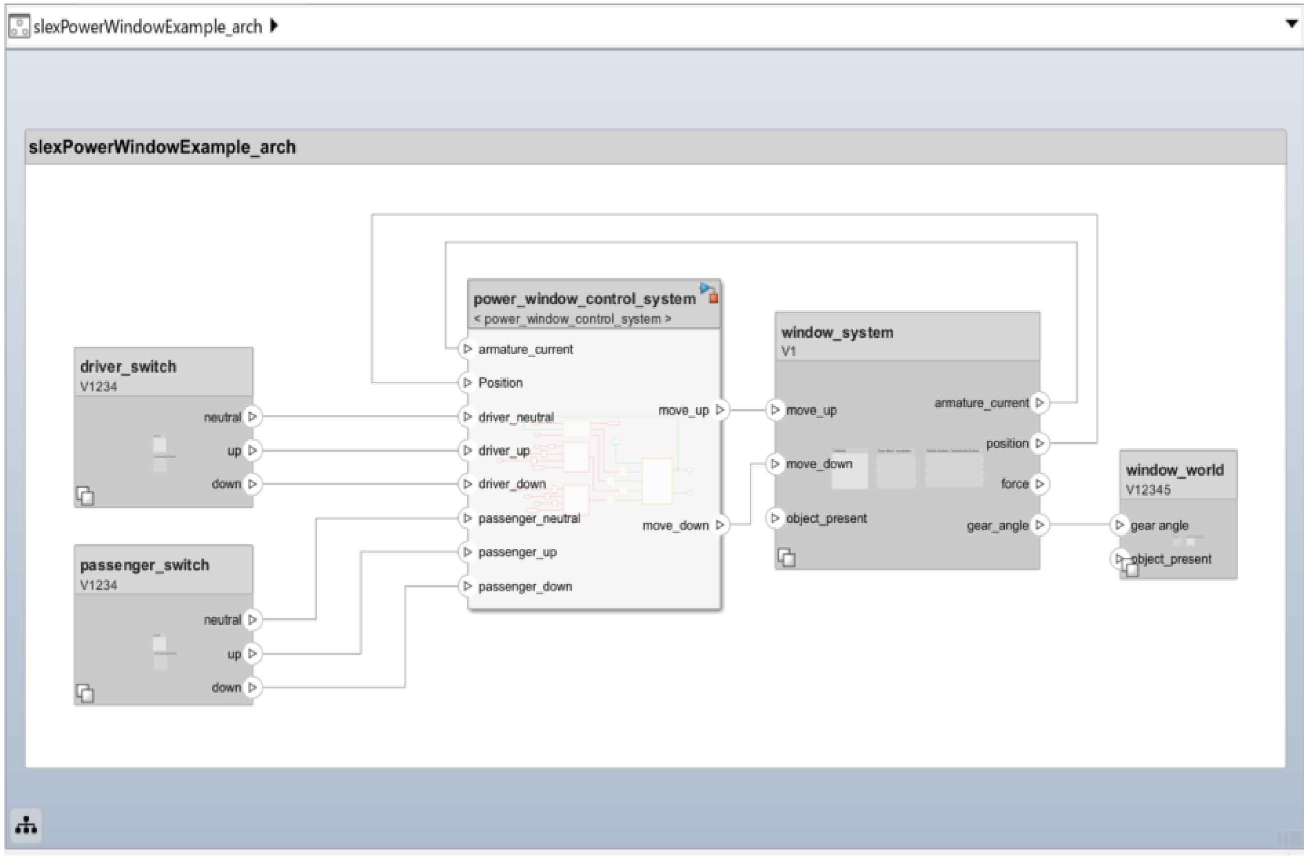

```
        /* End of Saturate: '<S210>/Saturation' */

        /* RelationalOperator: '<S196>/NotEqual' */
        NotEqual_n = (0.0F != Switch_f);

        /* Signum: '<S196>/SignPreSat' */
        if (Switch_f < 0.0F) {
          Switch_f = -1.0F;
        } else {
          if (Switch_f > 0.0F) {
            Switch_f = 1.0F;
          }
        }
```

SIMULINK®

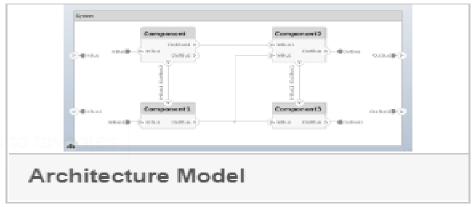
Simulation and Model-Based Design

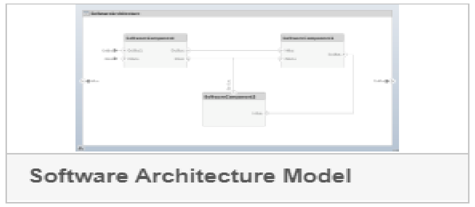
System Requirements:

- The window must fully open and fully close within 4 s.
- If the up is issued for between 200 ms and 1 s, the window must fully open. If the down command is issued for between 200 ms and 1 s, the window must fully close.
- The window must start moving 200 ms after the command is issued.
- The force to detect when an object is present is less than 100 N.
- When closing the window, if an object is in the way, stop closing the window and lower the window by approximately 10 cm

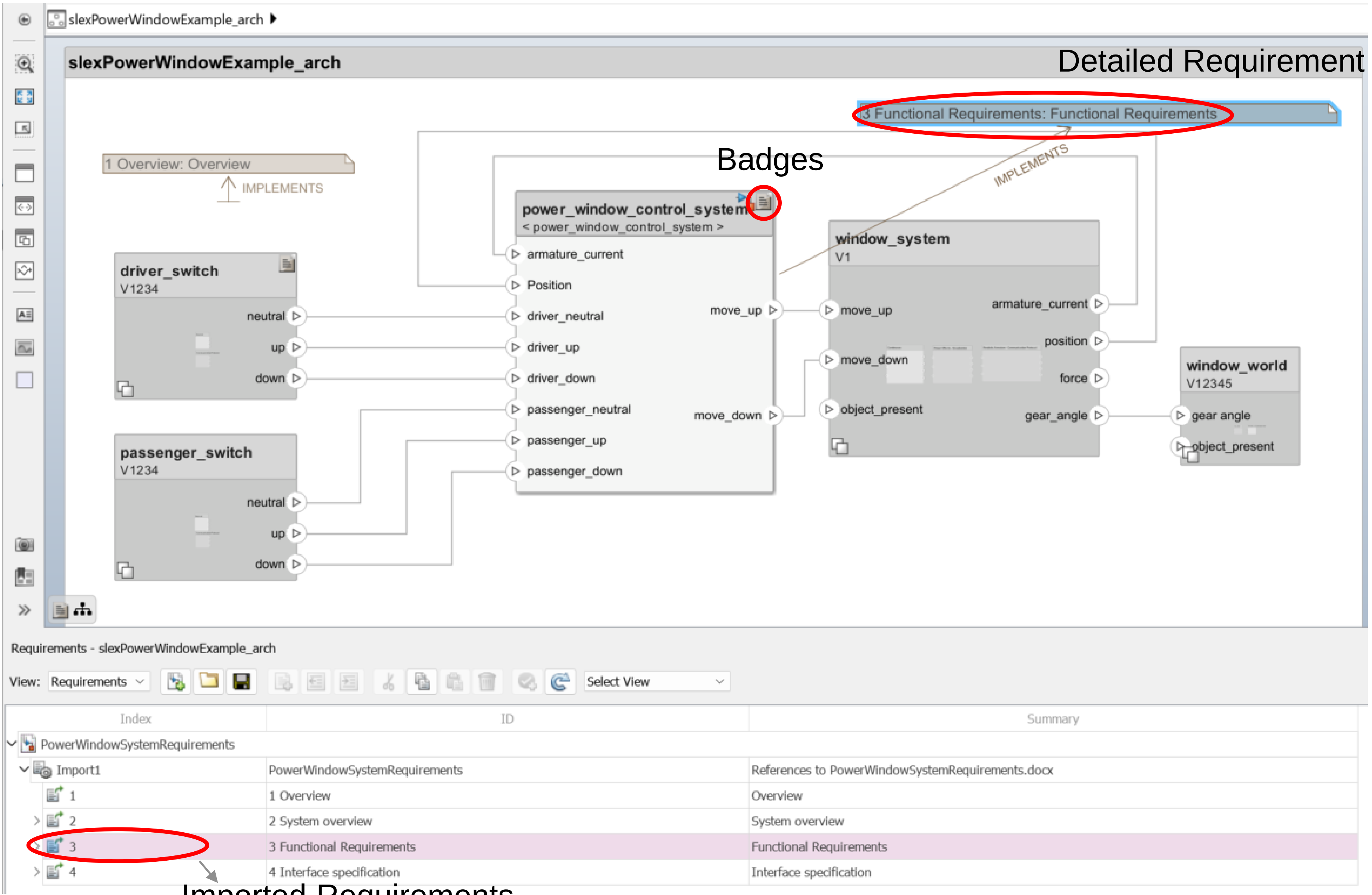


System Composer


Architecture Model

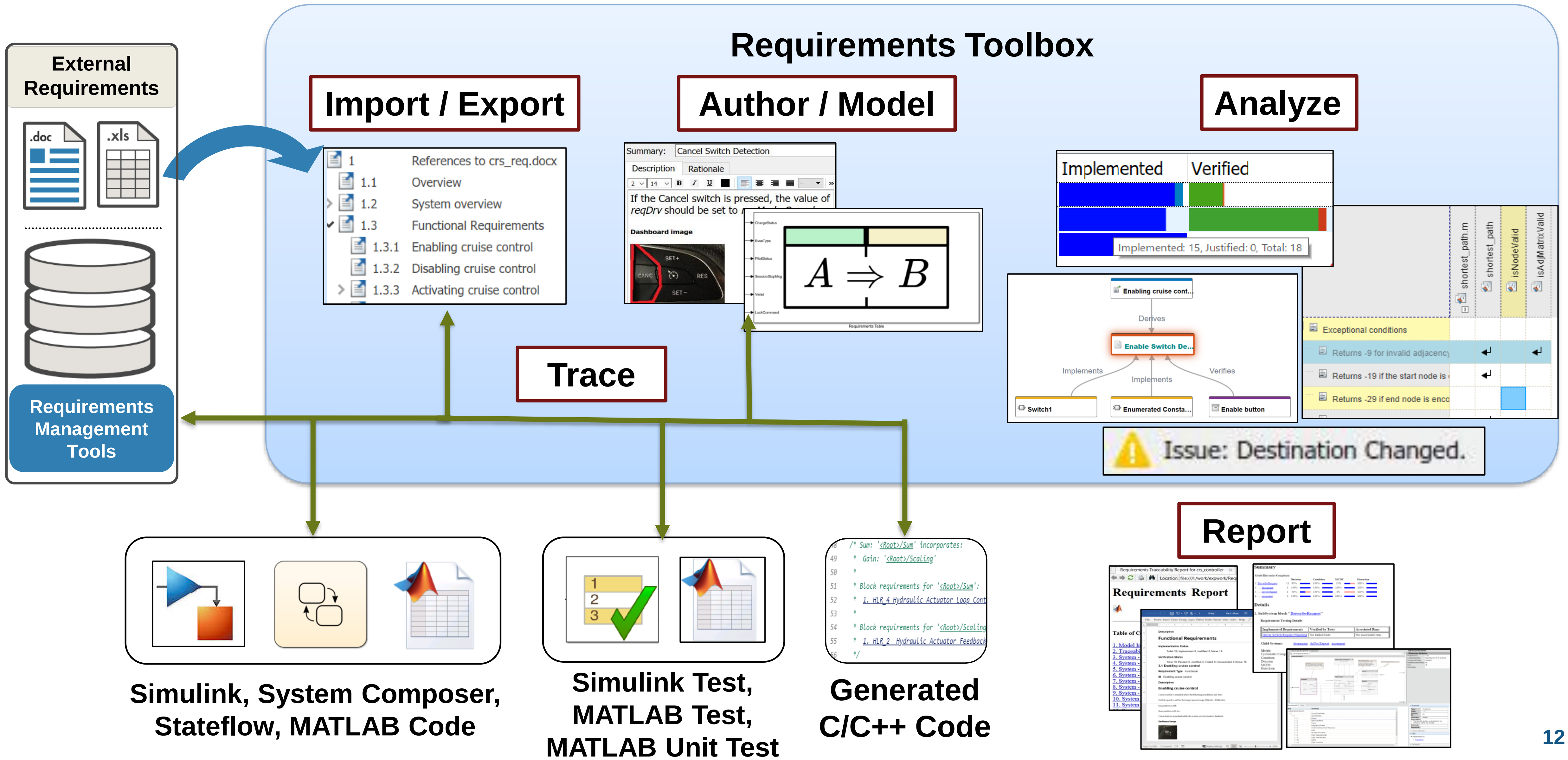

Software Architecture Model

Requirements and Model Linking:



Index	ID	Summary
PowerWindowSystemRequirements		
Import1	PowerWindowSystemRequirements	References to PowerWindowSystemRequirements.docx
1	1 Overview	Overview
2	2 System overview	System overview
3	3 Functional Requirements	Functional Requirements
4	4 Interface specification	Interface specification

Author, link, and validate for designs and tests



1		References to crs_req.docx
1.1		Overview
1.2		System overview
1.3		Functional Requirements
	1.3.1	Enabling cruise control
	1.3.2	Disabling cruise control
	1.3.3	Activating cruise control

Summary: Cancel Switch Detection

If the Cancel switch is pressed, the value of *reqDrv* should be set to

$$A \Rightarrow B$$

Implemented: 15, Justified: 0, Total: 18

Issue: Destination Changed.

```
/* Sum: '<Root>/Sum' incorporates:
 *  Gain: '<Root>/Scaling'
 *
 * Block requirements for '<Root>/Sum':
 *  1. HLR_4 Hydraulic Actuator Loop Cont
 *
 * Block requirements for '<Root>/Scaling
 *  1. HLR_2  Hydraulic Actuator Feedback
 */
```

Requirements Toolbox

Author, link, and validate requirements for designs and tests

Modeling&
Simulation

Import and Author Requirements

- Author requirements in MATLAB/Simulink
- Integrate with requirements tools

Model Requirements

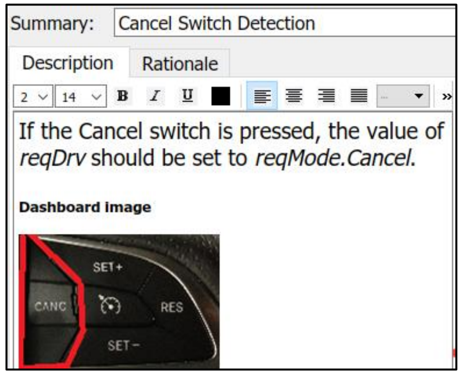
- Specify formal requirements
- Validate earlier with simulation

Requirements Traceability

- Trace to design, code and test
- Understand the impact of changes to design and test

Coverage and Impact Analysis

- Identify gaps in design or test
- Respond to requirement changes

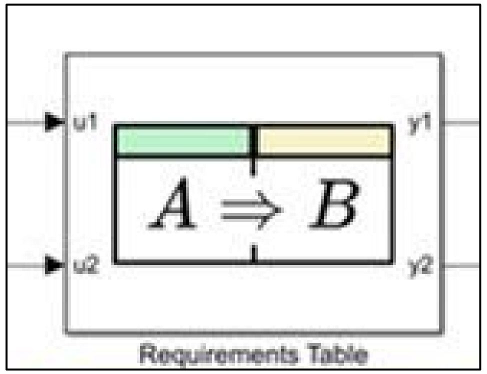

Summary: Cancel Switch Detection

Description | Rationale

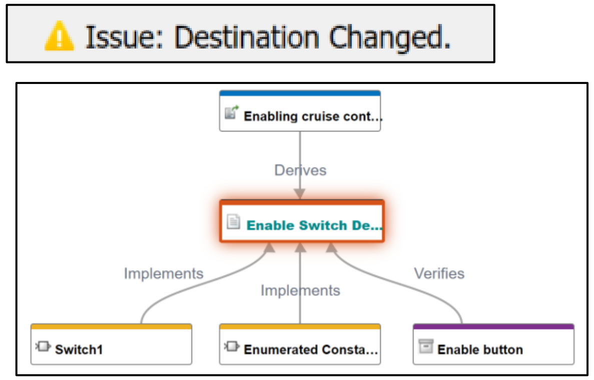
If the Cancel switch is pressed, the value of *reqDrv* should be set to *reqMode.Cancel*.

Dashboard image

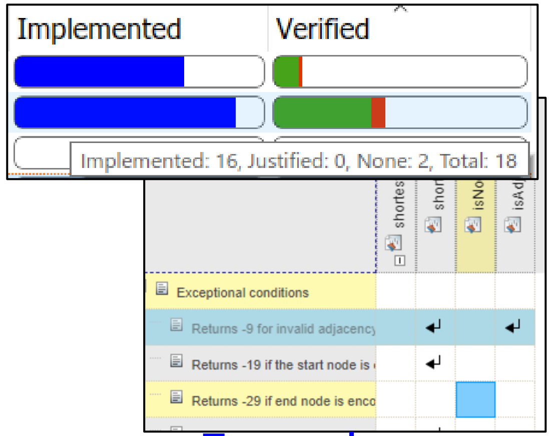
Examples



Examples


Issue: Destination Changed.

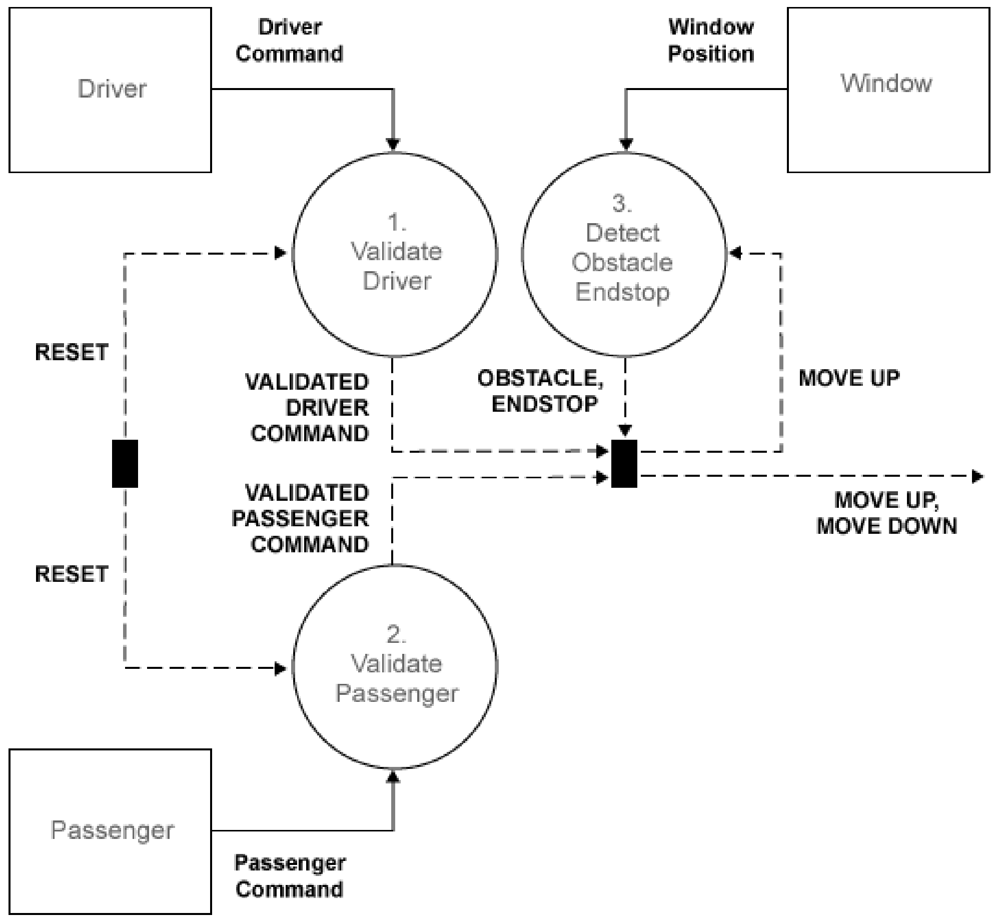
Examples

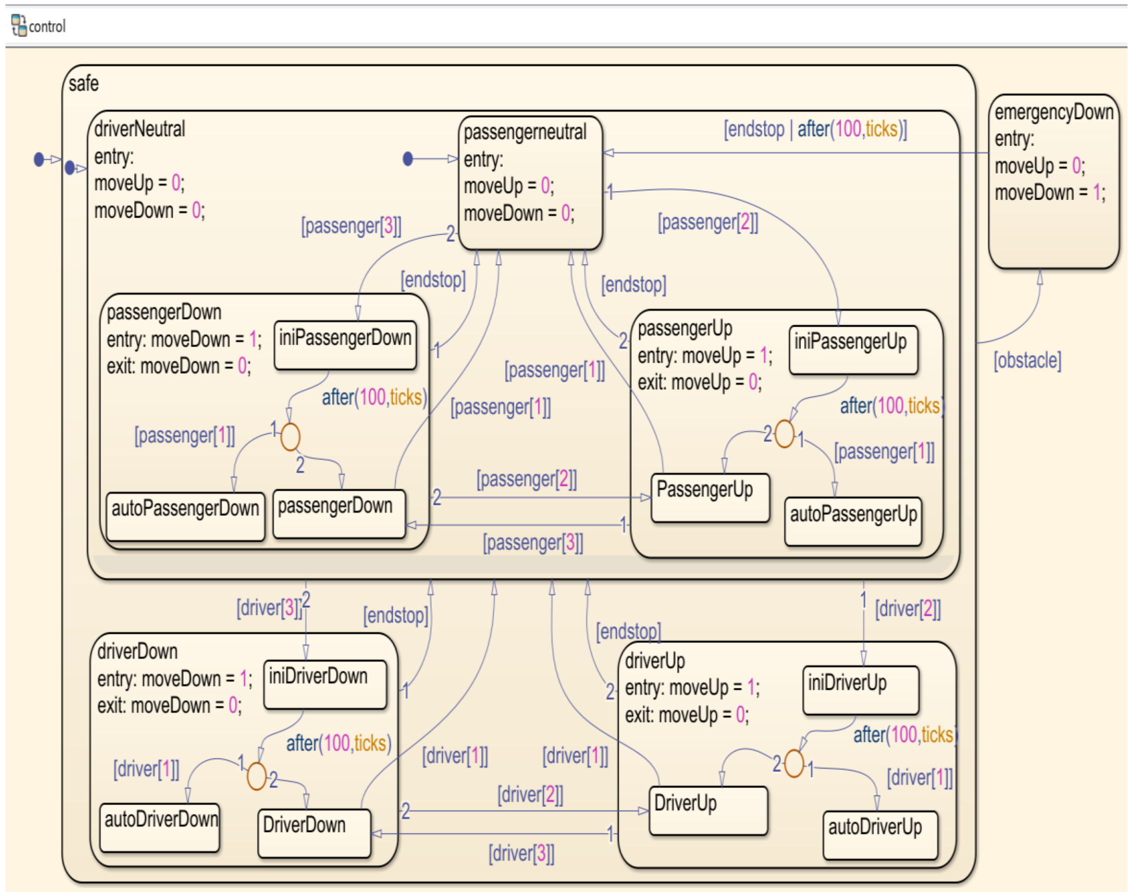

Implemented: 16, Justified: 0, None: 2, Total: 18

Implemented
Verified
shortes
shor
isNo
isAd
Exceptional conditions
Returns -9 for invalid adjacency
Returns -19 if the start node is
Returns -29 if end node is enco

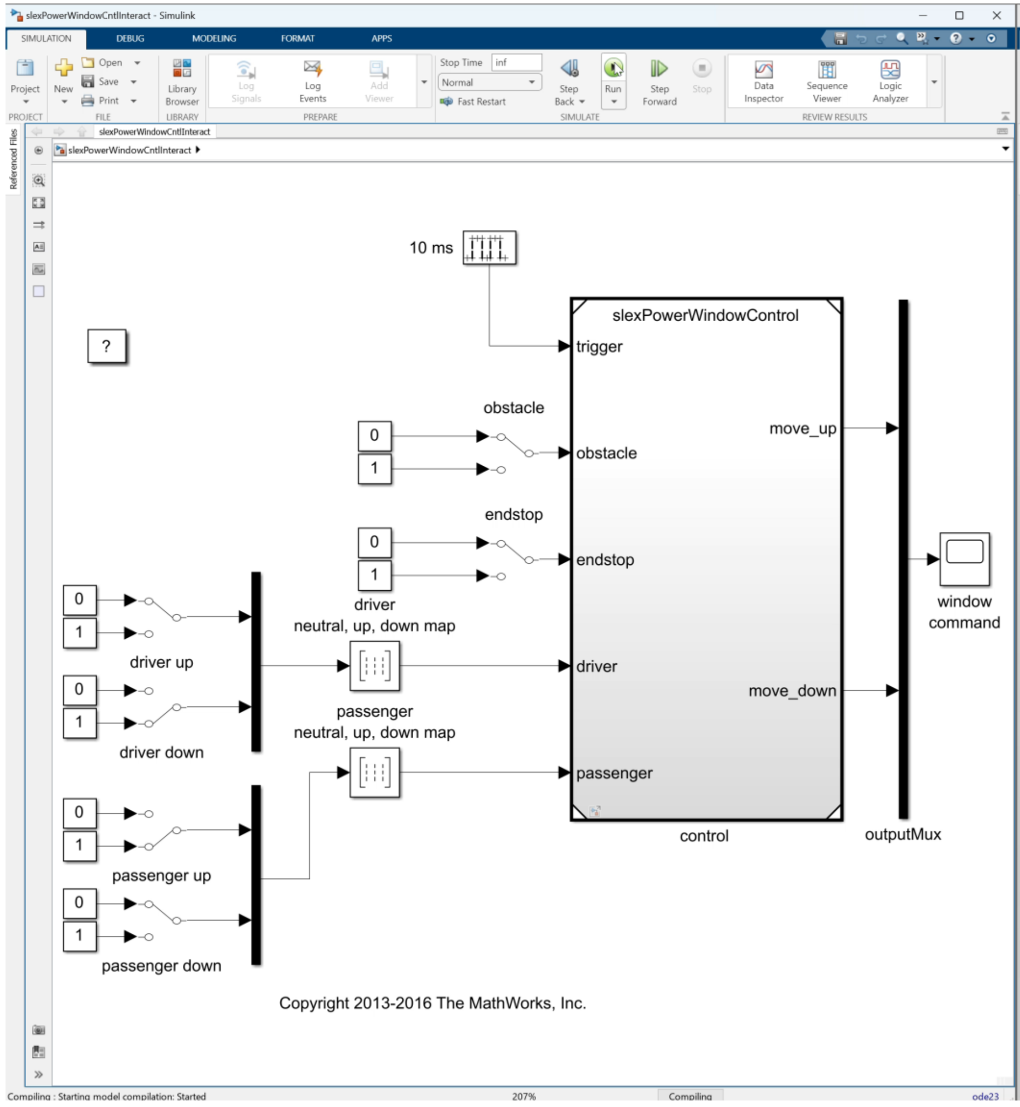
Examples

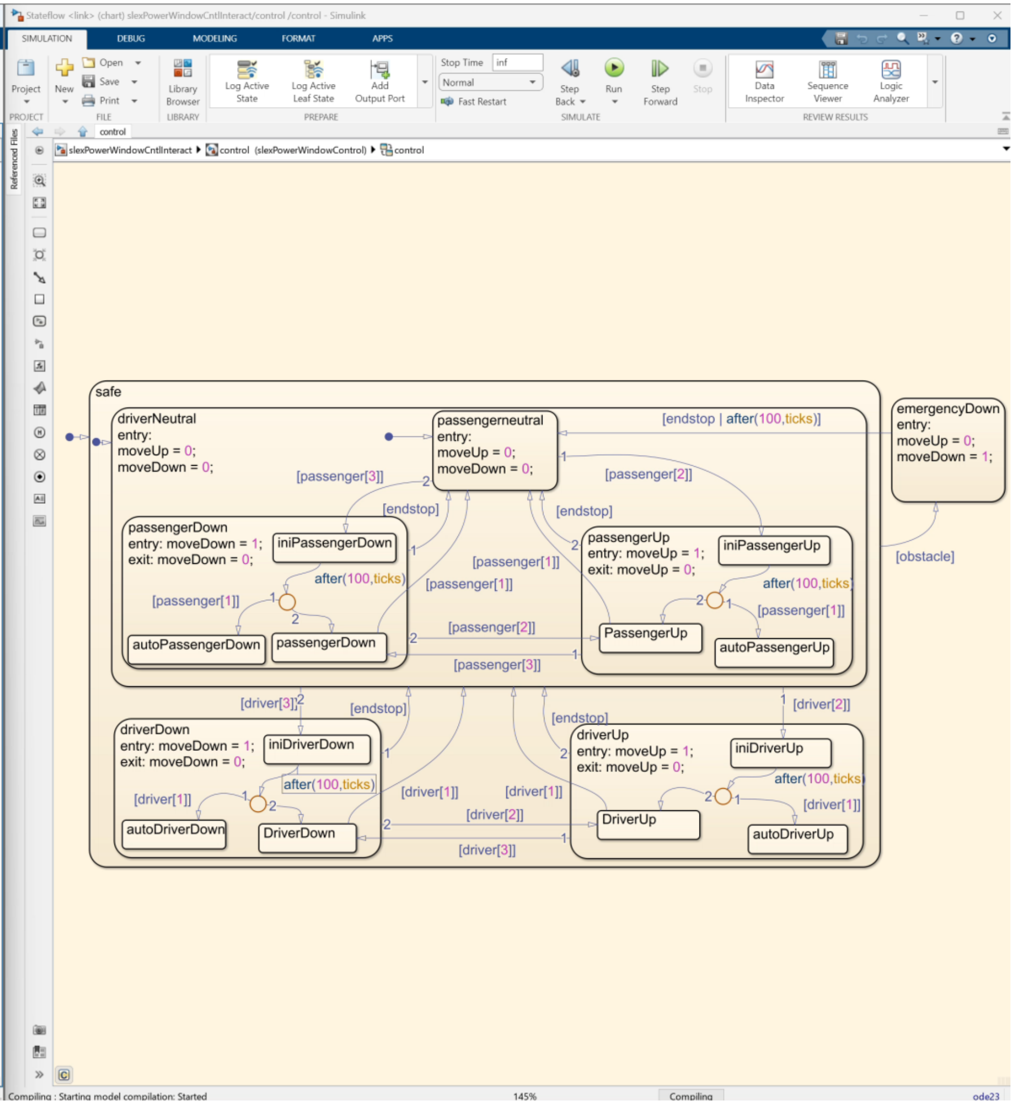
Modelling Software Requirements:



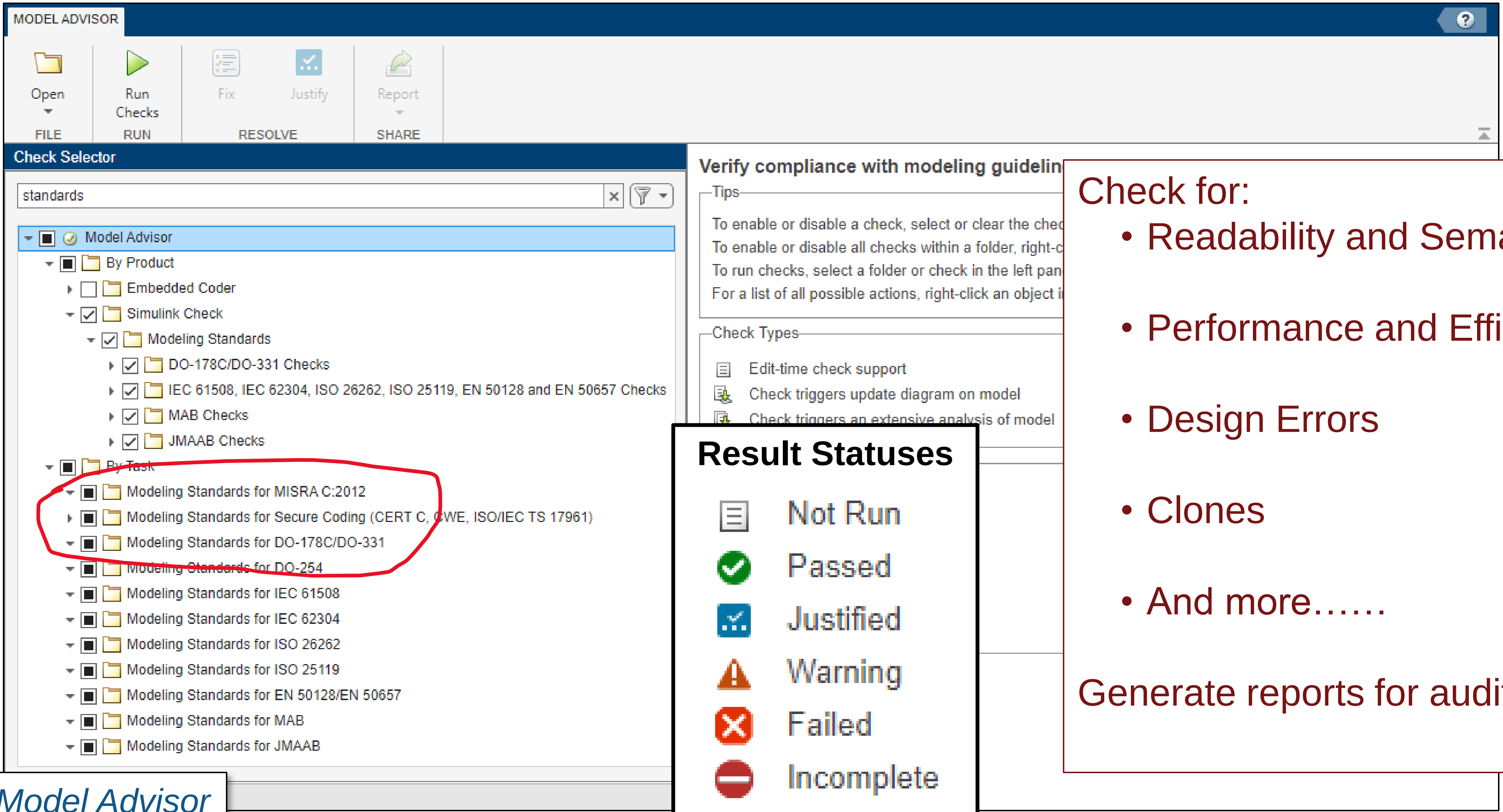


Create Interactive model for Simulation Analysis:





Run Model Advisor Checks:


MODEL ADVISOR

Check Selector

standards

- Model Advisor
  - By Product
    - Embedded Coder
    - ✓ Simulink Check
      - ✓ Modeling Standards
        - ✓ DO-178C/DO-331 Checks
        - ✓ IEC 61508, IEC 62304, ISO 26262, ISO 25119, EN 50128 and EN 50657 Checks
        - ✓ MAB Checks
        - ✓ JMAAB Checks
  - By Task
    - Modeling Standards for MISRA C:2012
    - Modeling Standards for Secure Coding (CERT C, CWE, ISO/IEC TS 17961)
    - Modeling Standards for DO-178C/DO-331
    - Modeling Standards for DO-254
    - Modeling Standards for IEC 61508
    - Modeling Standards for IEC 62304
    - Modeling Standards for ISO 26262
    - Modeling Standards for ISO 25119
    - Modeling Standards for EN 50128/EN 50657
    - Modeling Standards for MAB
    - Modeling Standards for JMAAB

Verify compliance with modeling guidelin

Tips

To enable or disable a check, select or clear the chec
To enable or disable all checks within a folder, right-c
To run checks, select a folder or check in the left pan
For a list of all possible actions, right-click an object i

Check Types

Edit-time check support
Check triggers update diagram on model
Check triggers an extensive analysis of model

Result Statuses

☰	Not Run
✓	Passed
✓	Justified
⚠	Warning
✗	Failed
⛔	Incomplete

Check for:

- Readability and Semantics
- Performance and Efficiency
- Design Errors
- Clones
- And more……

Generate reports for audits

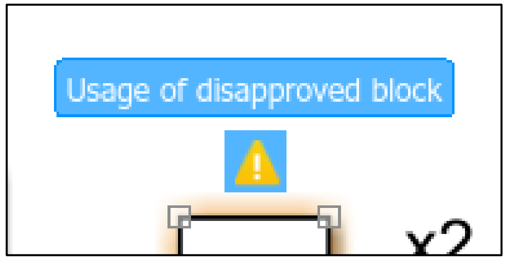
Model Advisor

Simulink Check

Automate verification and correct models to improve design

Standards & Guidelines Checks

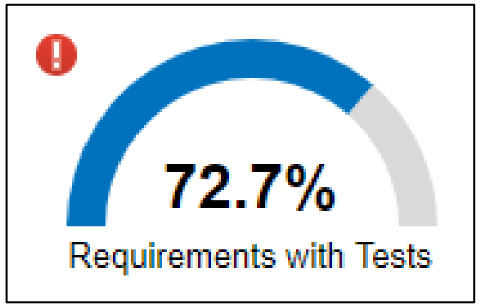
- Automate standards compliance
- Find and fix issues while you design
- Customize checks



Examples

Dashboards

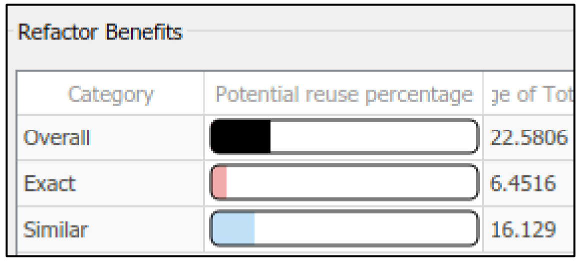
- Assess completeness and quality
- Analyze complexity, size, reusability



Examples

Model Refactoring

- Find clones and modeling patterns
- Refactor to improve maintainability



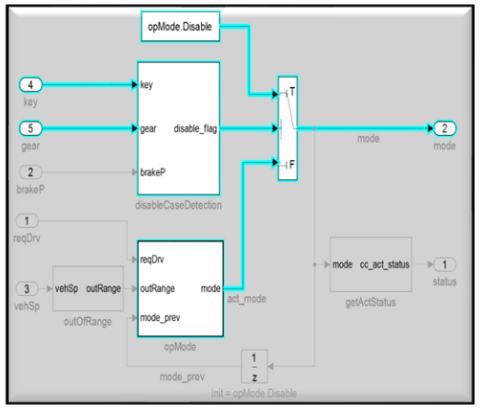
Refactor Benefits

Category	Potential reuse percentage	ge of Tot
Overall		22.5806
Exact		6.4516
Similar		16.129

Examples

Model Slicer

- Simplify models to isolate behavior
- Debug test failures



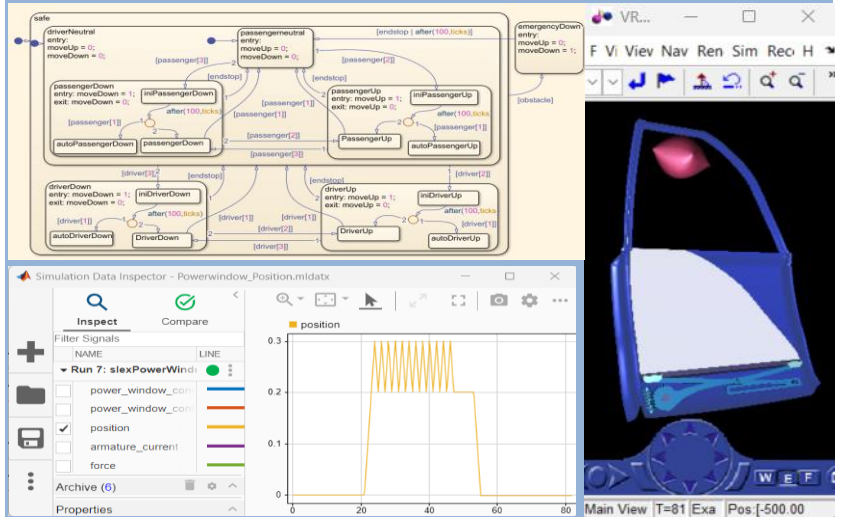
Examples

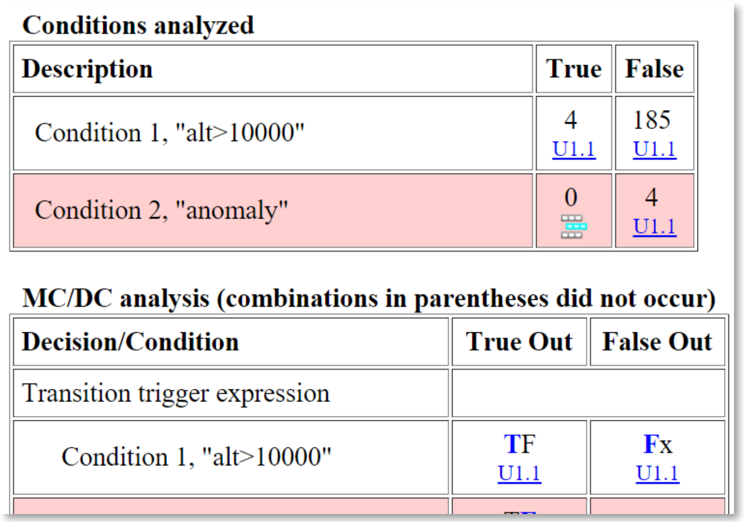
There are three key pieces to Model-Based Design

✓ Modeling & Simulation

Testing & Validation

Code Generation & Code Verification



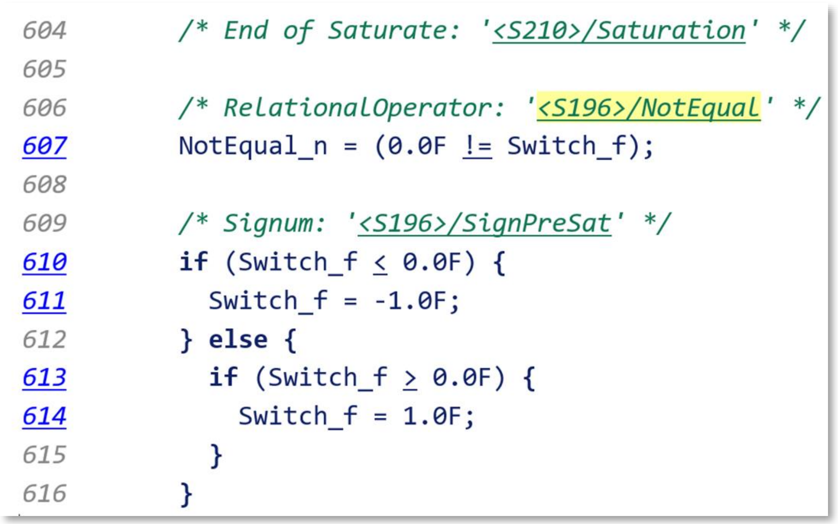


Conditions analyzed

Description	True	False
Condition 1, "alt>10000"	4 U1.1	185 U1.1
Condition 2, "anomaly"	0	4 U1.1

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition	True Out	False Out
Transition trigger expression		
Condition 1, "alt>10000"	TF U1.1	Fx U1.1



```
        /* End of Saturate: '<S210>/Saturation' */

        /* RelationalOperator: '<S196>/NotEqual' */
        NotEqual_n = (0.0F != Switch_f);

        /* Signum: '<S196>/SignPreSat' */
        if (Switch_f < 0.0F) {
          Switch_f = -1.0F;
        } else {
          if (Switch_f > 0.0F) {
            Switch_f = 1.0F;
          }
        }
```

SIMULINK®

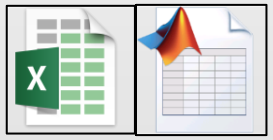
Simulation and Model-Based Design

Systematic Functional Testing with Simulink Test

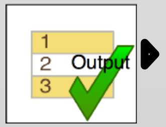
Test Case

Model Sim through SIL, PIL and HIL
Scale with Parallel Computing Toolbox and Continuous Integration

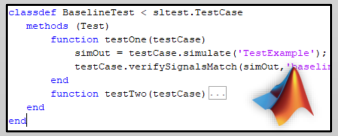
Inputs


Data file (input)

Signal Editor

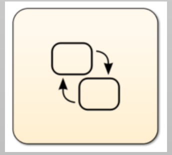


Test Sequence

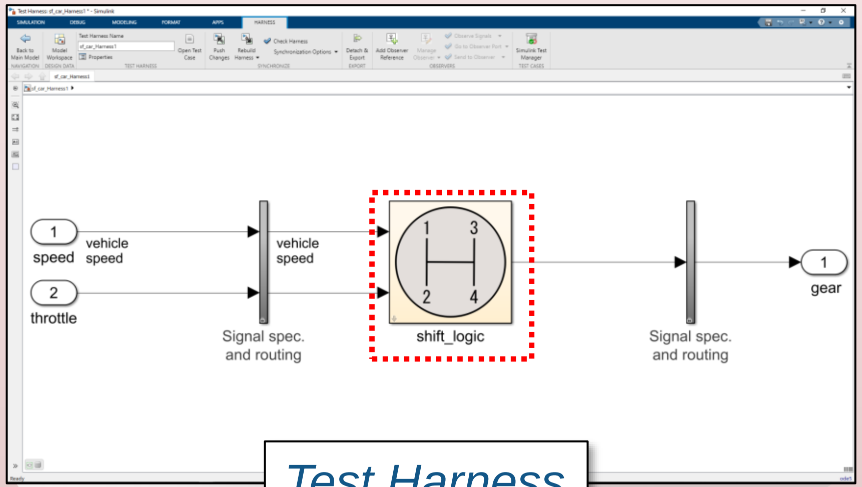


```
classdef BaselineTest < sltest.TestCase
    methods (Test)
        function testOne(testCase)
            simOut = testCase.simulate('TestExample');
            testCase.verifySignalsMatch(simOut,'baselin
        end
        function testTwo(testCase) ...
    end
end
```

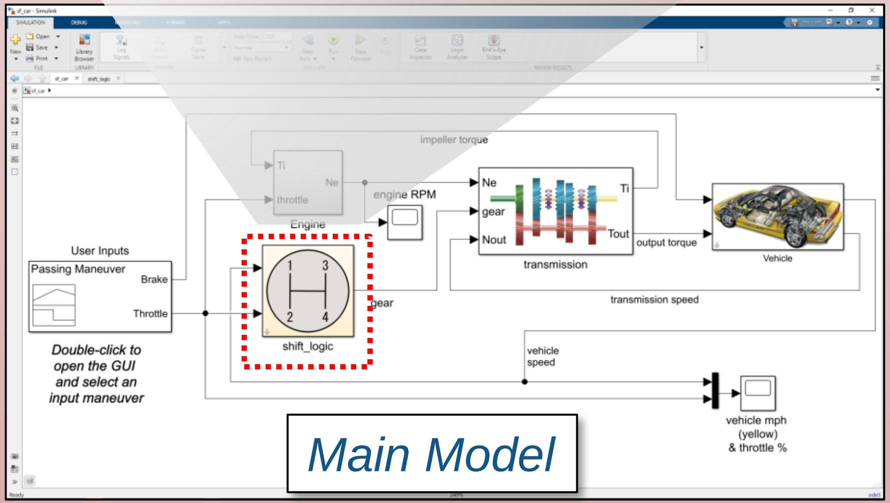
MATLAB Code


Stateflow

and more!

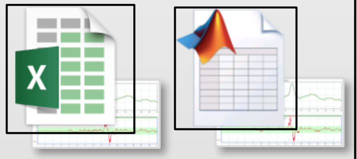


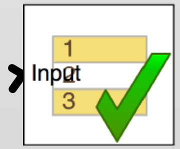
Test Harness



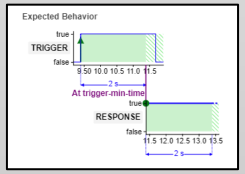
Main Model

Assessments

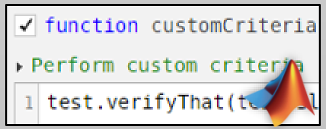

Data file baseline)



Test Assessment



Temporal Assessment



```
✓ function customCriteria
▸ Perform custom criteria
  test.verifyThat(te
```

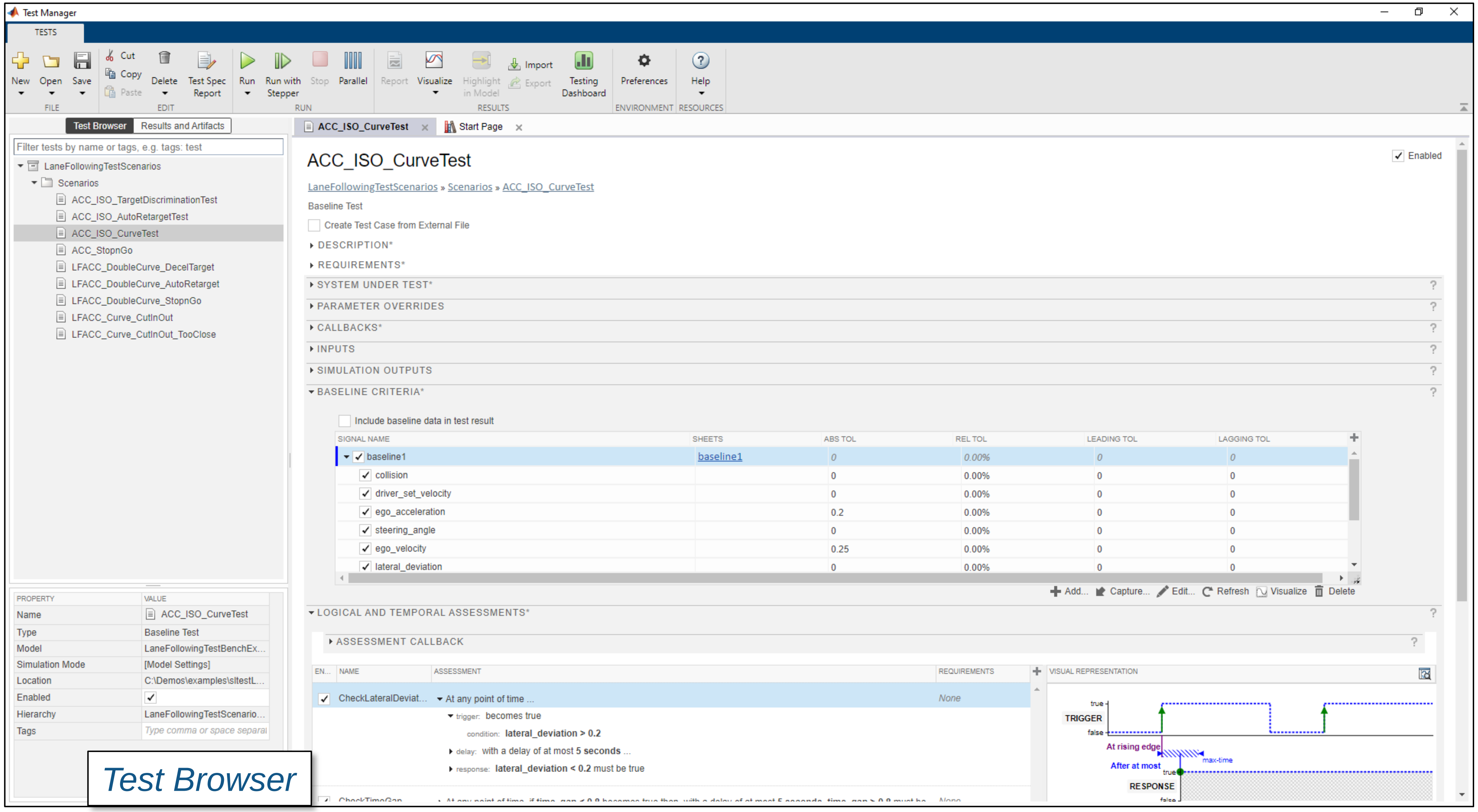
MATLAB Code

and more!

MATLAB EXPO

Test & Verification

Test Manager: Manage and organize tests



Filter tests by name or tags, e.g. tags: test

- LaneFollowingTestScenarios
  - Scenarios
    - ACC_ISO_TargetDiscriminationTest
    - ACC_ISO_AutoRetargetTest
    - ACC_ISO_CurveTest
    - ACC_StopnGo
    - LFACC_DoubleCurve_DecelTarget
    - LFACC_DoubleCurve_AutoRetarget
    - LFACC_DoubleCurve_StopnGo
    - LFACC_Curve_CutInOut
    - LFACC_Curve_CutInOut_TooClose

PROPERTY	VALUE
Name	ACC_ISO_CurveTest
Type	Baseline Test
Model	LaneFollowingTestBenchEx...
Simulation Mode	[Model Settings]
Location	C:\Demos\examples\sltestL...
Enabled	✓
Hierarchy	LaneFollowingTestScenario...
Tags	Type comma or space separat

ACC_ISO_CurveTest

LaneFollowingTestScenarios » Scenarios » ACC_ISO_CurveTest

Baseline Test

Create Test Case from External File

DESCRIPTION*

REQUIREMENTS*

SYSTEM UNDER TEST*

PARAMETER OVERRIDES

CALLBACKS*

INPUTS

SIMULATION OUTPUTS

BASELINE CRITERIA*

Include baseline data in test result

SIGNAL NAME	SHEETS	ABS TOL	REL TOL	LEADING TOL	LAGGING TOL
✓ baseline1	baseline1	0	0.00%	0	0
✓ collision		0	0.00%	0	0
✓ driver_set_velocity		0	0.00%	0	0
✓ ego_acceleration		0.2	0.00%	0	0
✓ steering_angle		0	0.00%	0	0
✓ ego_velocity		0.25	0.00%	0	0
✓ lateral_deviation		0	0.00%	0	0

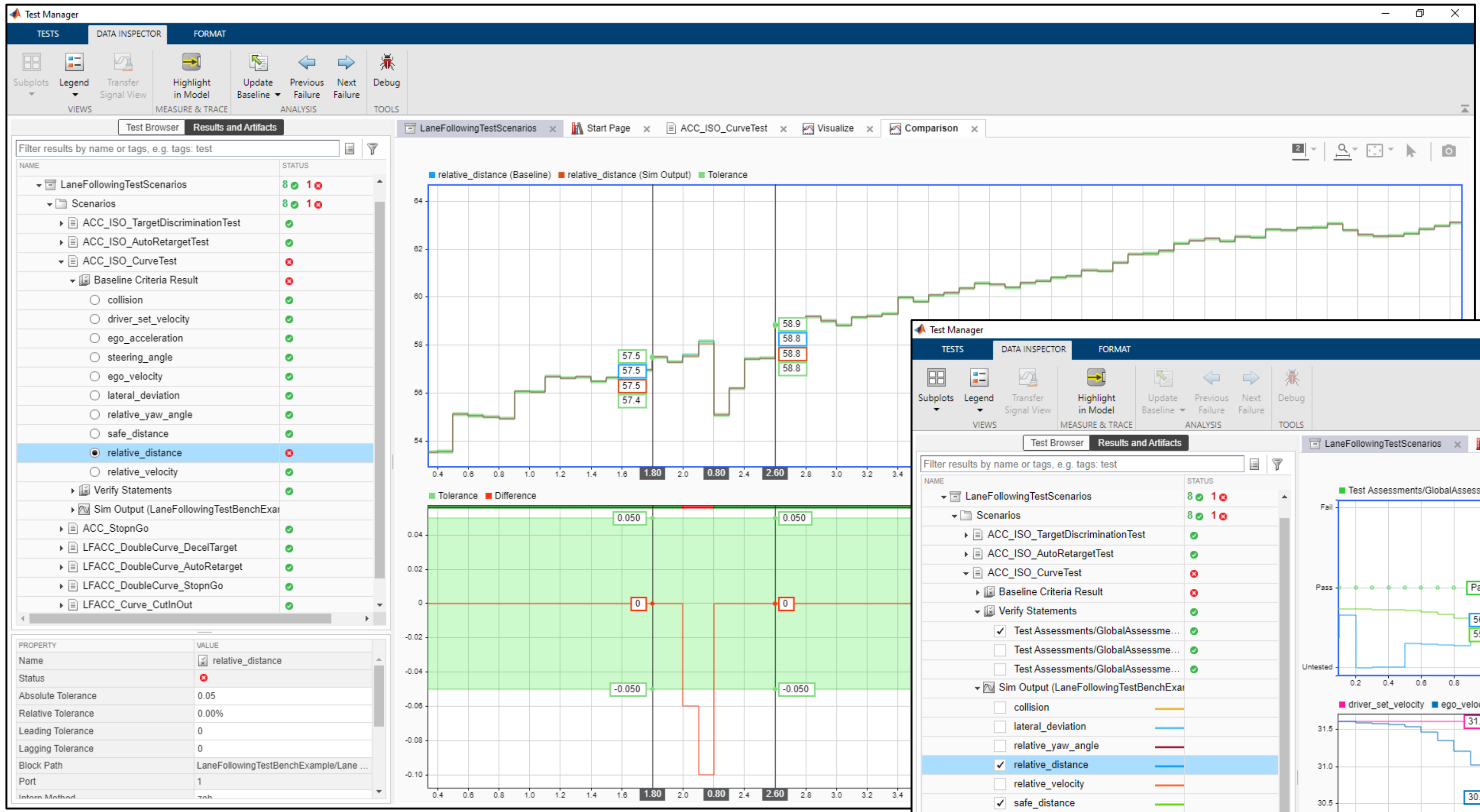
LOGICAL AND TEMPORAL ASSESSMENTS*

ASSESSMENT CALLBACK

EN...	NAME	ASSESSMENT	REQUIREMENTS
✓	CheckLateralDeviat...	At any point of time ... trigger: becomes true condition: lateral_deviation > 0.2 delay: with a delay of at most 5 seconds ... response: lateral_deviation < 0.2 must be true	None
✓	CheckTimeGap	At any point of time, if time_gap < 0.8 becomes true then, with a delay of at most 5 seconds, time_gap > 0.8 must be	None

Test Browser

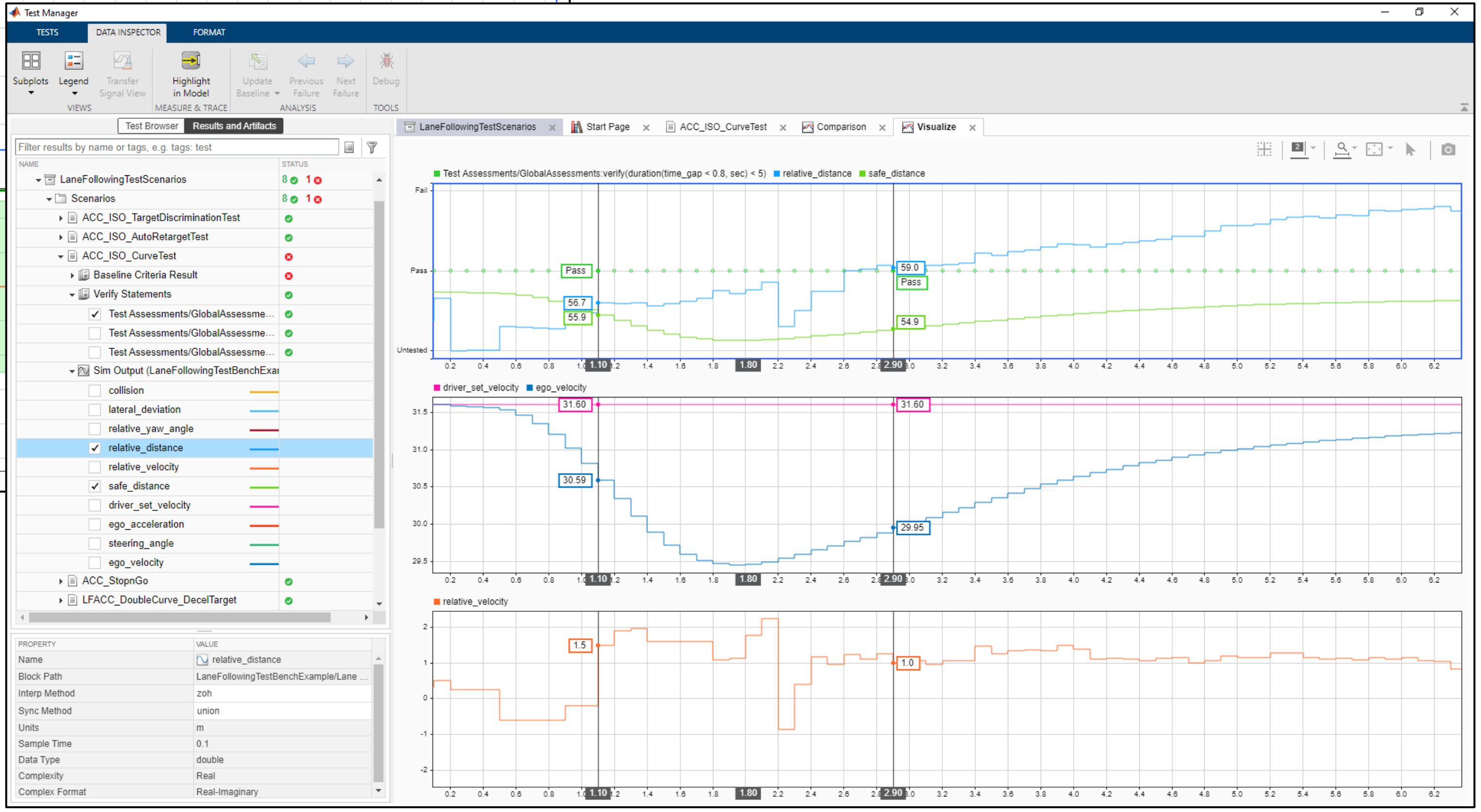
Test Manager: View and debug test results



Filter results by name or tags, e.g. tags: test

NAME	STATUS
LaneFollowingTestScenarios	8 ✓ 1 ✗
Scenarios	8 ✓ 1 ✗
ACC_ISO_TargetDiscriminationTest	✓
ACC_ISO_AutoRetargetTest	✓
ACC_ISO_CurveTest	✗
Baseline Criteria Result	✗
collision	✓
driver_set_velocity	✓
ego_acceleration	✓
steering_angle	✓
ego_velocity	✓
lateral_deviation	✓
relative_yaw_angle	✓
safe_distance	✓
relative_distance	✗
relative_velocity	✓
Verify Statements	✓
Sim Output (LaneFollowingTestBenchExa	
ACC_StopnGo	✓
LFACC_DoubleCurve_DecelTarget	✓
LFACC_DoubleCurve_AutoRetarget	✓
LFACC_DoubleCurve_StopnGo	✓
LFACC_Curve_CutInOut	✓

PROPERTY	VALUE
Name	relative_distance
Status	✗
Absolute Tolerance	0.05
Relative Tolerance	0.00%
Leading Tolerance	0
Lagging Tolerance	0
Block Path	LaneFollowingTestBenchExample/Lane ...
Port	1



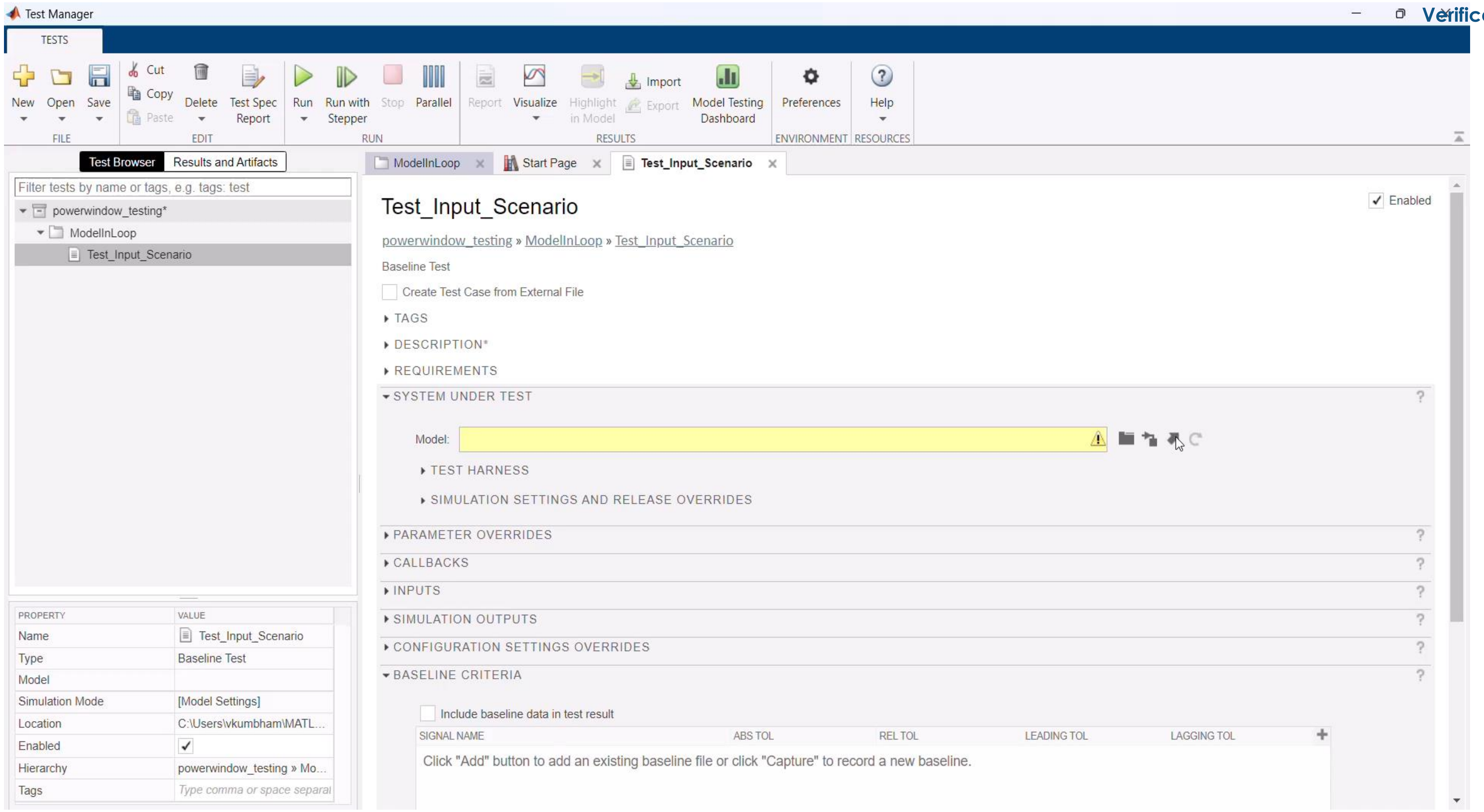
Filter results by name or tags, e.g. tags: test

NAME	STATUS
LaneFollowingTestScenarios	8 ✓ 1 ✗
Scenarios	8 ✓ 1 ✗
ACC_ISO_TargetDiscriminationTest	✓
ACC_ISO_AutoRetargetTest	✓
ACC_ISO_CurveTest	✗
Baseline Criteria Result	✗
Verify Statements	✓
✓ Test Assessments/GlobalAssessme...	✓
Test Assessments/GlobalAssessme...	✓
Test Assessments/GlobalAssessme...	✓
Sim Output (LaneFollowingTestBenchExa	
collision	
lateral_deviation	
relative_yaw_angle	
✓ relative_distance	
relative_velocity	
✓ safe_distance	
driver_set_velocity	
ego_acceleration	
steering_angle	
ego_velocity	
ACC_StopnGo	✓
LFACC_DoubleCurve_DecelTarget	✓

PROPERTY	VALUE
Name	relative_distance
Block Path	LaneFollowingTestBenchExample/Lane ...
Interp Method	zoh
Sync Method	union
Units	m
Sample Time	0.1
Data Type	double
Complexity	Real
Complex Format	Real-Imaginary

Test Results

Authoring Test Cases:



Filter tests by name or tags, e.g. tags: test

- powerwindow_testing*
  - ModelInLoop
    - Test_Input_Scenario

PROPERTY	VALUE
Name	Test_Input_Scenario
Type	Baseline Test
Model	
Simulation Mode	[Model Settings]
Location	C:\Users\vkumbham\MATL...
Enabled	✓
Hierarchy	powerwindow_testing » Mo...
Tags	Type comma or space separal

Test_Input_Scenario

✓ Enabled

powerwindow_testing » ModelInLoop » Test_Input_Scenario

Baseline Test

☐ Create Test Case from External File

- TAGS
- DESCRIPTION*
- REQUIREMENTS
- SYSTEM UNDER TEST
  - Model:
  - TEST HARNESS
  - SIMULATION SETTINGS AND RELEASE OVERRIDES
- PARAMETER OVERRIDES
- CALLBACKS
- INPUTS
- SIMULATION OUTPUTS
- CONFIGURATION SETTINGS OVERRIDES
- BASELINE CRITERIA

☐ Include baseline data in test result

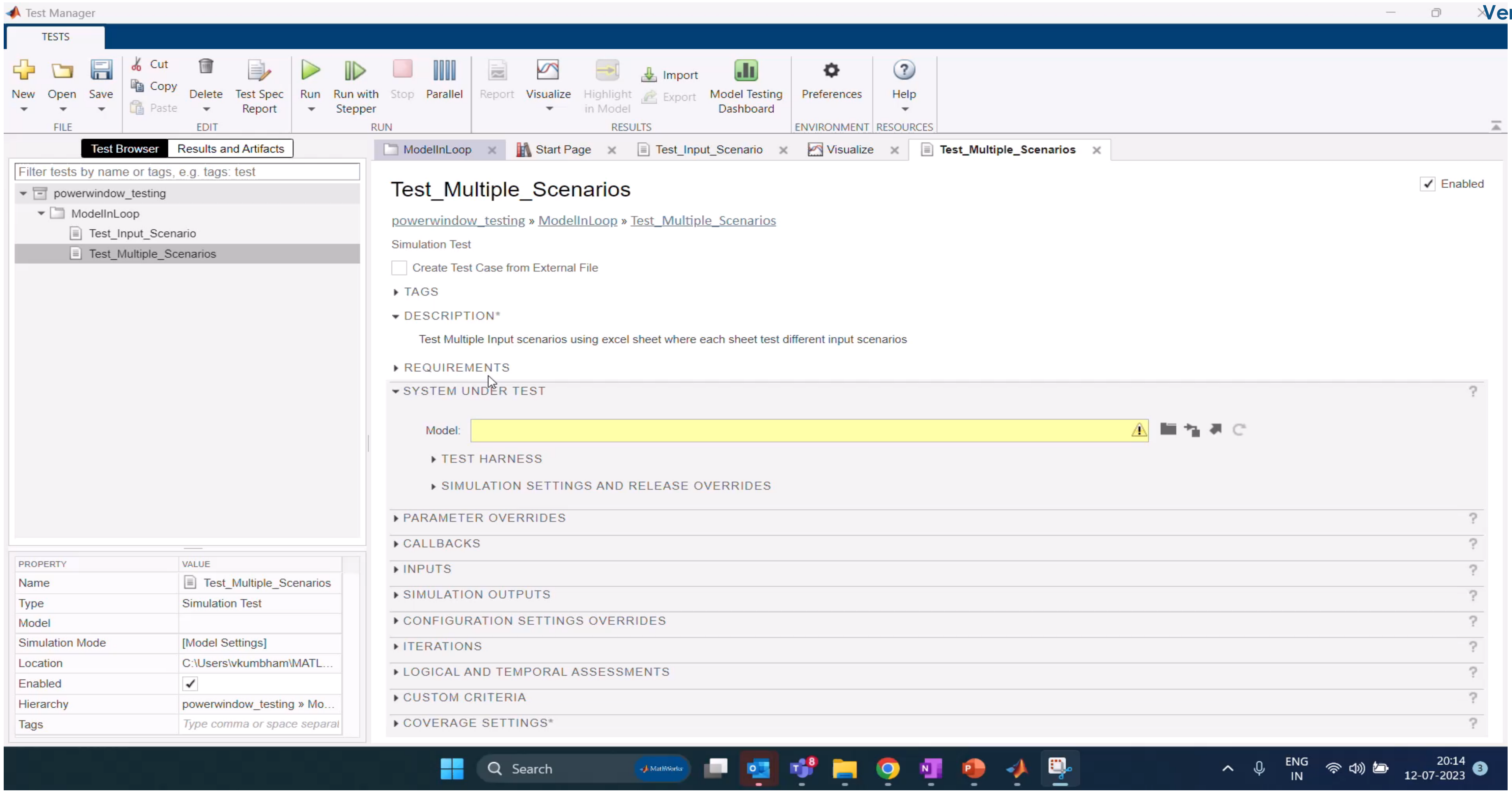
SIGNAL NAME	ABS TOL	REL TOL	LEADING TOL	LAGGING TOL

Click "Add" button to add an existing baseline file or click "Capture" to record a new baseline.

MATLAB EXPO

Test & Verification

Adding more test cases for better coverage:



Test Manager

TESTS

New | Open | Save | Cut | Copy | Paste | Delete | Test Spec Report | Run | Run with Stepper | Stop | Parallel | Report | Visualize | Highlight in Model | Import | Export | Model Testing Dashboard | Preferences | Help

FILE | EDIT | RUN | RESULTS | ENVIRONMENT | RESOURCES

Test Browser | Results and Artifacts

Filter tests by name or tags, e.g. tags: test

- powerwindow_testing
  - ModelInLoop
    - Test_Input_Scenario
    - Test_Multiple_Scenarios

PROPERTY	VALUE
Name	Test_Multiple_Scenarios
Type	Simulation Test
Model	
Simulation Mode	[Model Settings]
Location	C:\Users\vkumbham\MATL...
Enabled	✓
Hierarchy	powerwindow_testing » Mo...
Tags	Type comma or space separat

ModelInLoop | Start Page | Test_Input_Scenario | Visualize | Test_Multiple_Scenarios

Test_Multiple_Scenarios

✓ Enabled

powerwindow_testing » ModelInLoop » Test_Multiple_Scenarios

Simulation Test

Create Test Case from External File

TAGS

DESCRIPTION*

Test Multiple Input scenarios using excel sheet where each sheet test different input scenarios

REQUIREMENTS

SYSTEM UNDER TEST

Model:

TEST HARNESS

SIMULATION SETTINGS AND RELEASE OVERRIDES

PARAMETER OVERRIDES

CALLBACKS

INPUTS

SIMULATION OUTPUTS

CONFIGURATION SETTINGS OVERRIDES

ITERATIONS

LOGICAL AND TEMPORAL ASSESSMENTS

CUSTOM CRITERIA

COVERAGE SETTINGS*

Search | ENG IN | 20:14 12-07-2023

MATLAB EXPO

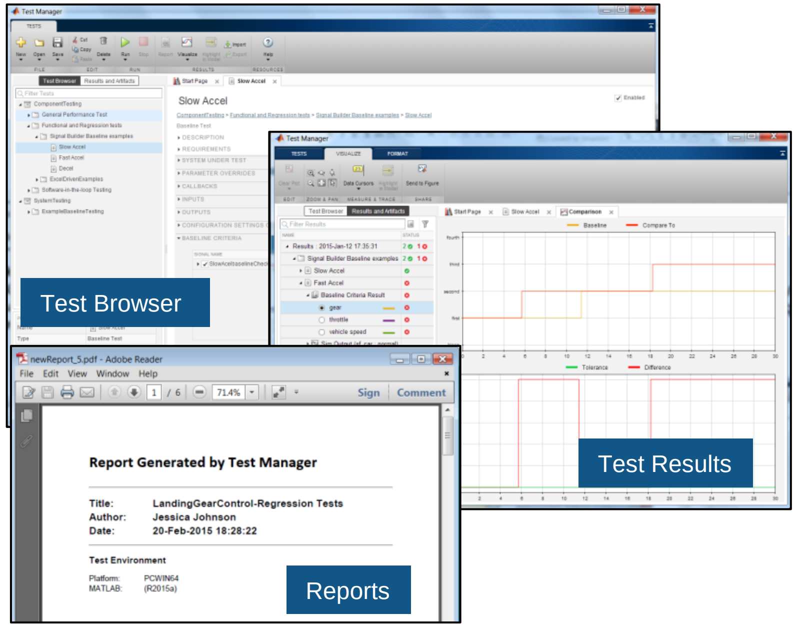
Test & Verification

Simulink Test

Develop, manage, and execute simulation-based tests

Test Manager

- Author, manage, organize tests
- Execute simulation, equivalence and baseline tests
- Review, export, report



Report Generated by Test Manager

Title: LandingGearControl-Regression Tests
Author: Jessica Johnson
Date: 20-Feb-2015 18:28:22

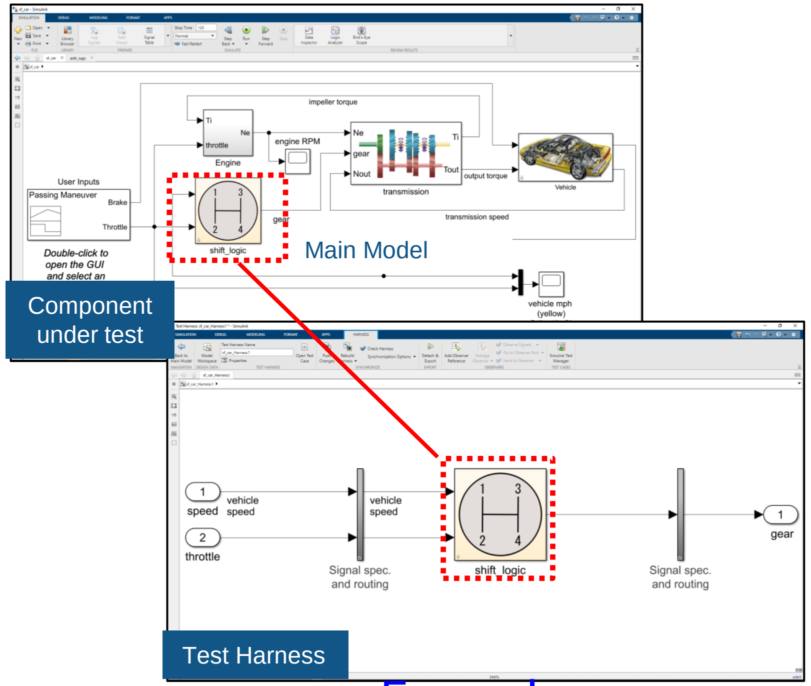
Test Environment

Platform: PCWIN64
MATLAB: (R2015a)

[Examples](#)

Test Harnesses

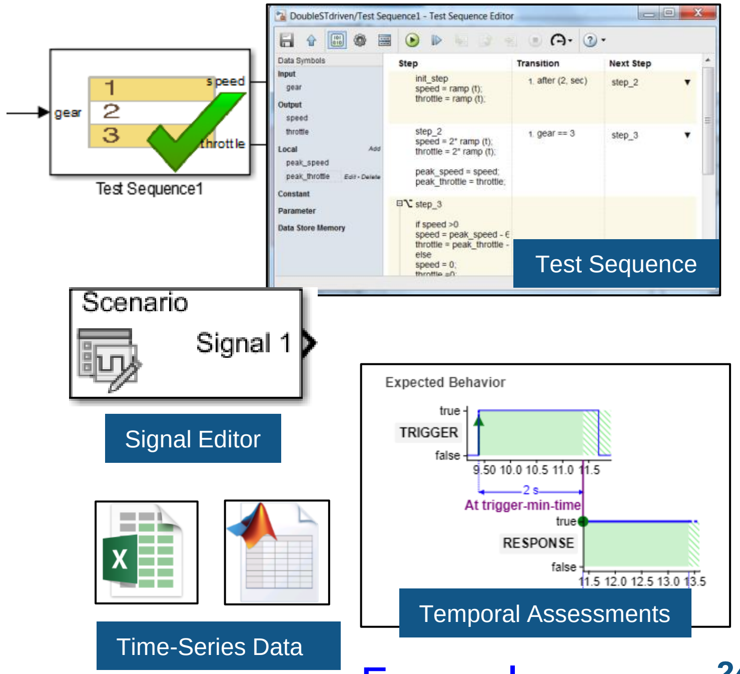
- Isolate Component Under Test
- Synchronized, simulation test environment



[Examples](#)

Test Authoring

- Specify test inputs, expected outputs, and tolerances
- Construct complex test sequences and assessments



[Examples](#)

MATLAB EXPO

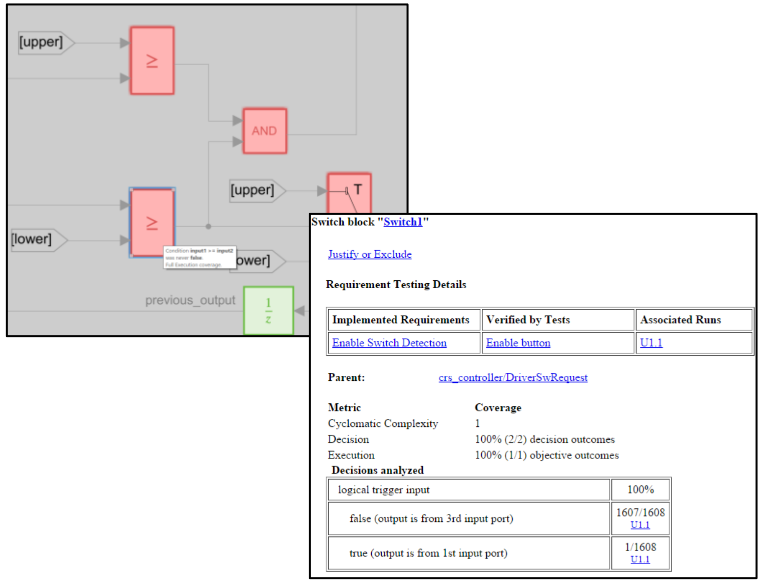
Test & Verification

Simulink Coverage

Measure test coverage in models and generated code

Model Coverage

- Measure test completeness
- Identify missing tests or unintended functionality



Switch block "Switch1"

Justify or Exclude

Requirement Testing Details

Implemented Requirements	Verified by Tests	Associated Runs
Enable Switch Detection	Enable button	U1.1

Parent: crs_controller/DriverSwRequest

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

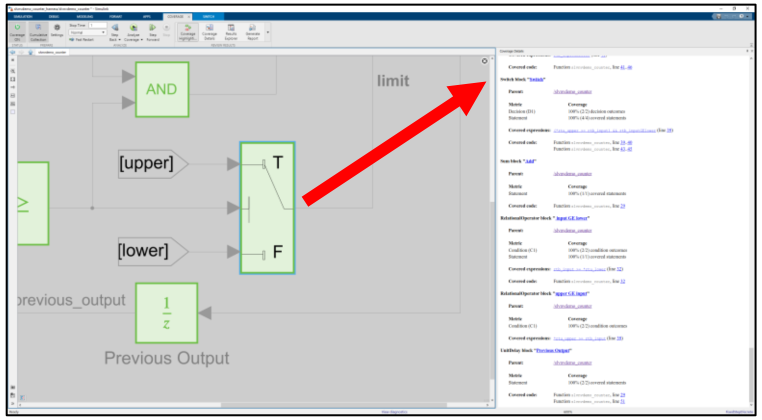
Decisions analyzed

logical trigger input	100%
false (output is from 3rd input port)	1607/1608 U1.1
true (output is from 1st input port)	1/1608 U1.1

Examples

Generated Code Coverage

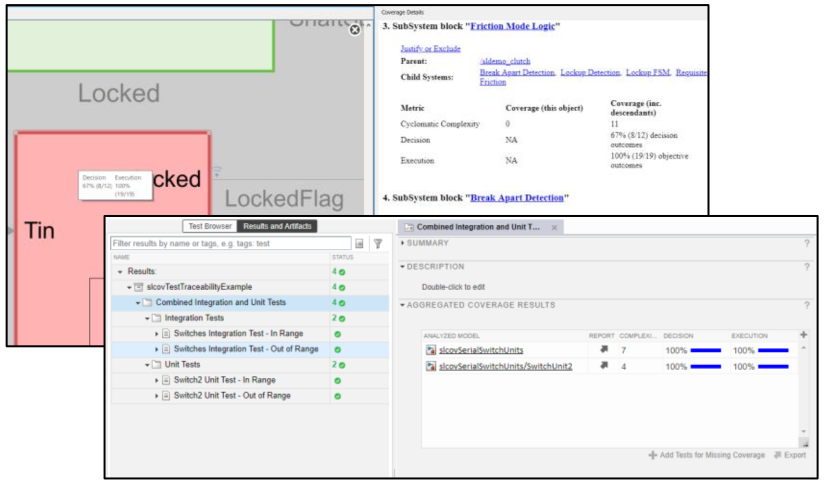
- Find untested generated code
- Map results from code to model object



Examples

Highlighting and Reporting

- View coverage results on diagrams
- Manage accumulated coverage results


Locked
cked
LockedFlag
Tin
Test Browser
Results and Artifacts
Filter results by name or tags, e.g. tags: test
SUMMARY
DESCRIPTION
Double-click to edit
AGGREGATED COVERAGE RESULTS
Add Tests for Missing Coverage
Export

Coverage Details

3. SubSystem block "Friction Mode Logic"

Justify or Exclude

Parent: /sldemo_clutch

Child Systems: Break Apart Detection, Lockup Detection, Lockup FSM, Requisite Friction

Metric	Coverage (this object)	Coverage (inc. descendants)
Cyclomatic Complexity	0	11
Decision	NA	67% (8/12) decision outcomes
Execution	NA	100% (19/19) objective outcomes

4. SubSystem block "Break Apart Detection"

NAME	STATUS
Results	4
slcovTestTraceabilityExample	4
Combined Integration and Unit Tests	4
Integration Tests	2
Switches Integration Test - In Range	
Switches Integration Test - Out of Range	
Unit Tests	2
Switch2 Unit Test - In Range	
Switch2 Unit Test - Out of Range	

Combined Integration and Unit T...

ANALYZED MODEL	REPORT	COMPLEXI...	DECISION	EXECUTION
slcovSerialSwitchUnits		7	100%	100%
slcovSerialSwitchUnits/SwitchUnit2		4	100%	100%

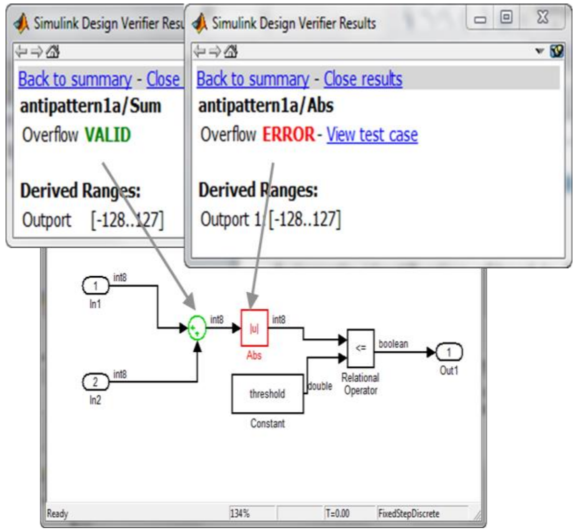
Examples

Simulink Design Verifier

Use formal methods to identify design errors

Design Error Detection

- **Uncover** hard to find dead logic and design flaws



Simulink Design Verifier Results

Back to summary - Close results

antipattern1a/Sum

Overflow VALID

Derived Ranges:

Outport [-128..127]

antipattern1a/Abs

Overflow ERROR - View test case

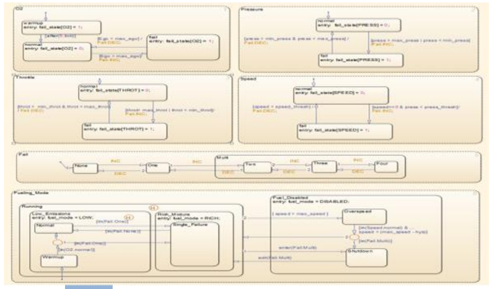
Derived Ranges:

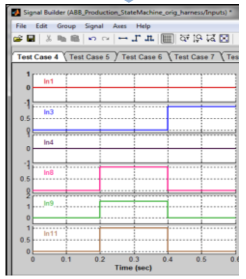
Outport 1 [-128..127]

Examples

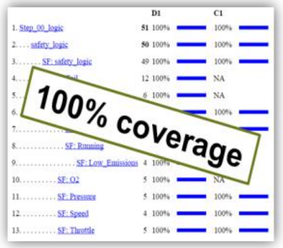
Test Generation

- **Automate** test vector generation to analyze missing coverage





Signal Builder (ABB_Production_StateMachine_orig_harness/Inputs)
File Edit Group Signal Axes Help
Test Case 4
Test Case 5
Test Case 6
Test Case 7
In1
In3
In4
In8
In9
In11
Time (sec)



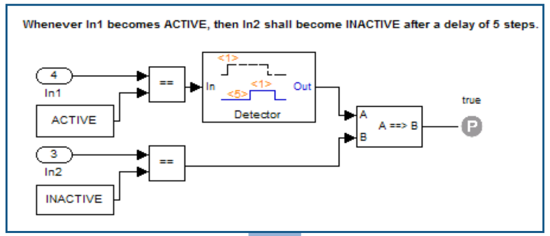
		D1		C1
1. Step_00_logic	51	100%		100%
2. . . . safety_logic	50	100%		100%
3. . . . . . . SF: safety_logic	49	100%		100%
4. . . [illegible]	12	100%		NA
5. . [illegible]	6	100%		NA
6. [illegible]	[illegible]	[illegible]		100%
7. . . . . [illegible]	[illegible]	[illegible]		[illegible]
8. . . . . SF: Running	[illegible]	[illegible]		[illegible]
9. . . . . SF: Low_Emissions	4	100%		[illegible]
10. . . . . SF: O2	5	100%		NA
11. . . . . SF: Pressure	5	100%		100%
12. . . . . SF: Speed	4	100%		100%
13. . . . . SF: Throttle	5	100%		100%

100% coverage

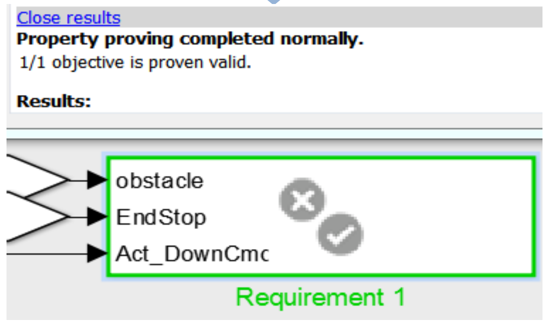
Examples

Requirements Proving

- **Prove** formally design meets requirements



Whenever In1 becomes ACTIVE, then In2 shall become INACTIVE after a delay of 5 steps.



Close results

Property proving completed normally.

1/1 objective is proven valid.

Results:

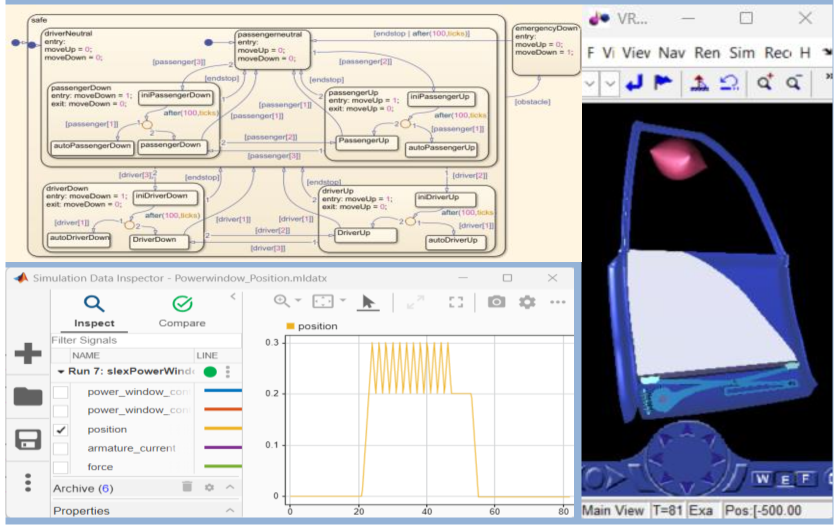
Examples

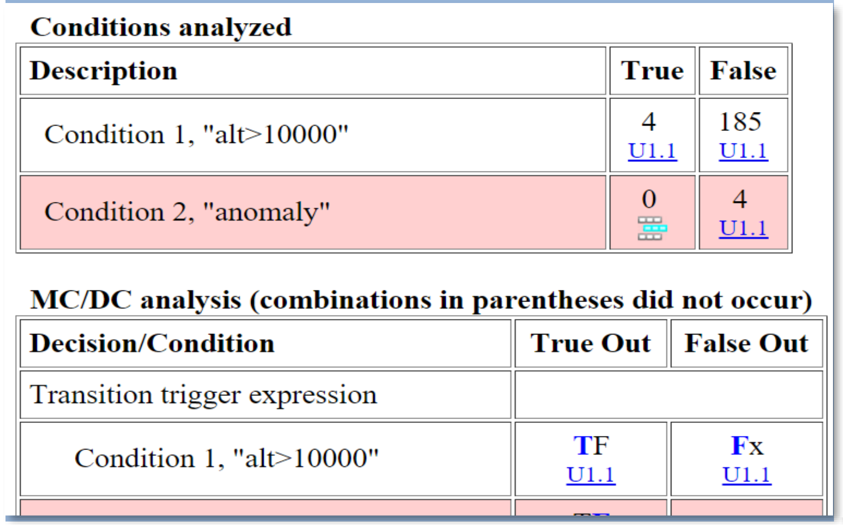
There are three key pieces to Model-Based Design

✓ Modeling & Simulation

✓ Test & Validation

Code Generation & Code Verification



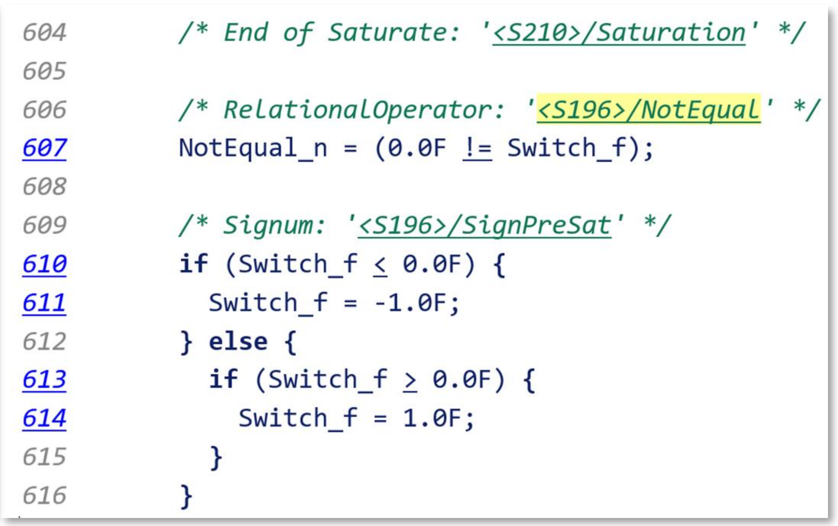


Conditions analyzed

Description	True	False
Condition 1, "alt>10000"	4 U1.1	185 U1.1
Condition 2, "anomaly"	0	4 U1.1

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition	True Out	False Out
Transition trigger expression		
Condition 1, "alt>10000"	TF U1.1	Fx U1.1



```
        /* End of Saturate: '<S210>/Saturation' */

        /* RelationalOperator: '<S196>/NotEqual' */
        NotEqual_n = (0.0F != Switch_f);

        /* Signum: '<S196>/SignPreSat' */
        if (Switch_f < 0.0F) {
          Switch_f = -1.0F;
        } else {
          if (Switch_f > 0.0F) {
            Switch_f = 1.0F;
          }
        }
```

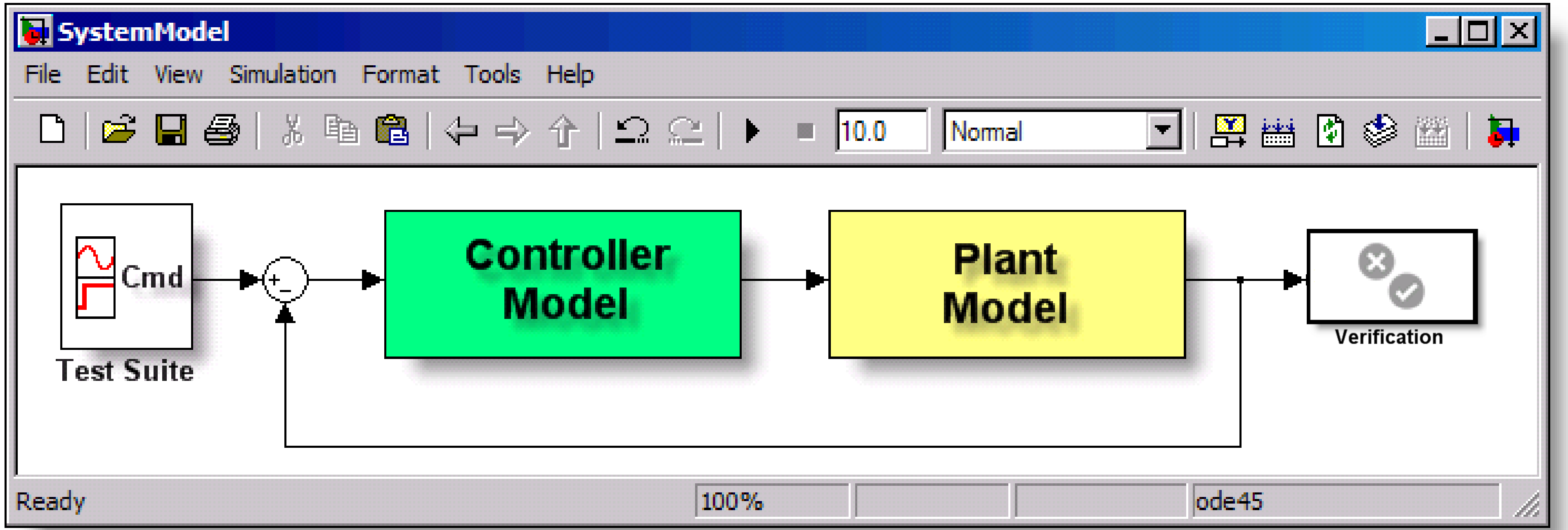
SIMULINK®

Simulation and Model-Based Design

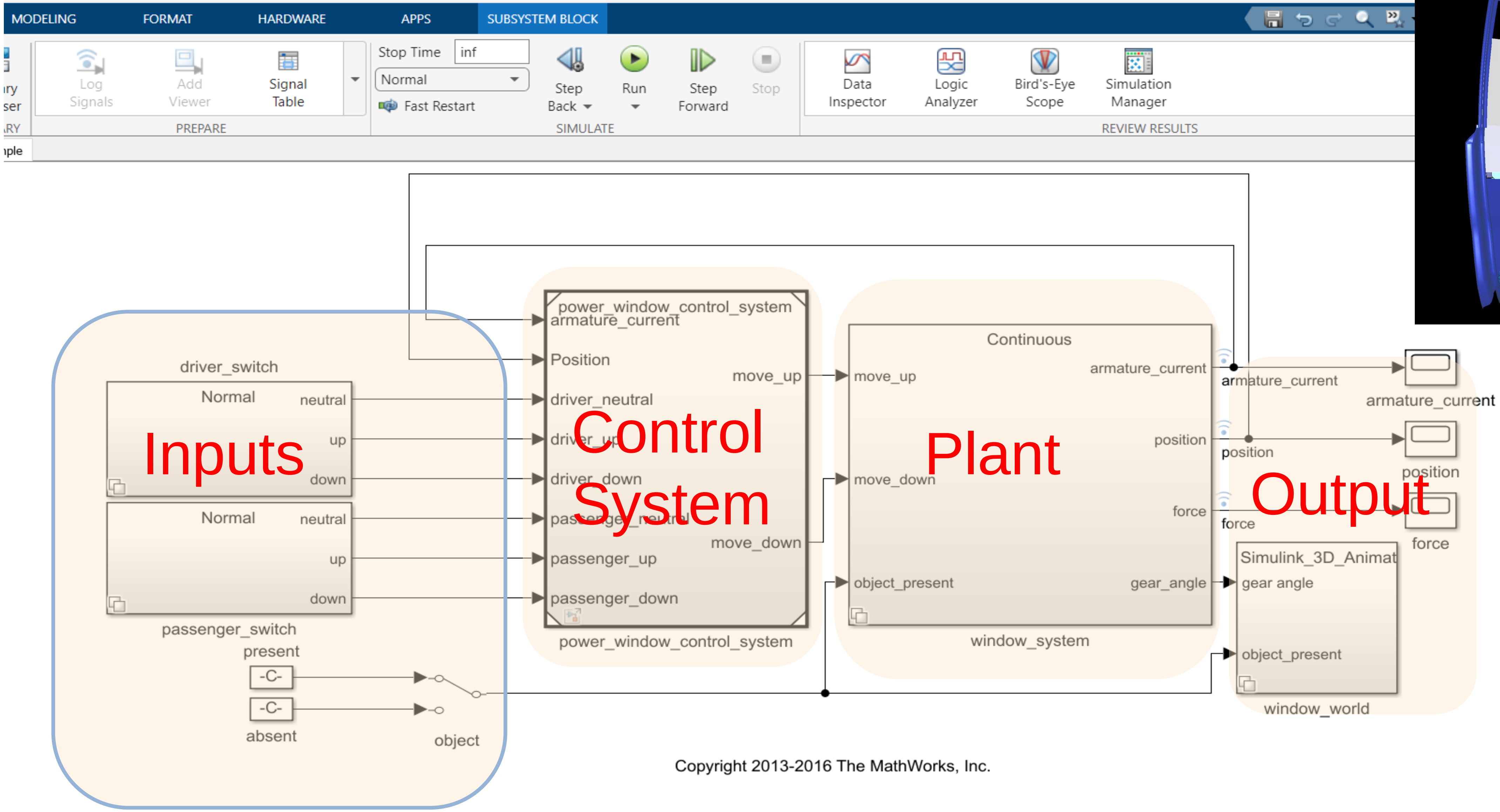
Model-in-the-Loop (MIL)

Verify models using simulations

- **Develop a model of the actual plant (hardware) in a simulation environment.**
- **Develop the controller model and verify if the controller can control the plant as per the requirement.**
- **Test the controller logic on the simulated model of the plant.**



Control Software testing: Model-in-the-Loop (MIL)

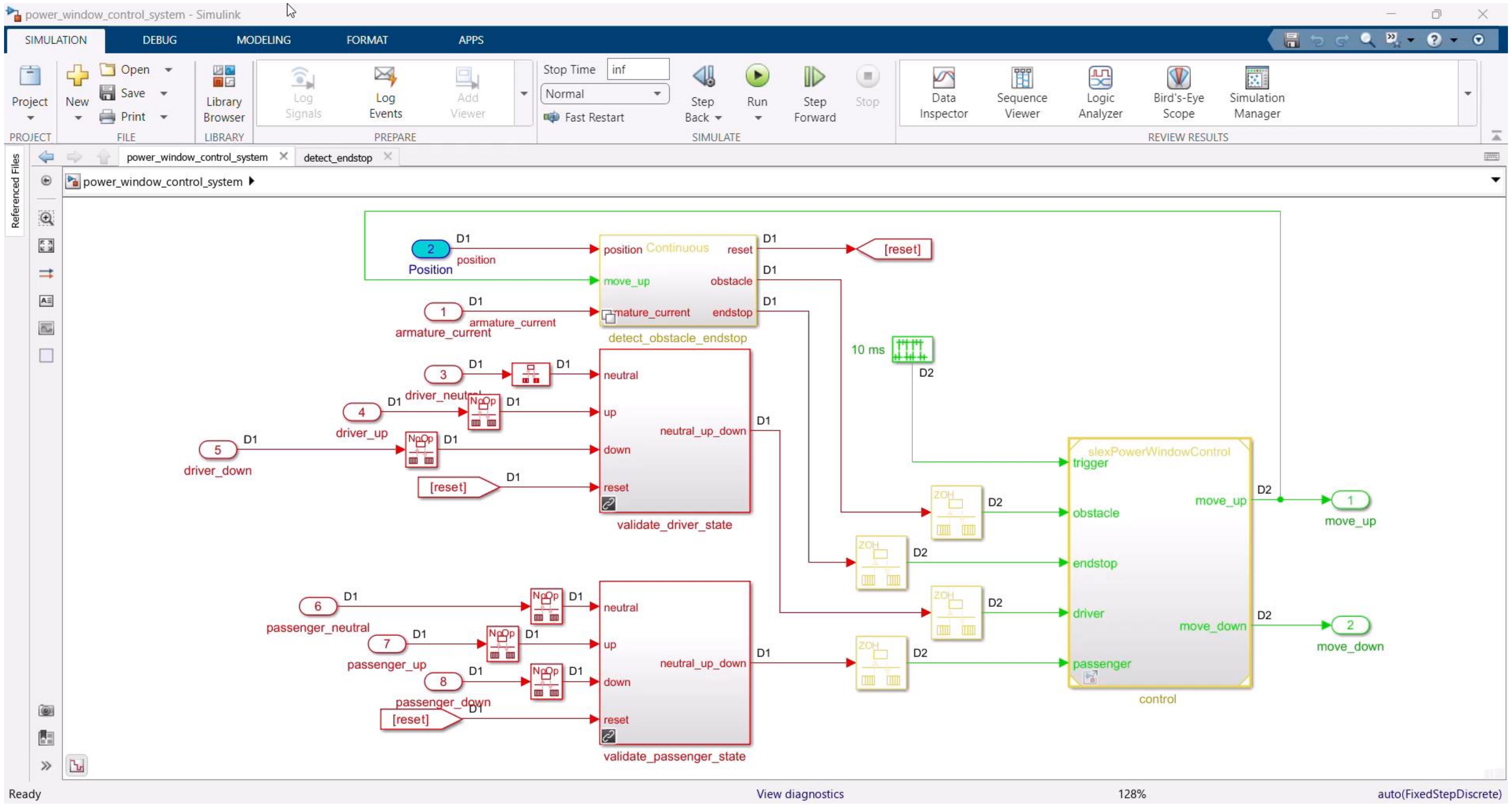


Copyright 2013-2016 The MathWorks, Inc.

MATLAB EXPO

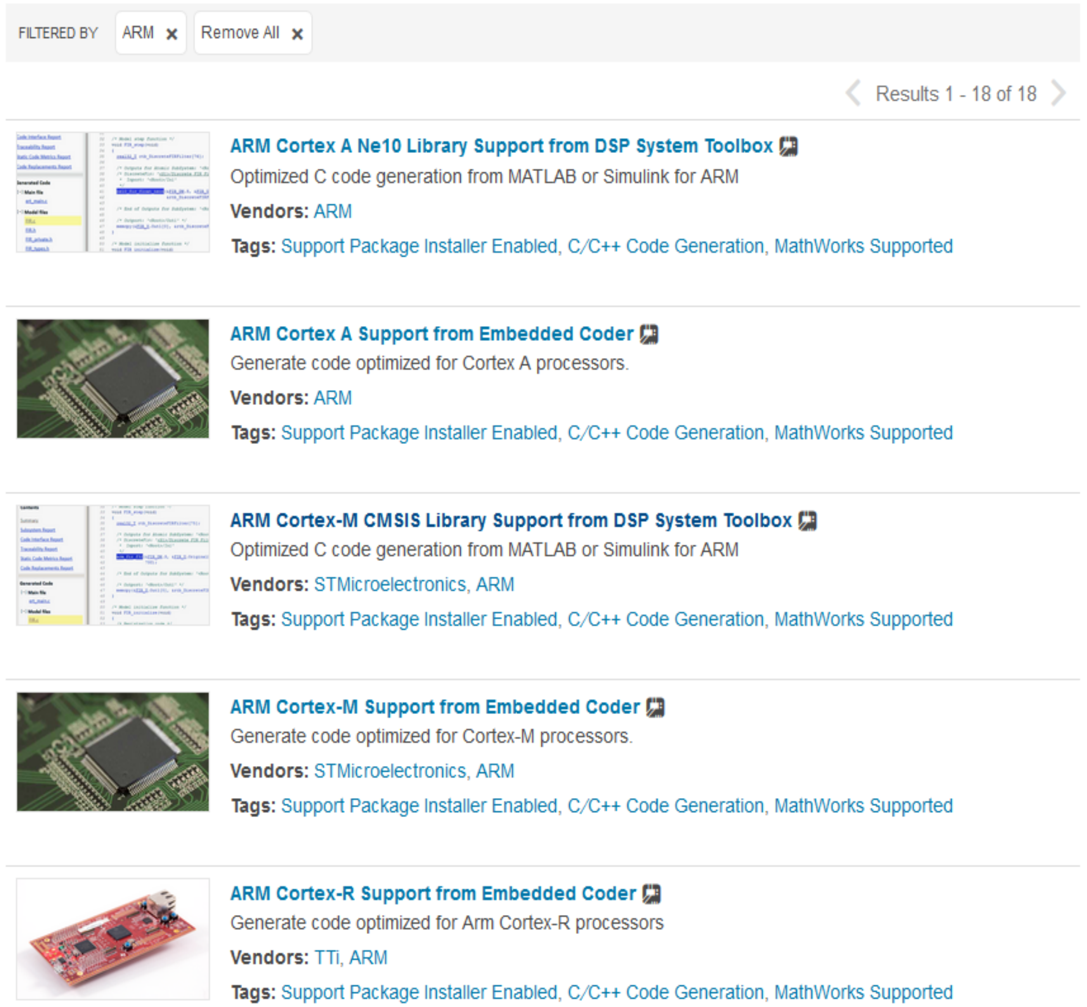
Code Gen & Verification

Automatic Code Generation:



Code Customization and Optimizations:

- Hardware Support Packages
- Code Replacement Libraries for Custom libraries eg.
  - ARM Cortex A Ne10
  - Intel SSE, AVX
  - ARM Cortex M CMSIS
- C Caller Block for external code integration
- S-Functions for legacy code
- Organization wide Custom Libraries via Code Replacement Libraries



FILTERED BY ARM × Remove All ×

Results 1 - 18 of 18

ARM Cortex A Ne10 Library Support from DSP System Toolbox
Optimized C code generation from MATLAB or Simulink for ARM
Vendors: ARM
Tags: Support Package Installer Enabled, C/C++ Code Generation, MathWorks Supported

ARM Cortex A Support from Embedded Coder
Generate code optimized for Cortex A processors.
Vendors: ARM
Tags: Support Package Installer Enabled, C/C++ Code Generation, MathWorks Supported

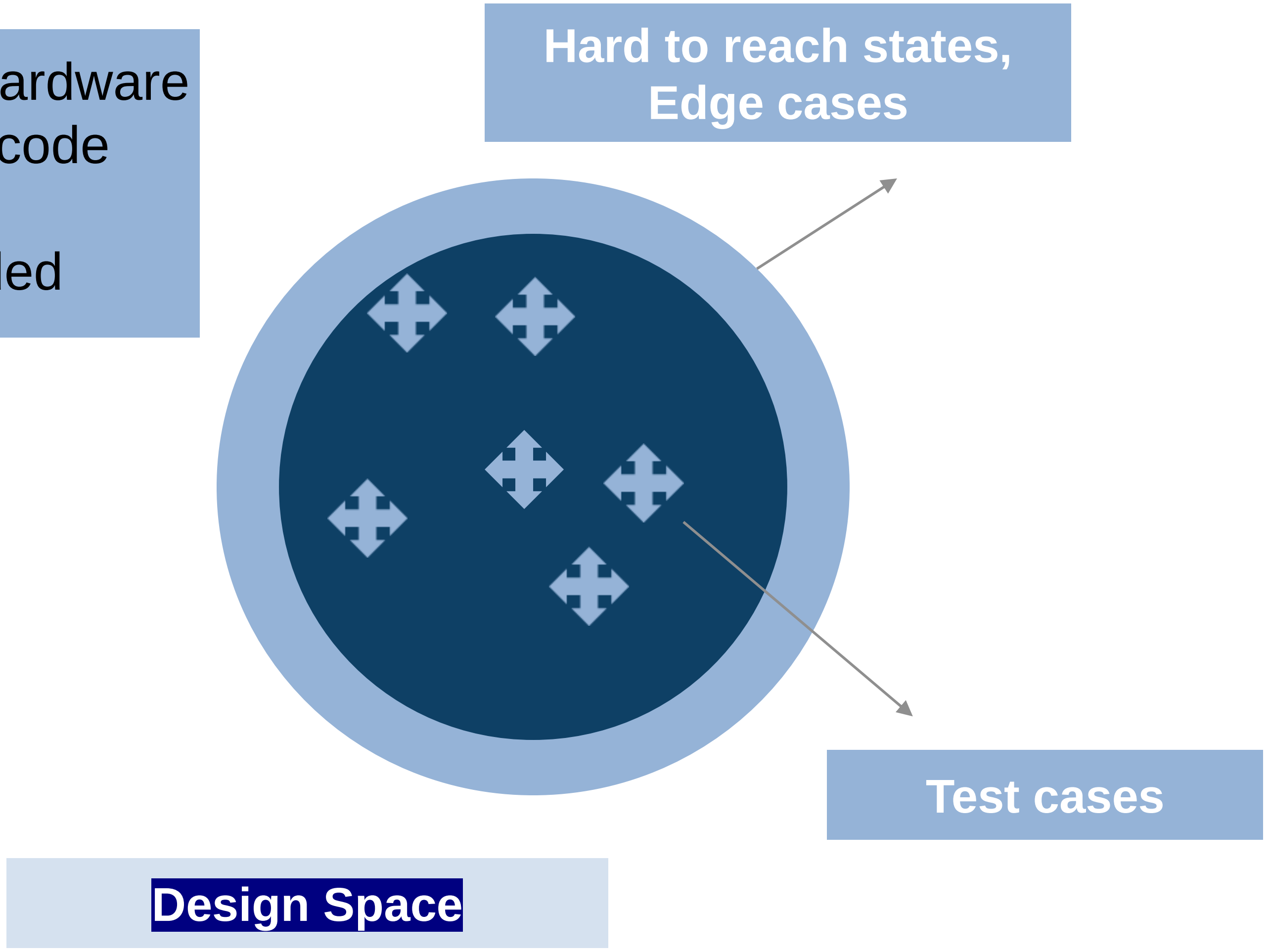
ARM Cortex-M CMSIS Library Support from DSP System Toolbox
Optimized C code generation from MATLAB or Simulink for ARM
Vendors: STMicroelectronics, ARM
Tags: Support Package Installer Enabled, C/C++ Code Generation, MathWorks Supported

ARM Cortex-M Support from Embedded Coder
Generate code optimized for Cortex-M processors.
Vendors: STMicroelectronics, ARM
Tags: Support Package Installer Enabled, C/C++ Code Generation, MathWorks Supported

ARM Cortex-R Support from Embedded Coder
Generate code optimized for Arm Cortex-R processors
Vendors: TTi, ARM
Tags: Support Package Installer Enabled, C/C++ Code Generation, MathWorks Supported

Why use Static Analysis?

- No dependency of hardware
- No execution of the code
- No instrumentation
- No tests cases needed



✓ Identify hard to reach states, Unusual runtime scenarios

✓ Apply consistent programming practices

✓ Run automated analysis early and often!

Polyspace Analysis on the Embedded Software

Code Generation

Hand-Written Code

Safe &Secure coding rules and known vulnerabilities

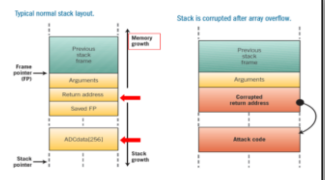
Secure code generation & deployment

- MISRA C
- CERT C, C++
- CWE
- TS 17961

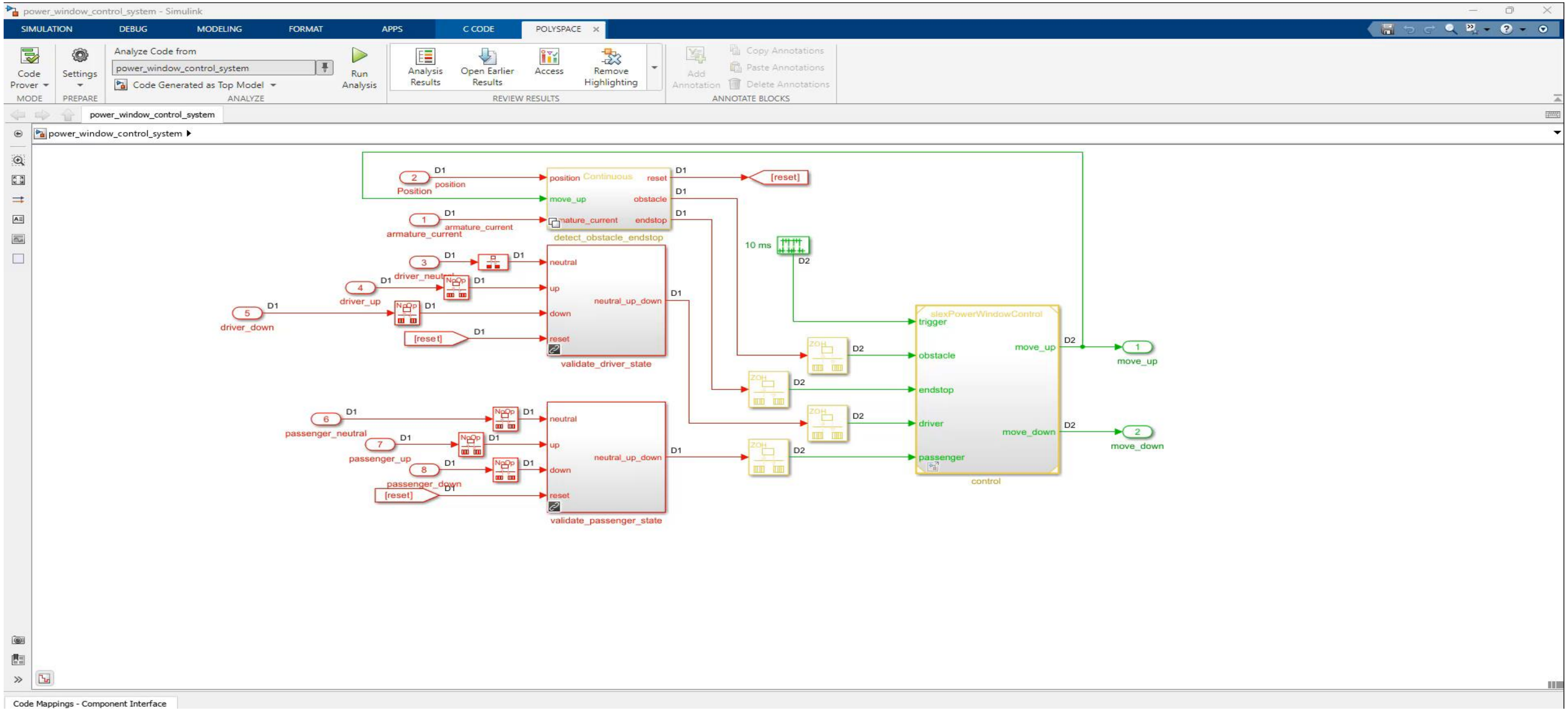
SW Implementation

Proof of robustness and Run-Time Error Detection

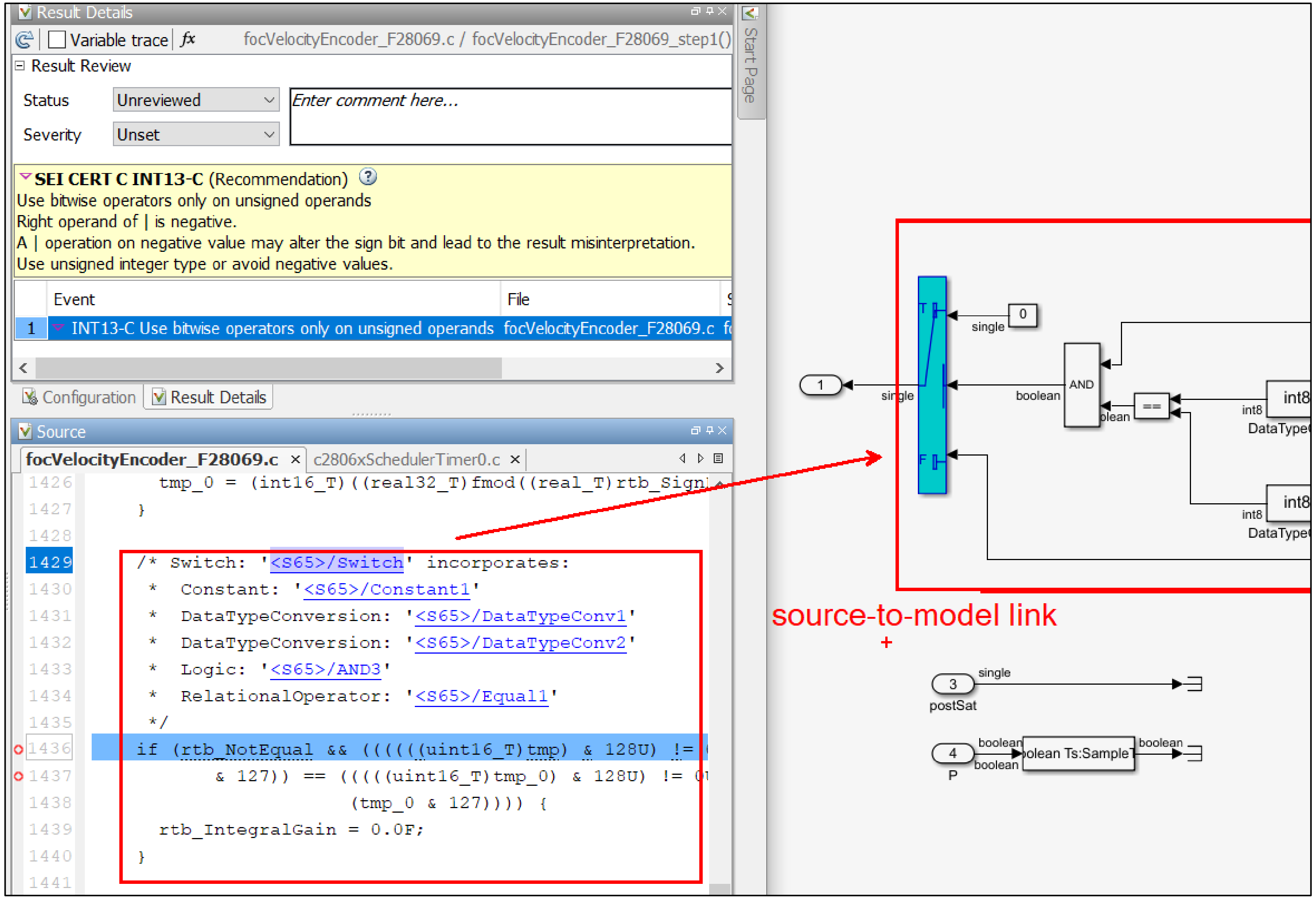
Code level security & robustness analysis



Launch Polyspace from Simulink



Traceability between Model and Code



Result Details

Variable trace | fx   focVelocityEncoder_F28069.c / focVelocityEncoder_F28069_step1()

Result Review

Status	Unreviewed
Severity	Unset

Enter comment here...

SEI CERT C INT13-C (Recommendation)
Use bitwise operators only on unsigned operands
Right operand of | is negative.
A | operation on negative value may alter the sign bit and lead to the result misinterpretation.
Use unsigned integer type or avoid negative values.

	Event	File
1	INT13-C Use bitwise operators only on unsigned operands	focVelocityEncoder_F28069.c

Configuration | Result Details

Source

focVelocityEncoder_F28069.c × | c2806xSchedulerTimer0.c ×

```
        tmp_0 = (int16_T)((real32_T)fmod((real_T)rtb_Signi
      }

      /* Switch: '<S65>/Switch' incorporates:
       *  Constant: '<S65>/Constant1'
       *  DataTypeConversion: '<S65>/DataTypeConv1'
       *  DataTypeConversion: '<S65>/DataTypeConv2'
       *  Logic: '<S65>/AND3'
       *  RelationalOperator: '<S65>/Equal1'
       */
      if (rtb_NotEqual && ((((((uint16_T)tmp) & 128U) !=
             & 127)) == ((((uint16_T)tmp_0) & 128U) != 0
                         (tmp_0 & 127)))) {
        rtb_IntegralGain = 0.0F;
      }
```

- Clickable links
- Bidirectional
- Trace requirements to code

Polyspace Source Code Analysis Solutions

✓ C
✓ C++
✓ Ada

Method:
Formal Method based analysis without need for code execution
→**Abstract Interpretation**

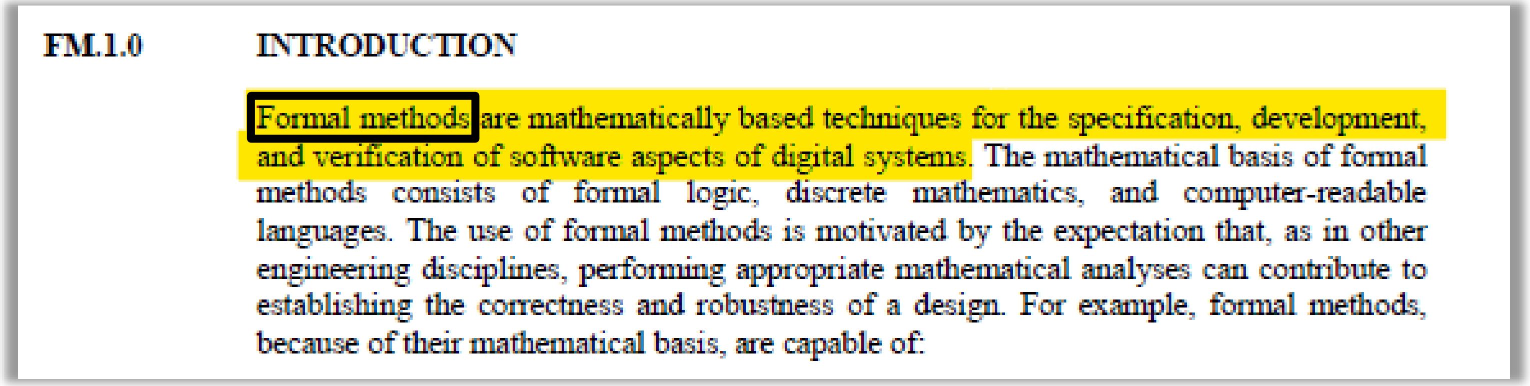
+

hand-written code

auto-generated code

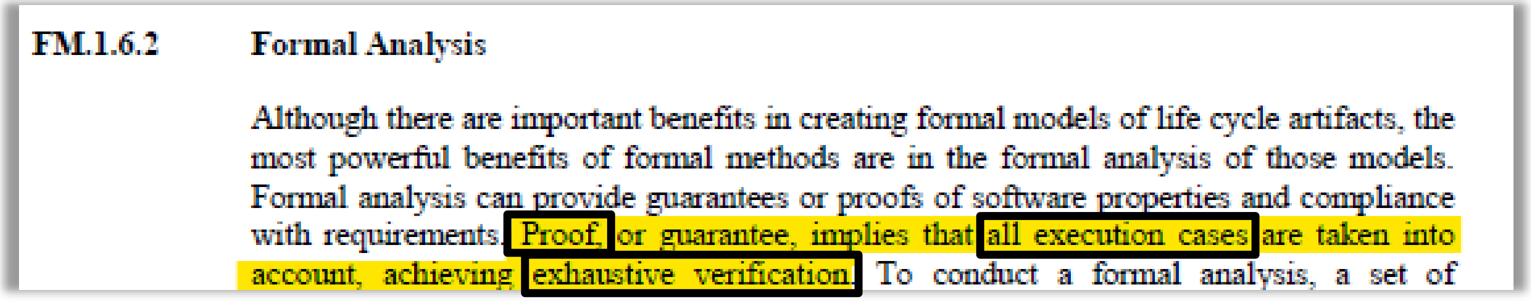
Goal:
- ✓ Find Runtime Errors (division by zero, overflows, etc.)
- ✓ Find Coding Standards violations
- ✓ Provide code metrics
- ✓ **Prove that all the software we rely on is safe and secure**

Formal Methods for Functional Safety



FM.1.0 INTRODUCTION

Formal methods are mathematically based techniques for the specification, development, and verification of software aspects of digital systems. The mathematical basis of formal methods consists of formal logic, discrete mathematics, and computer-readable languages. The use of formal methods is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analyses can contribute to establishing the correctness and robustness of a design. For example, formal methods, because of their mathematical basis, are capable of:



FM.1.6.2 Formal Analysis

Although there are important benefits in creating formal models of life cycle artifacts, the most powerful benefits of formal methods are in the formal analysis of those models. Formal analysis can provide guarantees or proofs of software properties and compliance with requirements. Proof, or guarantee, implies that all execution cases are taken into account, achieving exhaustive verification. To conduct a formal analysis, a set of

DO-333 Formal Methods Supplement

Sound analysis means that the method never asserts a property to be true when it may not be true" : False Negative

Source: DO-333 Supplement on Formal Methods

Polyspace Products

Bug Finder

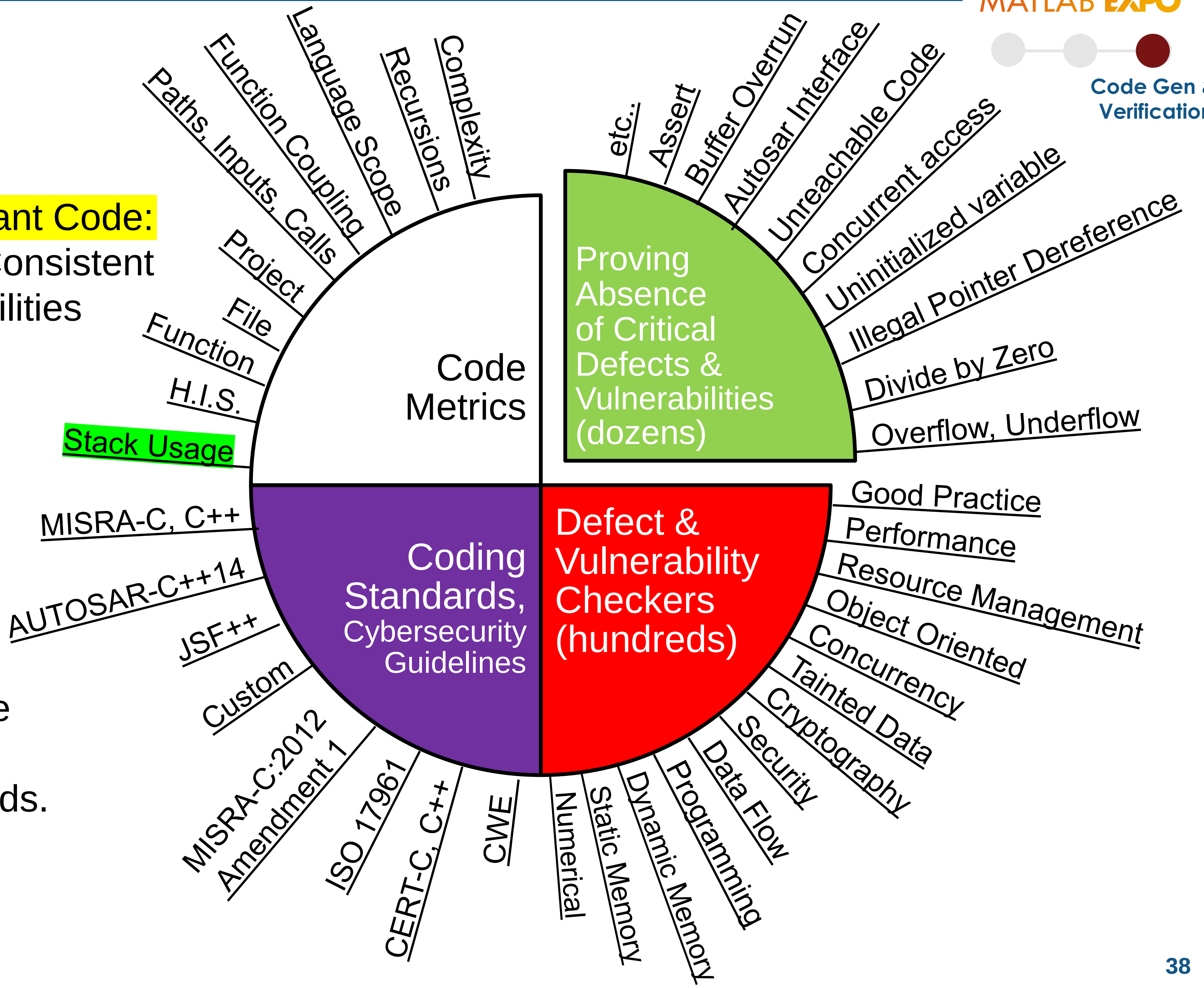
→High Quality, Secure, Compliant Code:

- Measurable, Maintainable, Consistent
- Very few defects or vulnerabilities
- Credits for functional safety, cybersecurity standards.

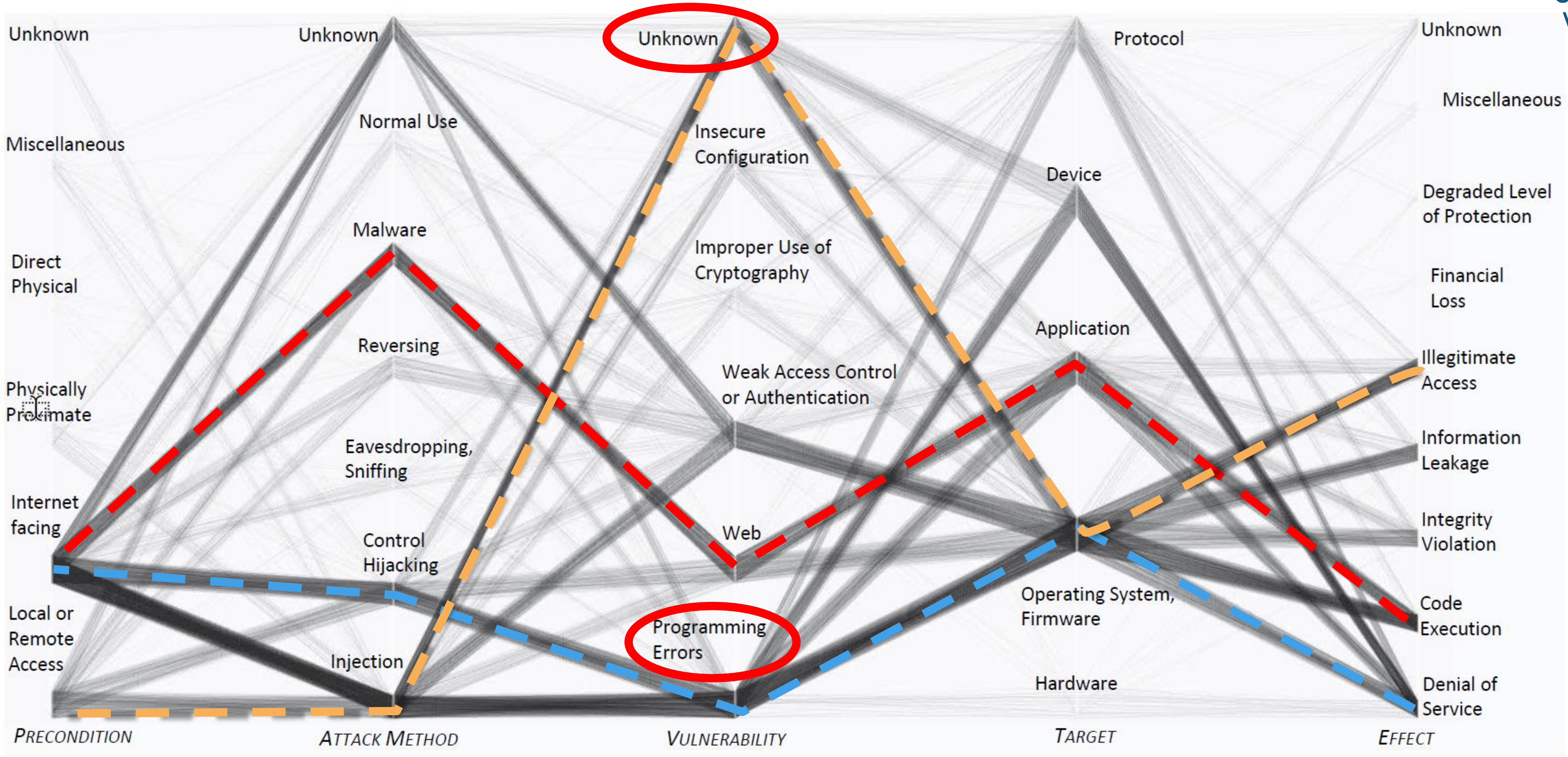
Code Prover

→Fully Trusted Components:

- Robust, Safe, Secure
- Proven free of critical runtime defects and vulnerabilities
- Additional credits for standards.



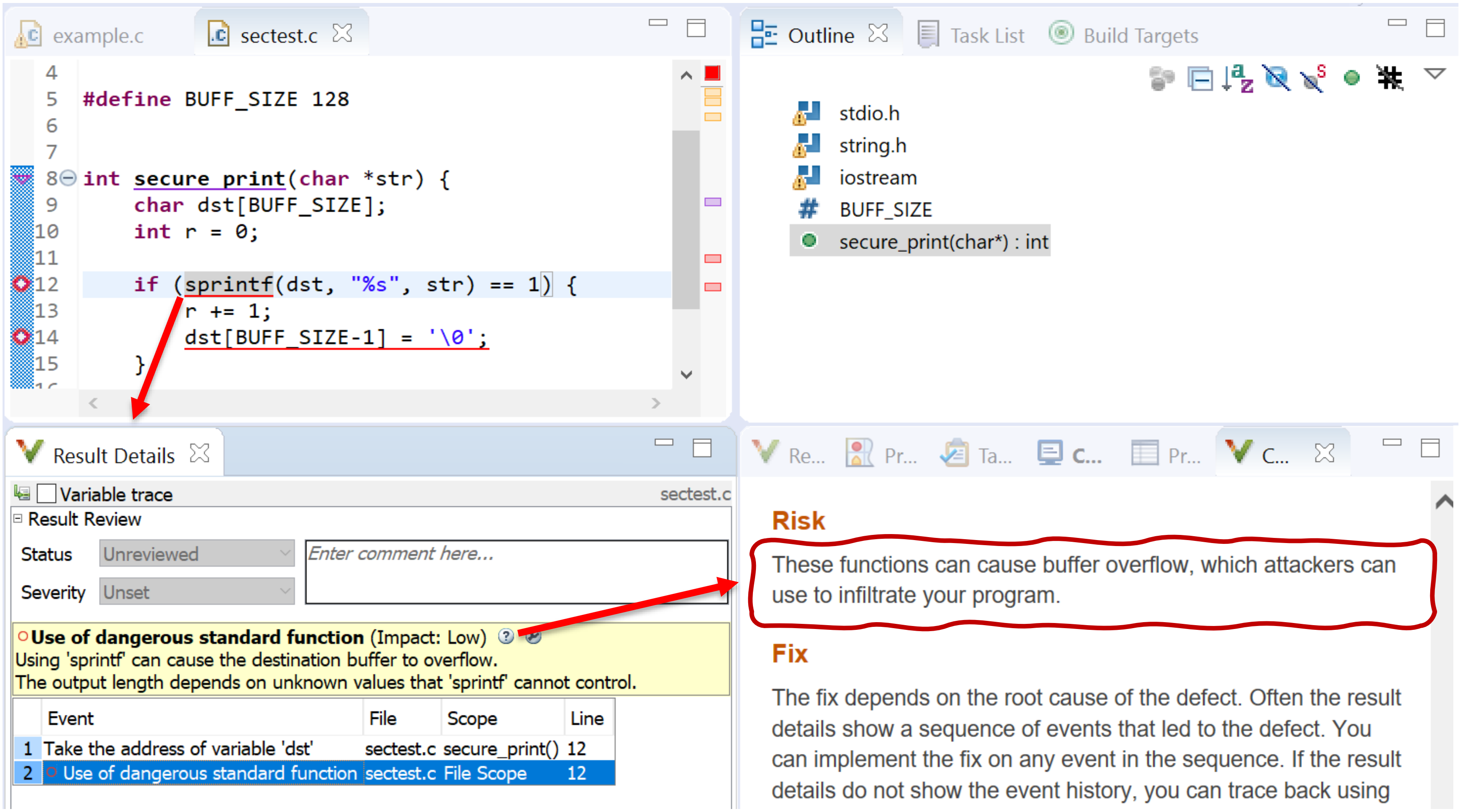
Common Cyber Attack Scenarios



Unknowns + lack of robustness => Anything can happen (beyond spec)

Source: *Embedded Systems Security*, D. Papp et al, IEEE Conf. Sec. Privacy & Trust, 2015.

Follow secure guidelines and practices as you code



```
#define BUFF_SIZE 128


int secure_print(char *str) {
    char dst[BUFF_SIZE];
    int r = 0;

    if (sprintf(dst, "%s", str) == 1) {
        r += 1;
        dst[BUFF_SIZE-1] = '\0';
    }
```

Use of dangerous standard function (Impact: Low)
Using 'sprintf' can cause the destination buffer to overflow.
The output length depends on unknown values that 'sprintf' cannot control.

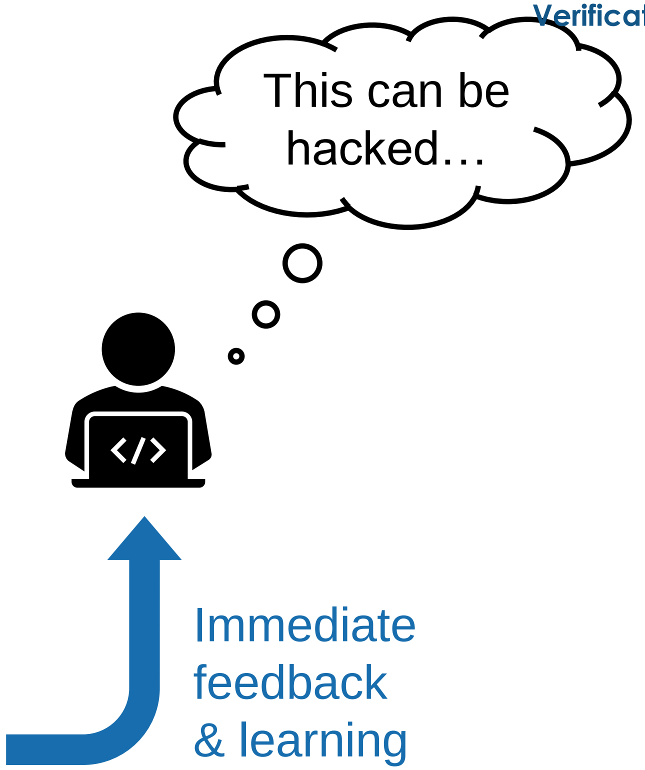
	Event	File	Scope	Line
1	Take the address of variable 'dst'	sectest.c	secure_print()	12
2	Use of dangerous standard function	sectest.c	File Scope	12

Risk

These functions can cause buffer overflow, which attackers can use to infiltrate your program.

Fix

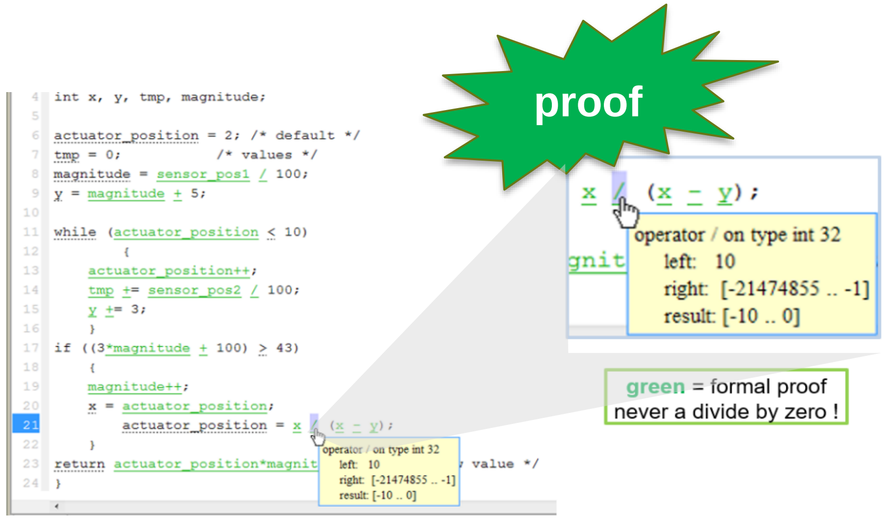
The fix depends on the root cause of the defect. Often the result details show a sequence of events that led to the defect. You can implement the fix on any event in the sequence. If the result details do not show the event history, you can trace back using



Polyspace has 99.4% coverage of secure coding guideline CERT-C(++), identifies common programming errors (CWE) and computes complexity metrics

Robustness “Testing” with Guarantees

- *Sound* static analysis with proof
  - Based on analysis, not execution
  - Requires no test harness
  - Considers all inputs & states
    - Boundary values, race conditions, sufficient checking of user inputs…?



```
int x, y, tmp, magnitude;

actuator_position = 2; /* default */
tmp = 0;               /* values */
magnitude = sensor_pos1 / 100;
y = magnitude + 5;

while (actuator_position < 10)
        {
    actuator_position++;
    tmp += sensor_pos2 / 100;
    y += 3;
    }
if ((3*magnitude + 100) > 43)
    {
    magnitude++;
    x = actuator_position;
        actuator_position = x / (x - y);
    }
return actuator_position*magnit                 value */
}
```

```
operator / on type int 32
   left:   10
   right:  [-21474855 .. -1]
   result: [-10 .. 0]
```

Fast, misses no bugs and automatic

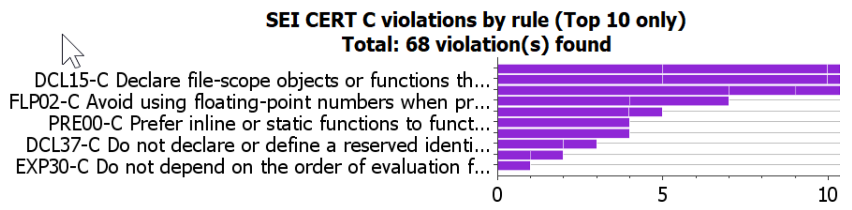
Proving Absence of Critical Defects & Vulnerabilities

- Assert
- Buffer overrun
- Divide by zero
- Uninitialised variable
- Unreachable code
- Stack Usage
- Data Flow
- Numerical
- Concurrent access
- Etc..

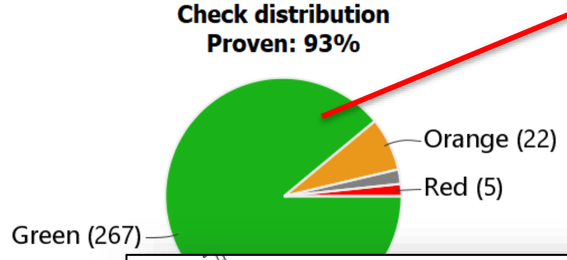
Static Application Security Testing (SAST)

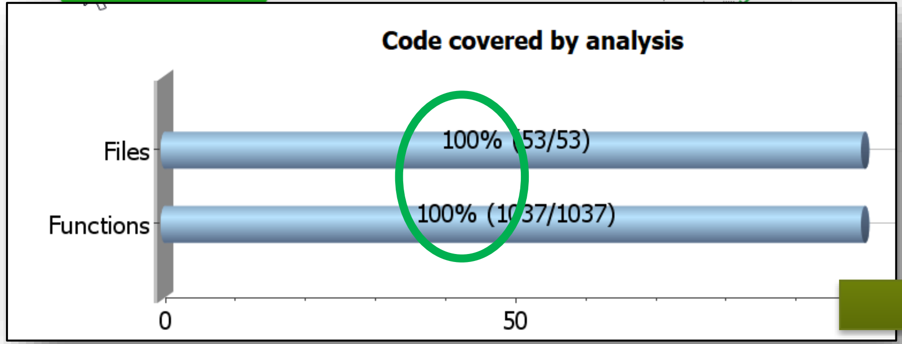
Polyspace Code Verifier

Enforce secure coding rules & best practices

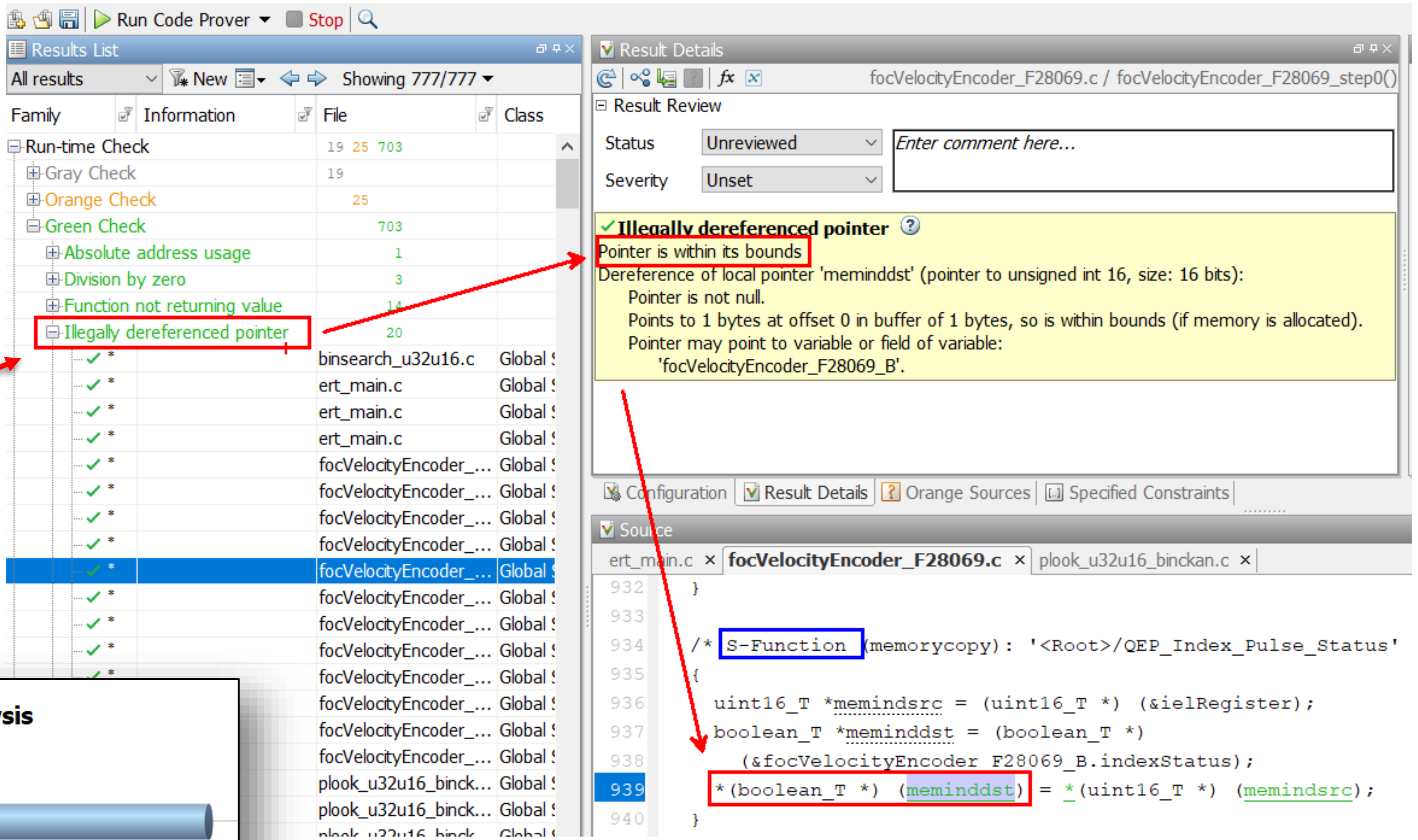


Analysis information: Configuration - Unreachable functions - Analysis





Prove absence of vulnerabilities



Family	Information	File	Class
Run-time Check		19 25 703	
Gray Check		19	
Orange Check		25	
Green Check		703	
Absolute address usage		1	
Division by zero		3	
Function not returning value		14	
Illegally dereferenced pointer		20	
✓		binsearch_u32u16.c	Global
✓		ert_main.c	Global
✓		ert_main.c	Global
✓		ert_main.c	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		focVelocityEncoder_...	Global
✓		plook_u32u16_binck...	Global
✓		plook_u32u16_binck...	Global

focVelocityEncoder_F28069.c / focVelocityEncoder_F28069_step0()

Result Review

Status: Unreviewed
Severity: Unset
Enter comment here...

✓ Illegally dereferenced pointer

Pointer is within its bounds

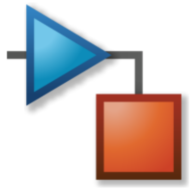
Dereference of local pointer 'meminddst' (pointer to unsigned int 16, size: 16 bits):
Pointer is not null.
Points to 1 bytes at offset 0 in buffer of 1 bytes, so is within bounds (if memory is allocated).
Pointer may point to variable or field of variable:
'focVelocityEncoder_F28069_B'.

```
  }

  /* S-Function (memorycopy): '<Root>/QEP_Index_Pulse_Status'
  {
    uint16_T *memindsrc = (uint16_T *) (&ielRegister);
    boolean_T *meminddst = (boolean_T *)
      (&focVelocityEncoder_F28069_B.indexStatus);
    *(boolean_T *) (meminddst) = *(uint16_T *) (memindsrc);
  }
```

Considers *all* inputs & *all* program states

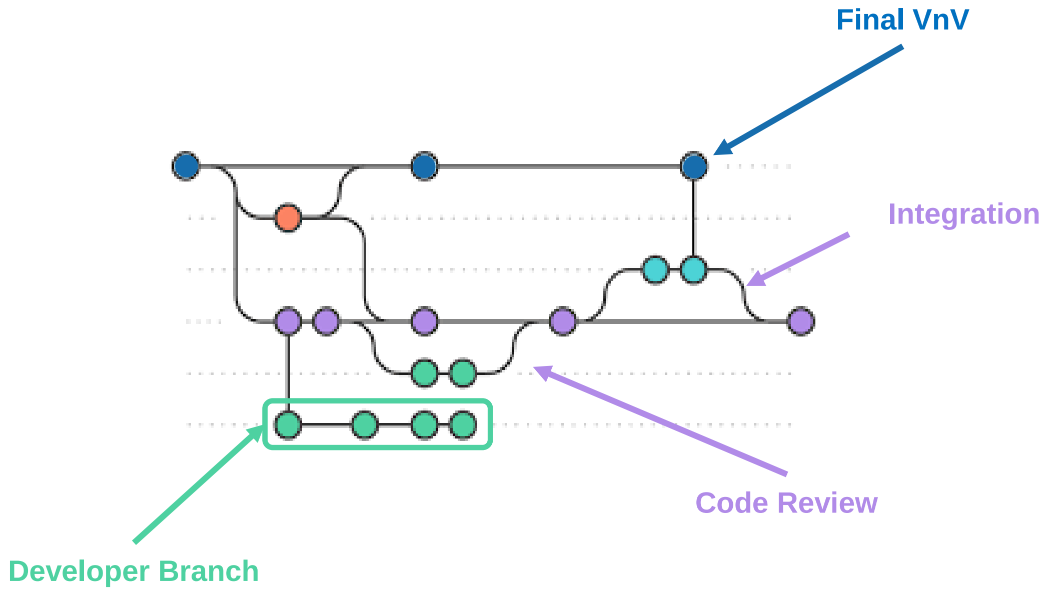
When to use Polyspace Product?

Embedded Software Development

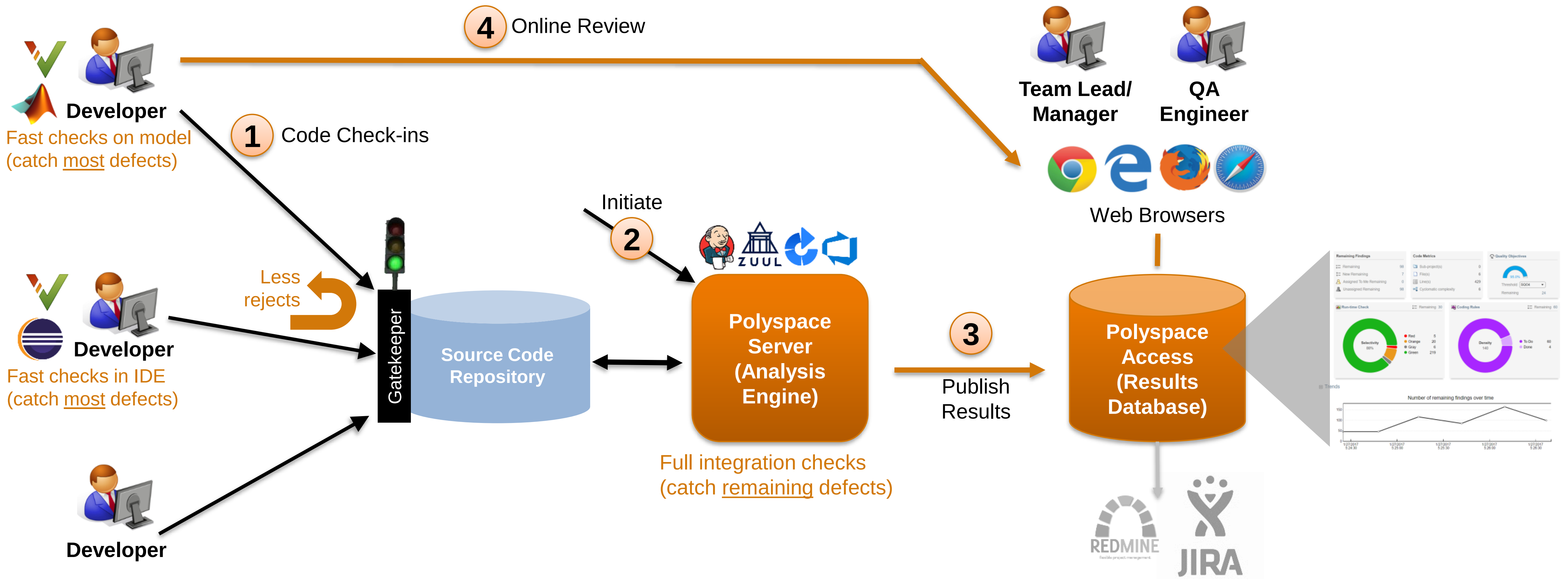


Generated code from high-level modeling language

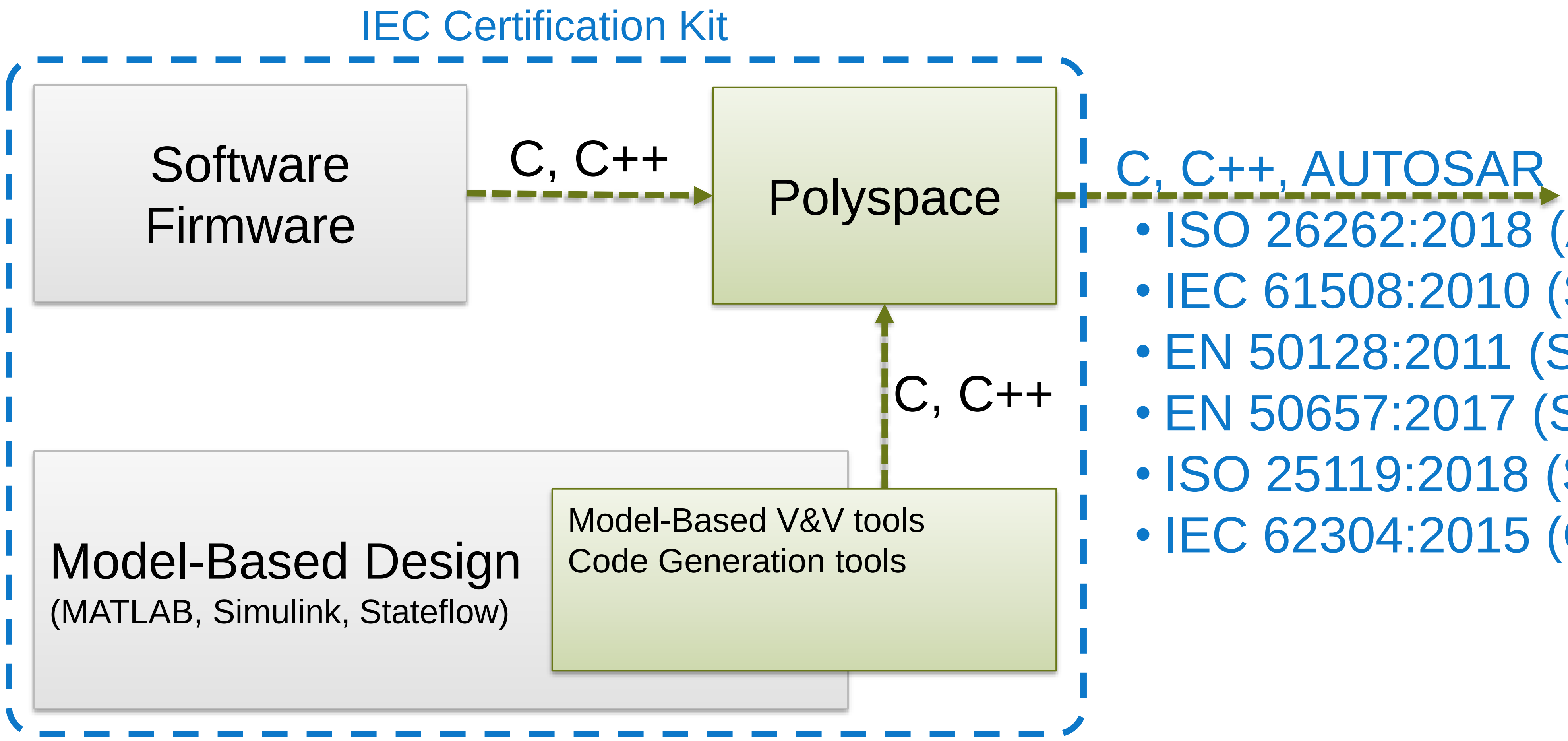
Integrated during the entire SDLC



DevOps Workflow for Code Analysis

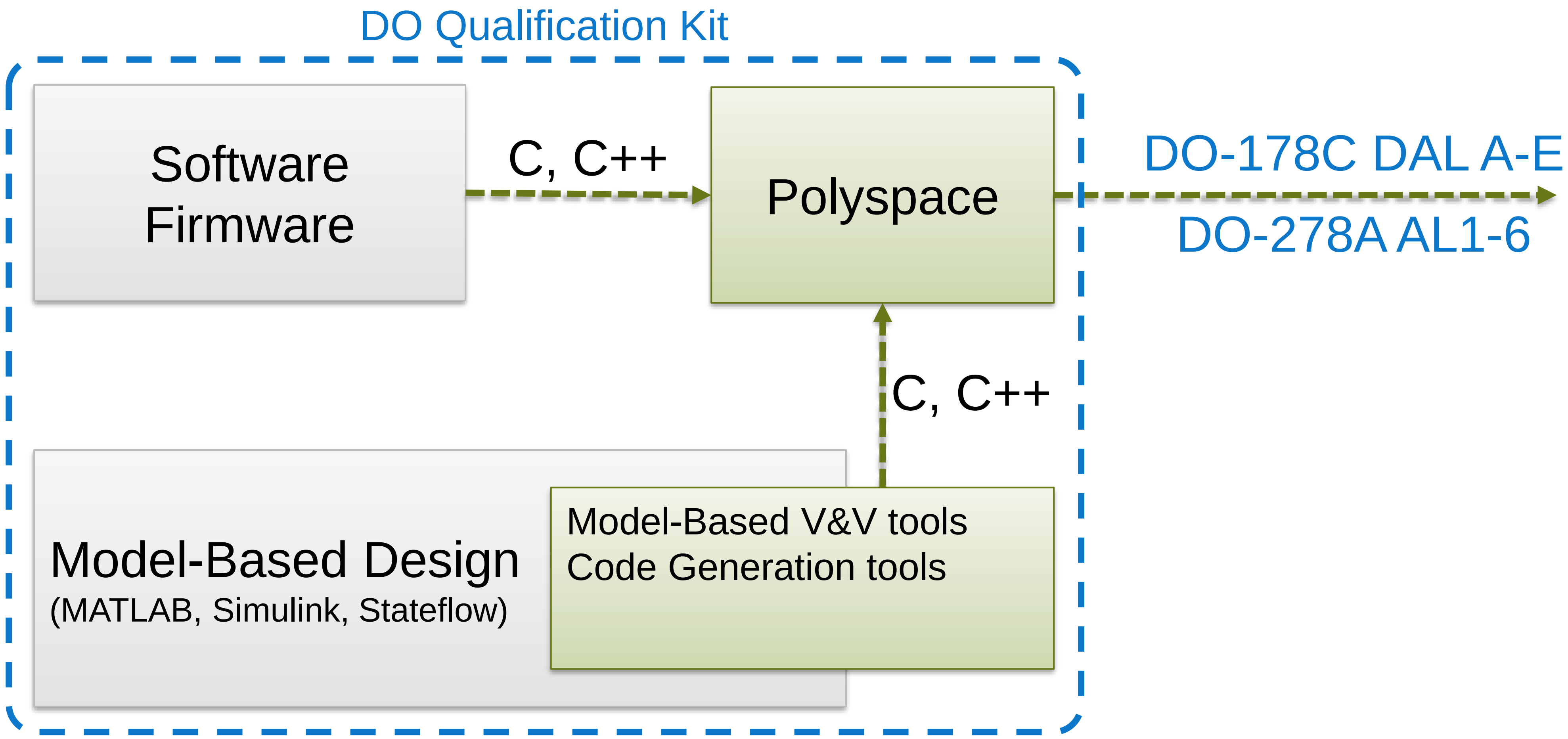

4 Online Review
Developer
Fast checks on model (catch most defects)
1 Code Check-ins
Developer
Fast checks in IDE (catch most defects)
Less rejects
Developer
Gatekeeper
Source Code Repository
Initiate
2
ZUUL
Polyspace Server (Analysis Engine)
Full integration checks (catch remaining defects)
3
Publish Results
Team Lead/ Manager
QA Engineer
Web Browsers
Polyspace Access (Results Database)
REDMINE
JIRA

Compliance to Industry Standards-IEC Certification Kit



- ISO 26262:2018 (ASIL A-D)
- IEC 61508:2010 (SIL 1-4)
- EN 50128:2011 (SIL 0-4)
- EN 50657:2017 (SIL 0-4)
- ISO 25119:2018 (SRL B,1-3)
- IEC 62304:2015 (Class A-C)

Compliance to Industry Standards- DO Qualification Kit



DO Qualification Kit Overview

Supported Standards

DO-178C D.A.L A-E	DO-278A A.L. 1-6
Airborne Software	Ground-based and Space-based Software

Supported Supplements

DO-331	DO-332	DO-333
Model-Based	Object-Oriented	Formal Methods

DO-178C Source Code Considerations

→ Reduce manual code inspection

Source Code Verification per 6.3.4	
Source code is verifiable	✓
Source code conforms to standards	✓
Source code is accurate and consistent	✓

✓ **Indicates item is covered by Polyspace**

DO-178C Robustness Verification Considerations (High/Low Level)

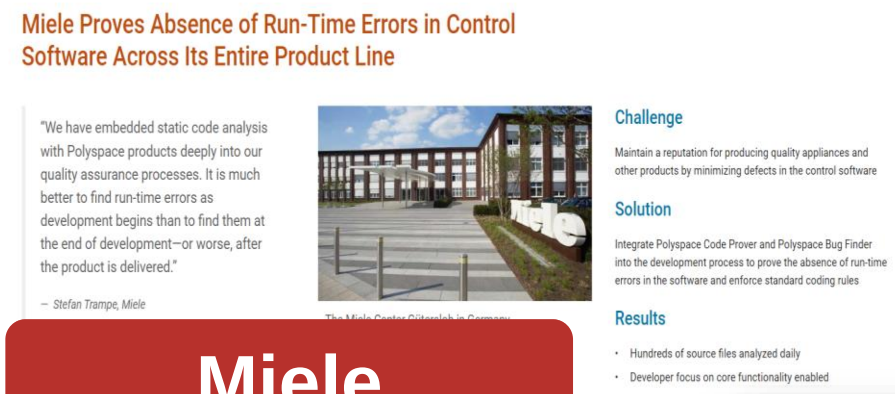
→ *Reduce robustness testing*

Abnormal Inputs and Conditions per FM.6.7.b	
Real and integer variables	O ✓
System initialization during abnormal conditions	O
Possible failure modes of incoming data	O
Loops with computed loop variables	O ✓
Protection mechanisms for exceeding frame times	O
Time-related function overflows	O ✓
State transitions not allowed by requirements	O

Verification of Software Integrity per FM.6.7.c	
Incorrect initialization of variables and constants	O ✓
Parameter passing errors	O ✓
Data corruption, especially global	O ✓
Inadequate end-to-end numerical resolution	O ✓
Incorrect sequencing of events and operations	O ✓

O ✓ **Indicates item is covered by Polyspace Code Prover**

Polyspace used across Industries(remake)



Miele Proves Absence of Run-Time Errors in Control Software Across Its Entire Product Line

"We have embedded static code analysis with Polyspace products deeply into our quality assurance processes. It is much better to find run-time errors as development begins than to find them at the end of development—or worse, after the product is delivered."

— Stefan Trampe, Miele

The Miele Center Gütersloh in Germany.

Challenge

Maintain a reputation for producing quality appliances and other products by minimizing defects in the control software

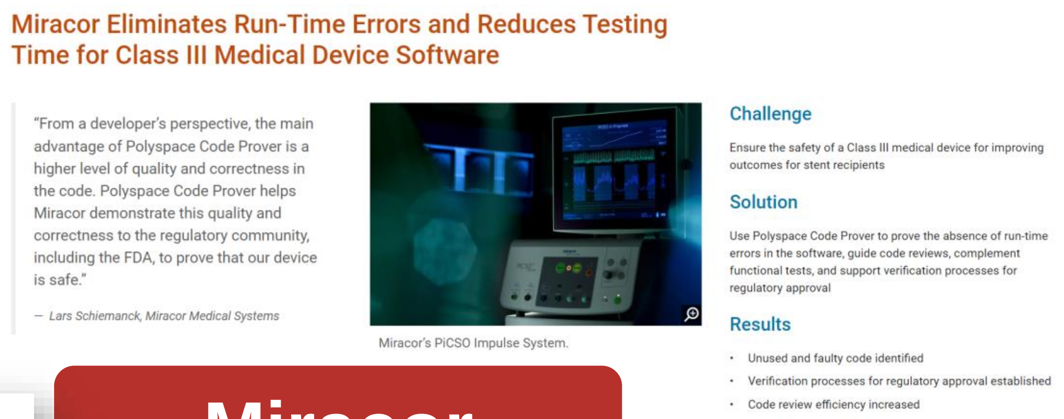
Solution

Integrate Polyspace Code Prover and Polyspace Bug Finder into the development process to prove the absence of run-time errors in the software and enforce standard coding rules

Results

- Hundreds of source files analyzed daily
- Developer focus on core functionality enabled

Miele



Miracor Eliminates Run-Time Errors and Reduces Testing Time for Class III Medical Device Software

"From a developer's perspective, the main advantage of Polyspace Code Prover is a higher level of quality and correctness in the code. Polyspace Code Prover helps Miracor demonstrate this quality and correctness to the regulatory community, including the FDA, to prove that our device is safe."

— Lars Schiemanck, Miracor Medical Systems

Miracor's PiCSO Impulse System.

Challenge

Ensure the safety of a Class III medical device for improving outcomes for stent recipients

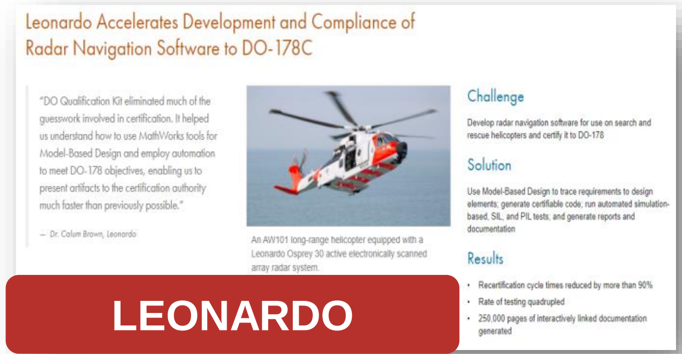
Solution

Use Polyspace Code Prover to prove the absence of run-time errors in the software, guide code reviews, complement functional tests, and support verification processes for regulatory approval

Results

- Unused and faulty code identified
- Verification processes for regulatory approval established
- Code review efficiency increased

Miracor



Leonardo Accelerates Development and Compliance of Radar Navigation Software to DO-178C

"DO Qualification Kit eliminated much of the guesswork involved in certification. It helped us understand how to use MathWorks tools for Model-Based Design and employ automation to meet DO-178 objectives, enabling us to present artifacts to the certification authority much faster than previously possible."

— Dr. Colum Brown, Leonardo

An AW101 long-range helicopter equipped with a Leonardo Osprey 30 active electronically scanned array radar system.

Challenge

Develop radar navigation software for use on search and rescue helicopters and certify it to DO-178

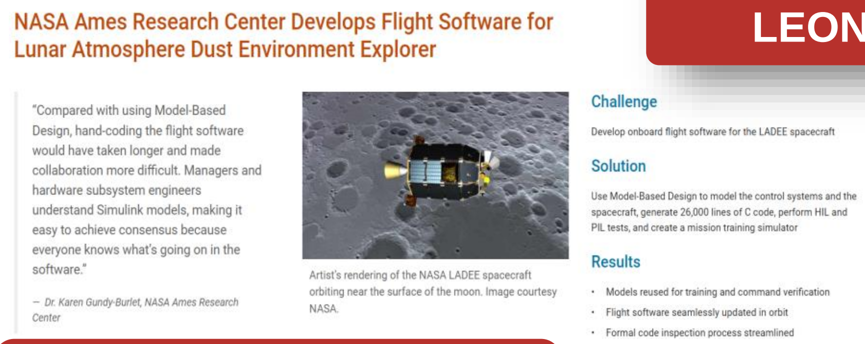
Solution

Use Model-Based Design to trace requirements to design elements; generate certifiable code; run automated simulation-based, SIL, and PIL tests; and generate reports and documentation

Results

- Recertification cycle times reduced by more than 90%
- Rate of testing quadrupled
- 250,000 pages of interactively linked documentation generated

LEONARDO



NASA Ames Research Center Develops Flight Software for Lunar Atmosphere Dust Environment Explorer

"Compared with using Model-Based Design, hand-coding the flight software would have taken longer and made collaboration more difficult. Managers and hardware subsystem engineers understand Simulink models, making it easy to achieve consensus because everyone knows what's going on in the software."

— Dr. Karen Gundy-Burlet, NASA Ames Research Center

Artist's rendering of the NASA LADEE spacecraft orbiting near the surface of the moon. Image courtesy NASA.

Challenge

Develop onboard flight software for the LADEE spacecraft

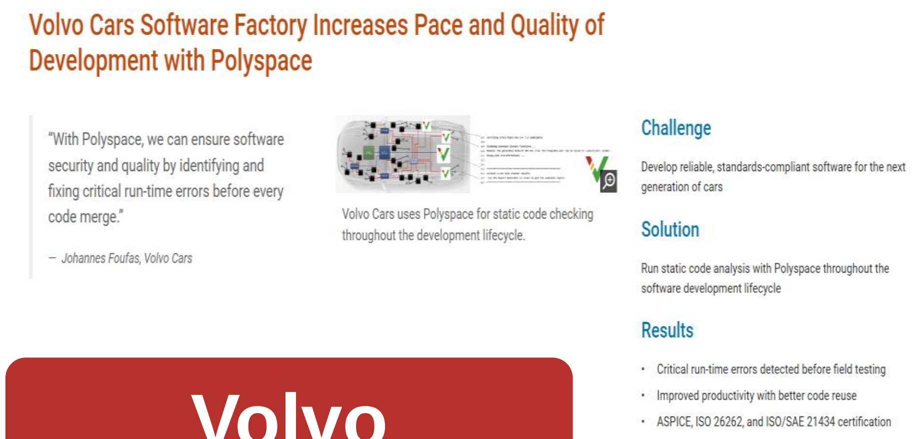
Solution

Use Model-Based Design to model the control systems and the spacecraft, generate 26,000 lines of C code, perform HIL and PIL tests, and create a mission training simulator

Results

- Models reused for training and command verification
- Flight software seamlessly updated in orbit
- Formal code inspection process streamlined

NASA



Volvo Cars Software Factory Increases Pace and Quality of Development with Polyspace

"With Polyspace, we can ensure software security and quality by identifying and fixing critical run-time errors before every code merge."

— Johannes Foufas, Volvo Cars

Volvo Cars uses Polyspace for static code checking throughout the development lifecycle.

Challenge

Develop reliable, standards-compliant software for the next generation of cars

Solution

Run static code analysis with Polyspace throughout the software development lifecycle

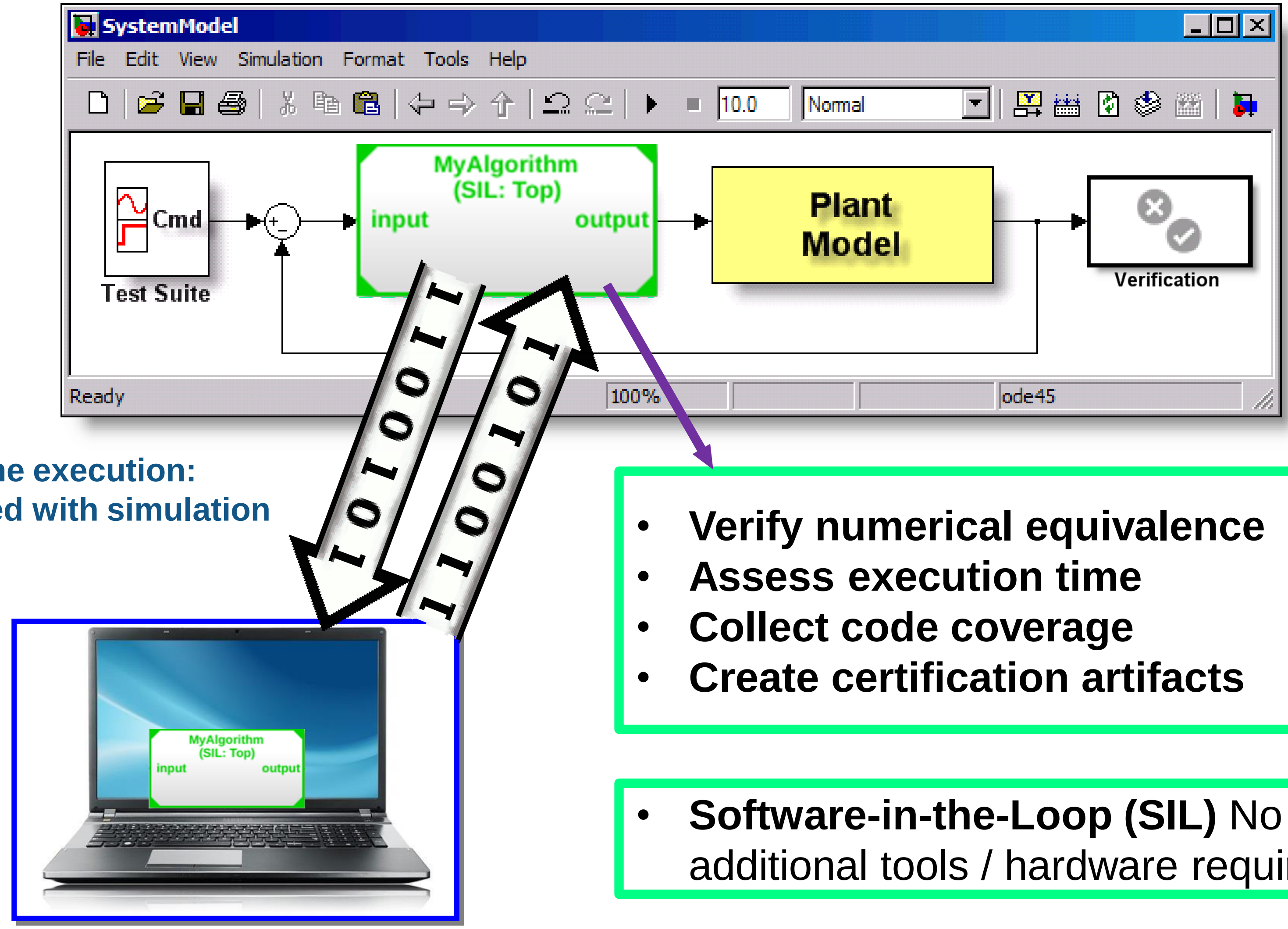
Results

- Critical run-time errors detected before field testing
- Improved productivity with better code reuse
- ASPICE, ISO 26262, and ISO/SAE 21434 certification

Volvo

Software-in-the-Loop (SIL)

Verify compiled object code matches simulation



Non-real-time execution: synchronized with simulation

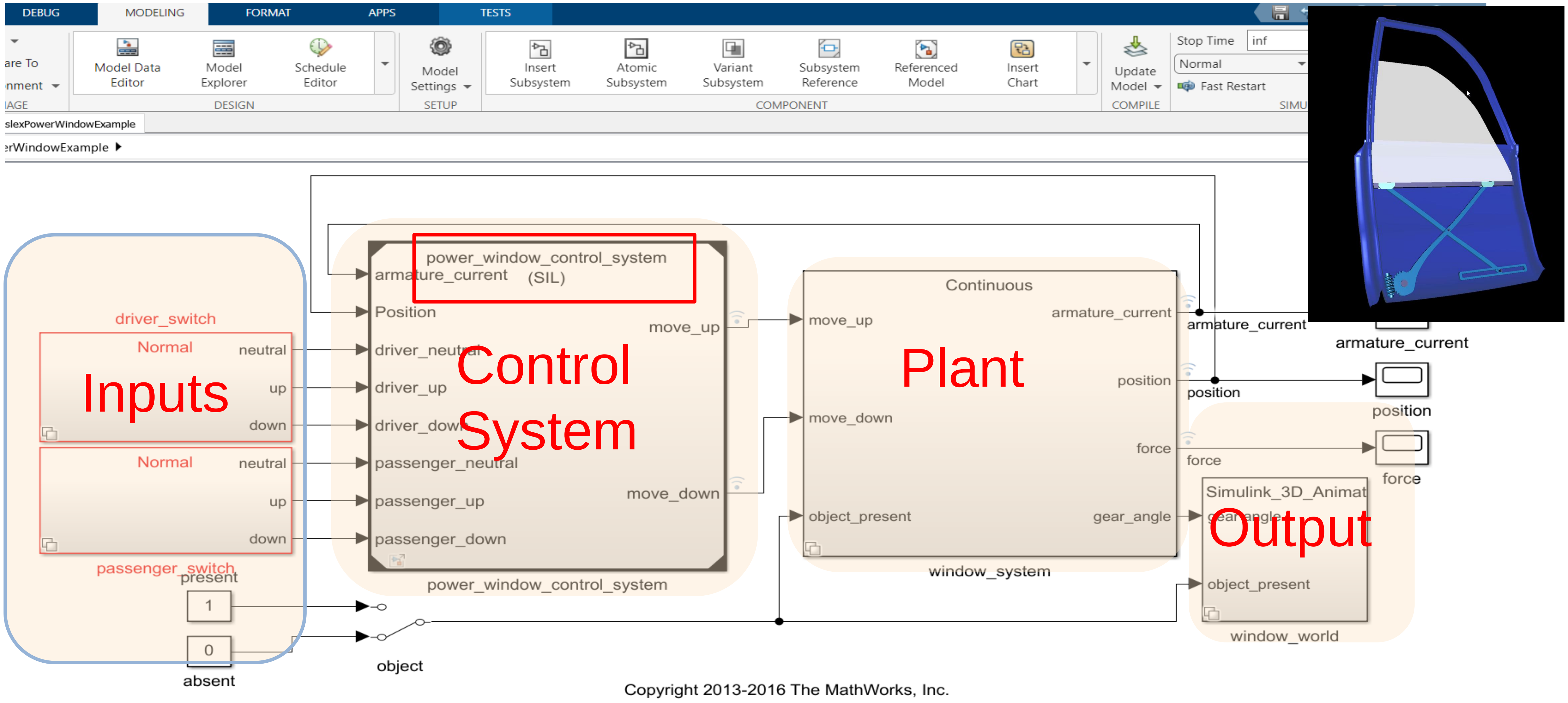
- **Verify numerical equivalence**
- **Assess execution time**
- **Collect code coverage**
- **Create certification artifacts**

- **Software-in-the-Loop (SIL)** No additional tools / hardware required

MATLAB EXPO

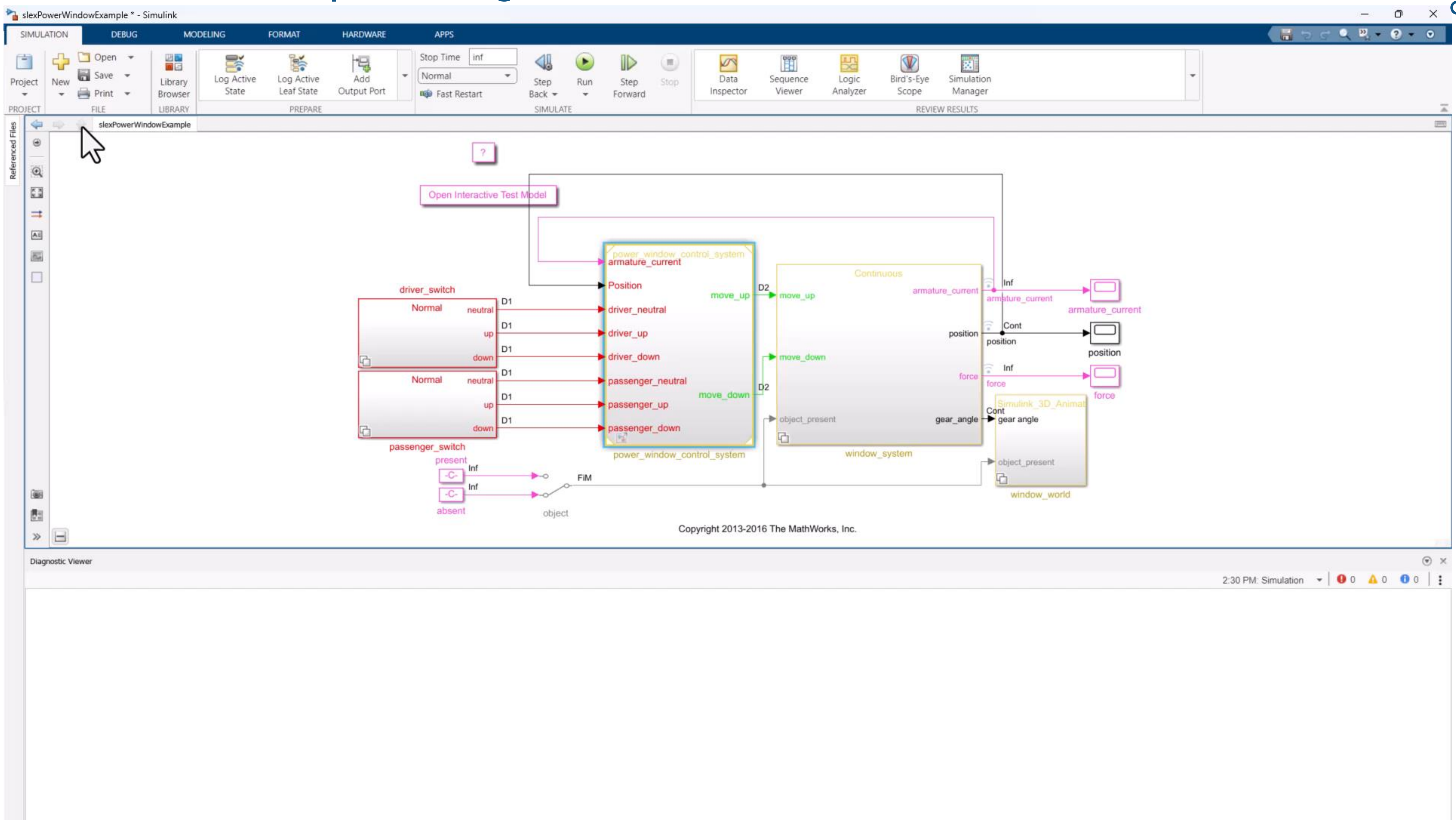
Control Software testing: Software-in-the-Loop (SIL)

Code Gen & Verification



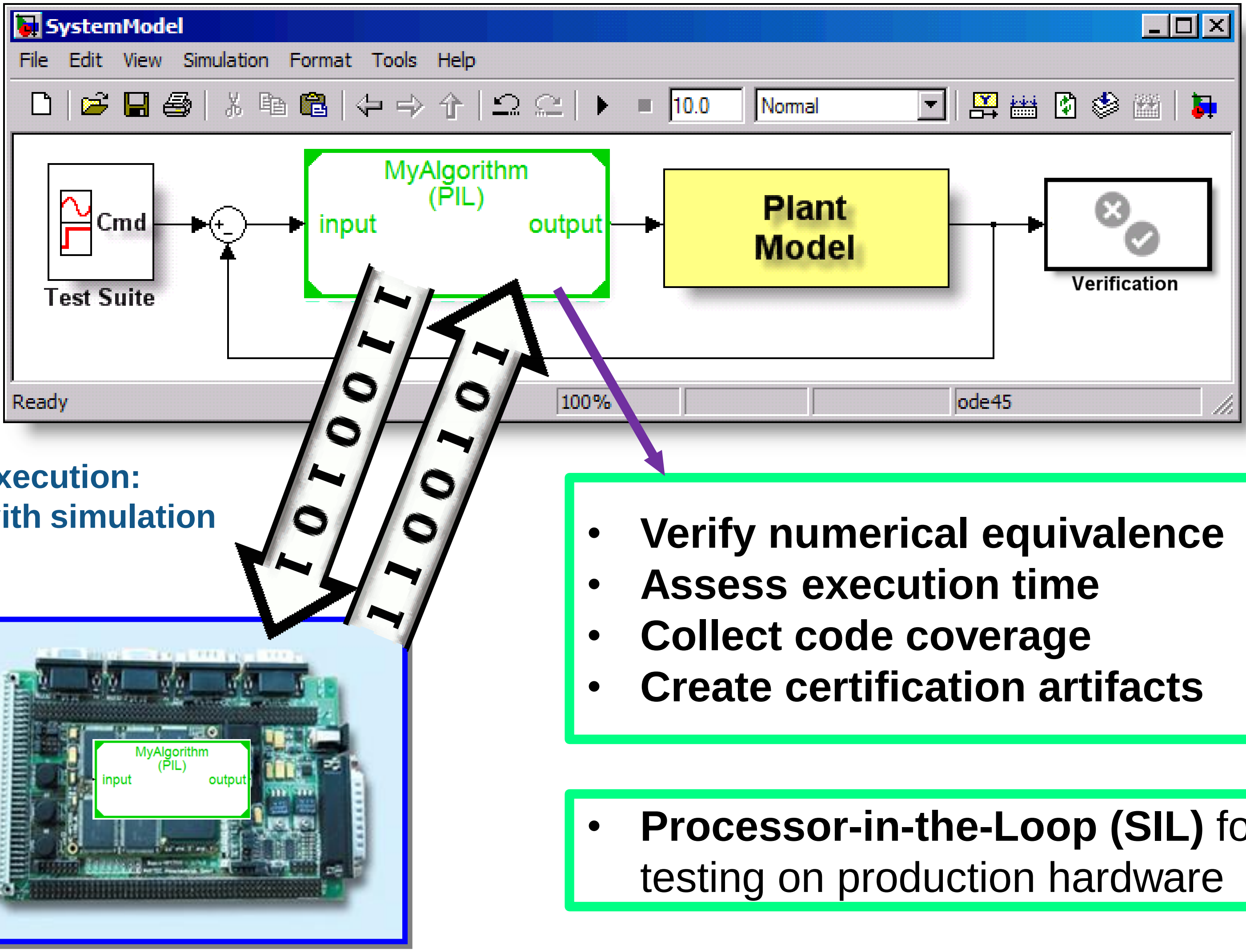
Software-In-Loop Testing:

Code Gen & Verification



Processor-in-the-Loop (PIL)

Verify compiled object code matches simulation

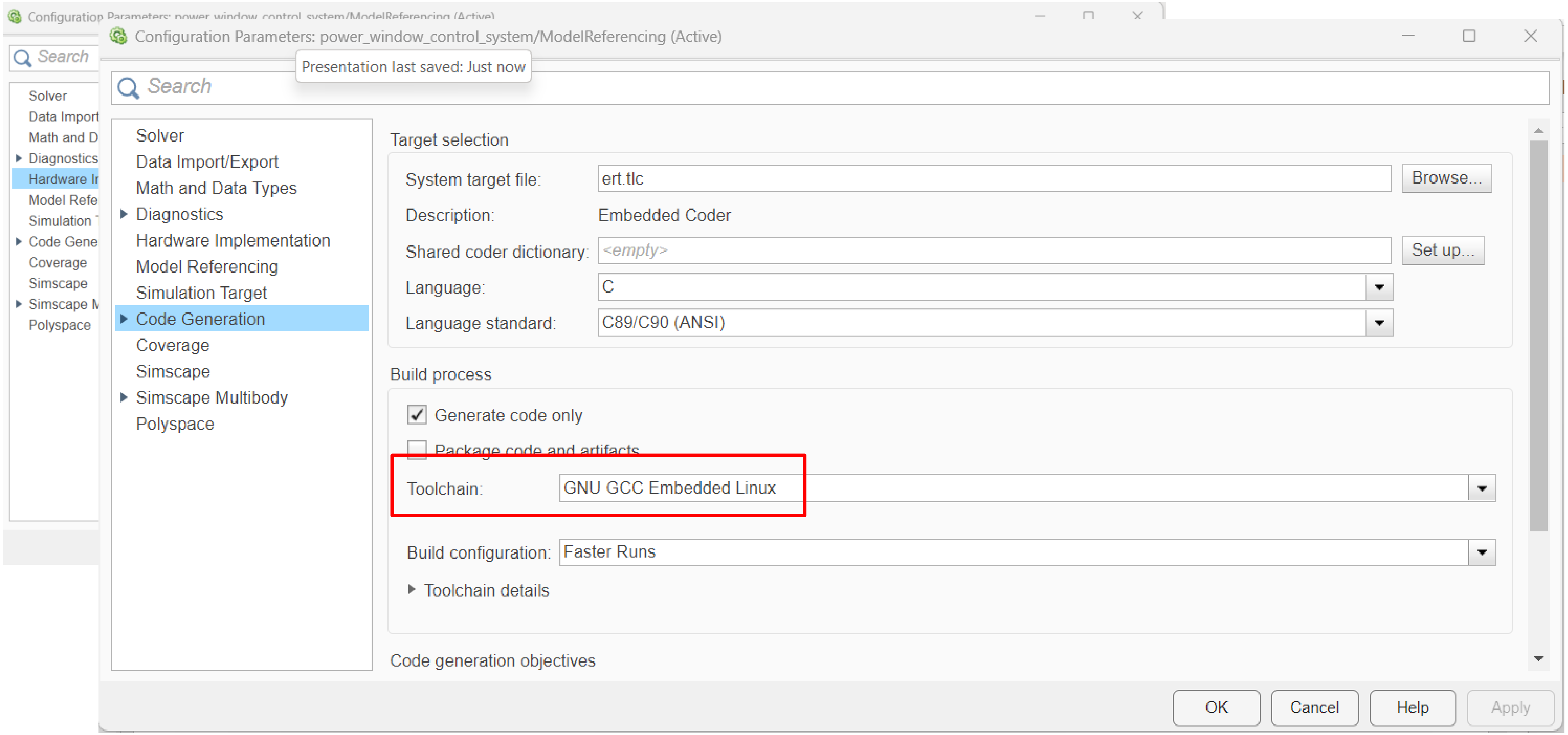


Non-real-time execution: synchronized with simulation

- **Verify numerical equivalence**
- **Assess execution time**
- **Collect code coverage**
- **Create certification artifacts**

- **Processor-in-the-Loop (SIL)** for testing on production hardware

Generate processor executables:



Configuration Parameters: power_window_control_system/ModelReferencing (Active)

Presentation last saved: Just now

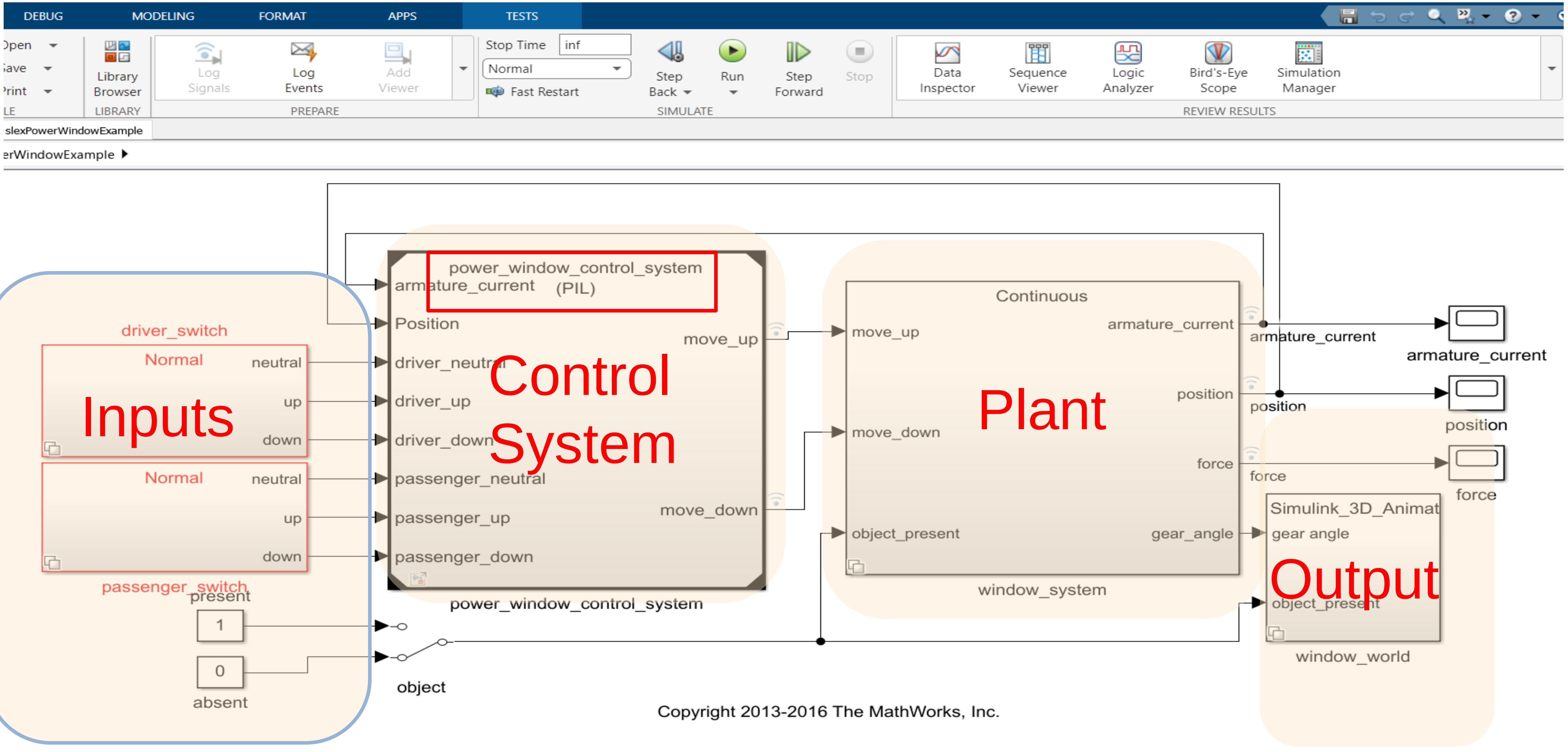
Target selection

System target file: ert.tlc
Description: Embedded Coder
Shared coder dictionary: <empty>
Language: C
Language standard: C89/C90 (ANSI)

Build process

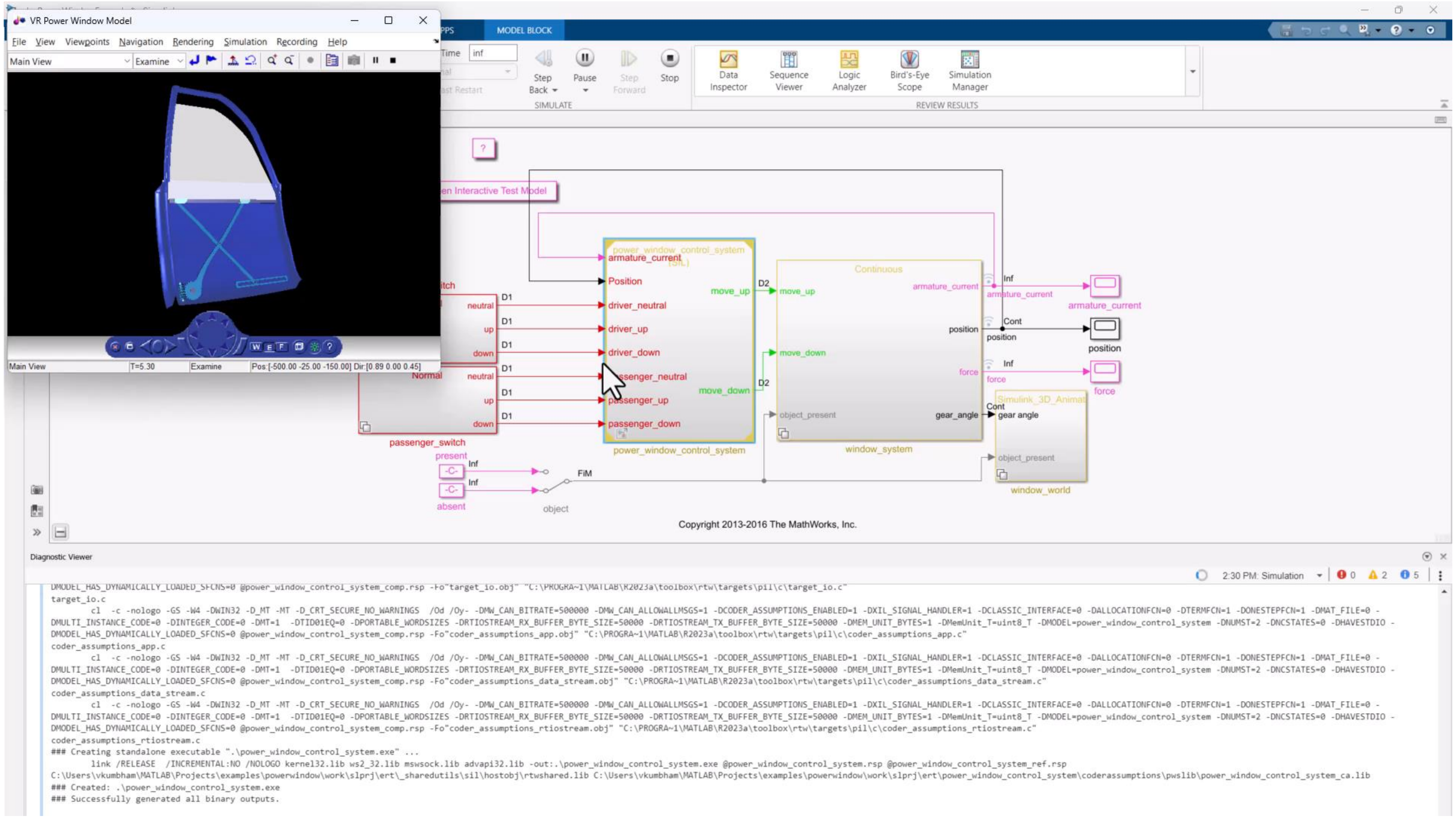
☑ Generate code only
☐ Package code and artifacts
Toolchain: GNU GCC Embedded Linux
Build configuration: Faster Runs

Control Software testing: Processor-in-the-Loop (PIL)



Code Gen & Verification

Control Software testing: Processor-in-the-Loop (PIL)



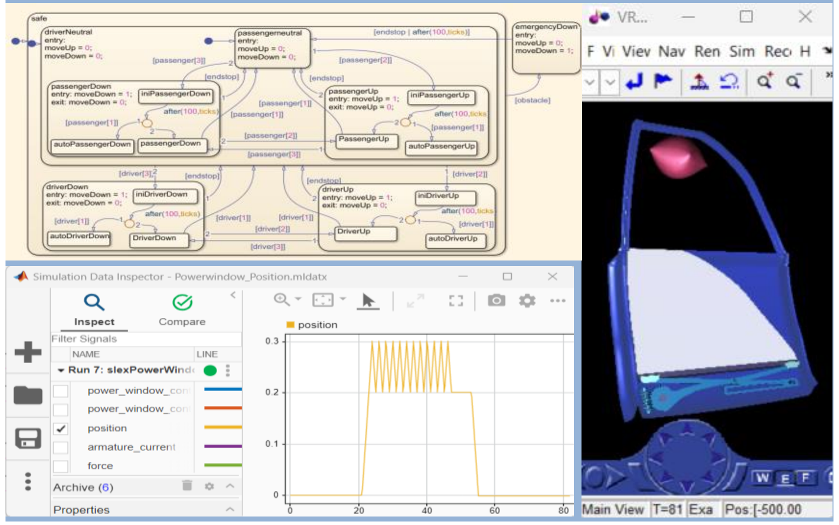
```
DMODEL_HAS_DYNAMICALLY_LOADED_SFCNS=0 @power_window_control_system_comp.rsp -Fo"target_io.obj" "C:\PROGRA~1\MATLAB\R2023a\toolbox\rtw\targets\pil\c\target_io.c"
target_io.c
        cl  -c -nologo -GS -W4 -DWIN32 -D_MT -MT -D_CRT_SECURE_NO_WARNINGS  /Od /Oy- -DMW_CAN_BITRATE=500000 -DMW_CAN_ALLOWALLMSGS=1 -DCODER_ASSUMPTIONS_ENABLED=1 -DXIL_SIGNAL_HANDLER=1 -DCLASSIC_INTERFACE=0 -DALLOCATIONFCN=0 -DTERMFCN=1 -DONESTEPFCN=1 -DMAT_FILE=0 -
DMULTI_INSTANCE_CODE=0 -DINTEGER_CODE=0 -DMT=1  -DTID01EQ=0 -DPORTABLE_WORDSIZES -DRTIOSTREAM_RX_BUFFER_BYTE_SIZE=50000 -DRTIOSTREAM_TX_BUFFER_BYTE_SIZE=50000 -DMEM_UNIT_BYTES=1 -DMemUnit_T=uint8_T -DMODEL=power_window_control_system -DNUMST=2 -DNCSTATES=0 -DHAVESTDIO -
DMODEL_HAS_DYNAMICALLY_LOADED_SFCNS=0 @power_window_control_system_comp.rsp -Fo"coder_assumptions_app.obj" "C:\PROGRA~1\MATLAB\R2023a\toolbox\rtw\targets\pil\c\coder_assumptions_app.c"
coder_assumptions_app.c
        cl  -c -nologo -GS -W4 -DWIN32 -D_MT -MT -D_CRT_SECURE_NO_WARNINGS  /Od /Oy- -DMW_CAN_BITRATE=500000 -DMW_CAN_ALLOWALLMSGS=1 -DCODER_ASSUMPTIONS_ENABLED=1 -DXIL_SIGNAL_HANDLER=1 -DCLASSIC_INTERFACE=0 -DALLOCATIONFCN=0 -DTERMFCN=1 -DONESTEPFCN=1 -DMAT_FILE=0 -
DMULTI_INSTANCE_CODE=0 -DINTEGER_CODE=0 -DMT=1  -DTID01EQ=0 -DPORTABLE_WORDSIZES -DRTIOSTREAM_RX_BUFFER_BYTE_SIZE=50000 -DRTIOSTREAM_TX_BUFFER_BYTE_SIZE=50000 -DMEM_UNIT_BYTES=1 -DMemUnit_T=uint8_T -DMODEL=power_window_control_system -DNUMST=2 -DNCSTATES=0 -DHAVESTDIO -
DMODEL_HAS_DYNAMICALLY_LOADED_SFCNS=0 @power_window_control_system_comp.rsp -Fo"coder_assumptions_data_stream.obj" "C:\PROGRA~1\MATLAB\R2023a\toolbox\rtw\targets\pil\c\coder_assumptions_data_stream.c"
coder_assumptions_data_stream.c
        cl  -c -nologo -GS -W4 -DWIN32 -D_MT -MT -D_CRT_SECURE_NO_WARNINGS  /Od /Oy- -DMW_CAN_BITRATE=500000 -DMW_CAN_ALLOWALLMSGS=1 -DCODER_ASSUMPTIONS_ENABLED=1 -DXIL_SIGNAL_HANDLER=1 -DCLASSIC_INTERFACE=0 -DALLOCATIONFCN=0 -DTERMFCN=1 -DONESTEPFCN=1 -DMAT_FILE=0 -
DMULTI_INSTANCE_CODE=0 -DINTEGER_CODE=0 -DMT=1  -DTID01EQ=0 -DPORTABLE_WORDSIZES -DRTIOSTREAM_RX_BUFFER_BYTE_SIZE=50000 -DRTIOSTREAM_TX_BUFFER_BYTE_SIZE=50000 -DMEM_UNIT_BYTES=1 -DMemUnit_T=uint8_T -DMODEL=power_window_control_system -DNUMST=2 -DNCSTATES=0 -DHAVESTDIO -
DMODEL_HAS_DYNAMICALLY_LOADED_SFCNS=0 @power_window_control_system_comp.rsp -Fo"coder_assumptions_rtiostream.obj" "C:\PROGRA~1\MATLAB\R2023a\toolbox\rtw\targets\pil\c\coder_assumptions_rtiostream.c"
coder_assumptions_rtiostream.c
### Creating standalone executable ".\power_window_control_system.exe" ...
        link /RELEASE  /INCREMENTAL:NO /NOLOGO kernel32.lib ws2_32.lib mswsock.lib advapi32.lib -out:.\power_window_control_system.exe @power_window_control_system.rsp @power_window_control_system_ref.rsp
C:\Users\vkumbham\MATLAB\Projects\examples\powerwindow\work\slprj\ert\_sharedutils\sil\hostobj\rtwshared.lib C:\Users\vkumbham\MATLAB\Projects\examples\powerwindow\work\slprj\ert\power_window_control_system\coderassumptions\pwslib\power_window_control_system_ca.lib
### Created: .\power_window_control_system.exe
### Successfully generated all binary outputs.
```

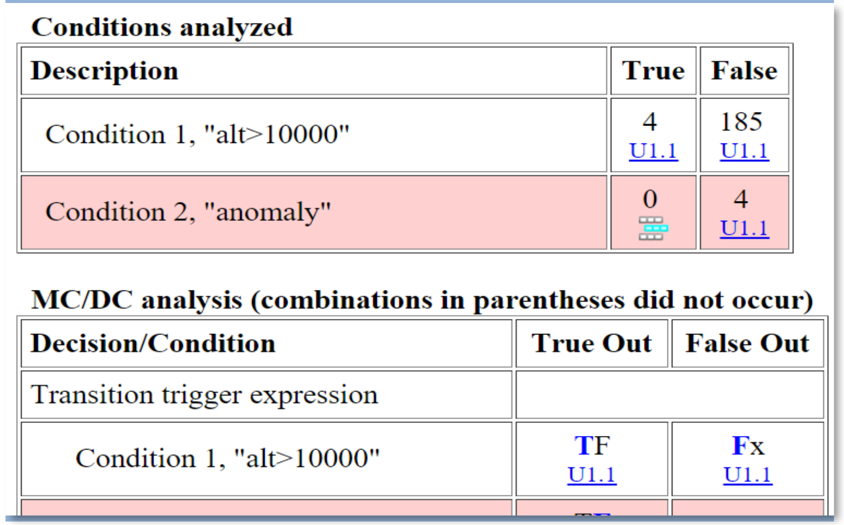
There are three key pieces to Model-Based Design

✓ Modeling & Simulation

✓ Test & Validation

✓ Code Generation & Code Verification



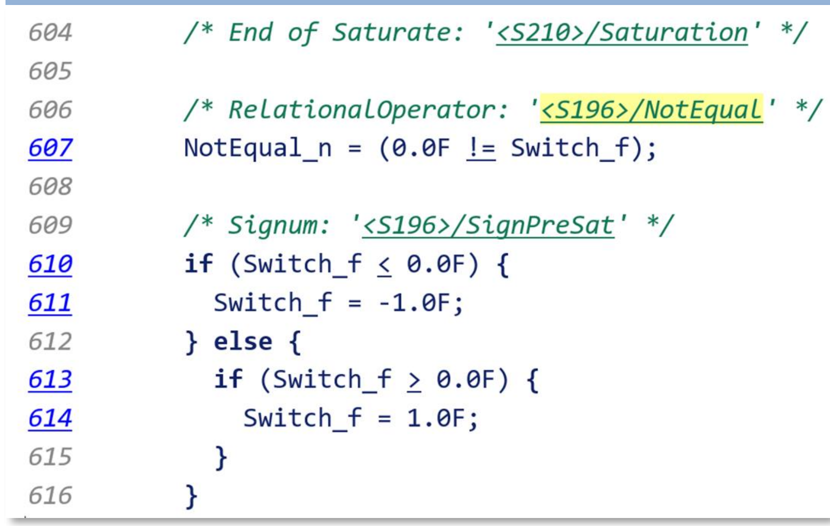


Conditions analyzed

Description	True	False
Condition 1, "alt>10000"	4 U1.1	185 U1.1
Condition 2, "anomaly"	0	4 U1.1

MC/DC analysis (combinations in parentheses did not occur)

Decision/Condition	True Out	False Out
Transition trigger expression		
Condition 1, "alt>10000"	TF U1.1	Fx U1.1



```
      /* End of Saturate: '<S210>/Saturation' */

      /* RelationalOperator: '<S196>/NotEqual' */
      NotEqual_n = (0.0F != Switch_f);

      /* Signum: '<S196>/SignPreSat' */
      if (Switch_f < 0.0F) {
        Switch_f = -1.0F;
      } else {
        if (Switch_f > 0.0F) {
          Switch_f = 1.0F;
        }
      }
```

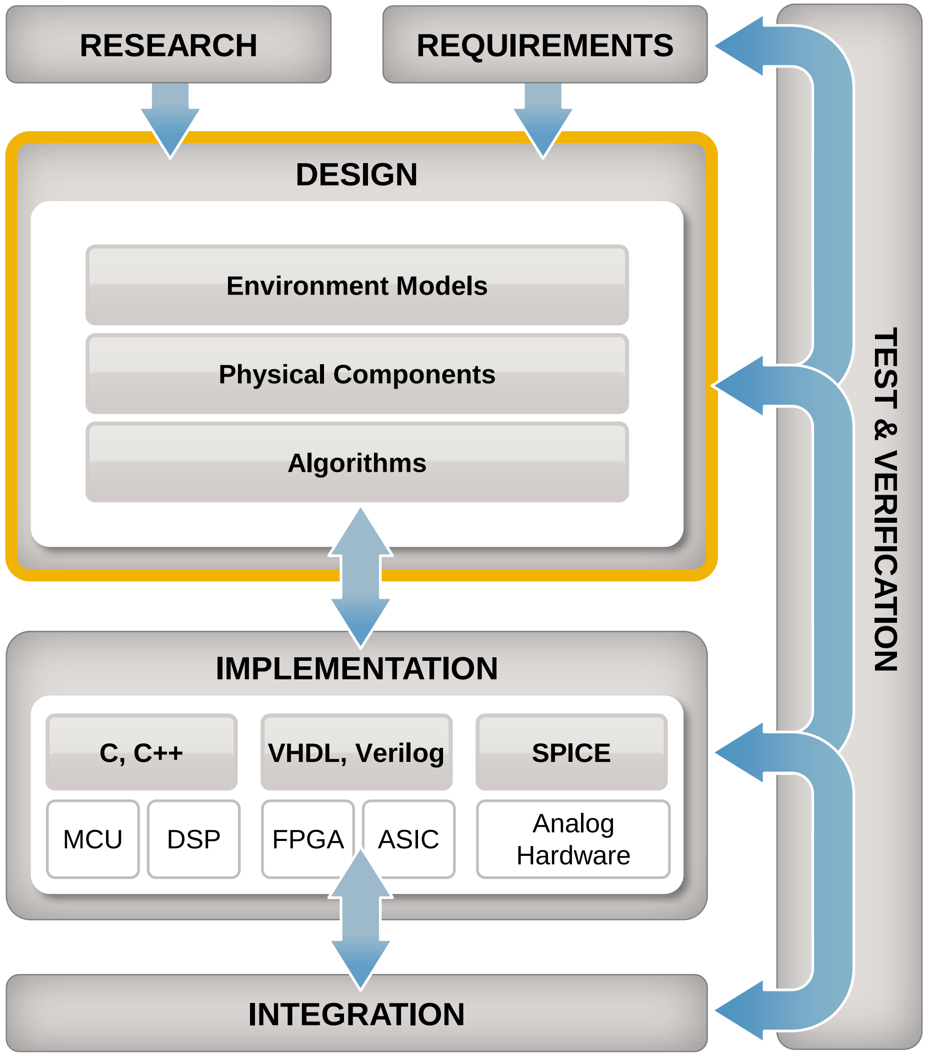
SIMULINK®

Simulation and Model-Based Design

Quantifiable benefits of Model-Based Design

Model-Based Design enabled Continental to verify our design in-vehicle earlier, **eliminating six months of hardware development** and one prototype build. **Verification time was cut by up to 50 percent. 90 percent of application automatically coded.**
Thomas Ehl, Continental

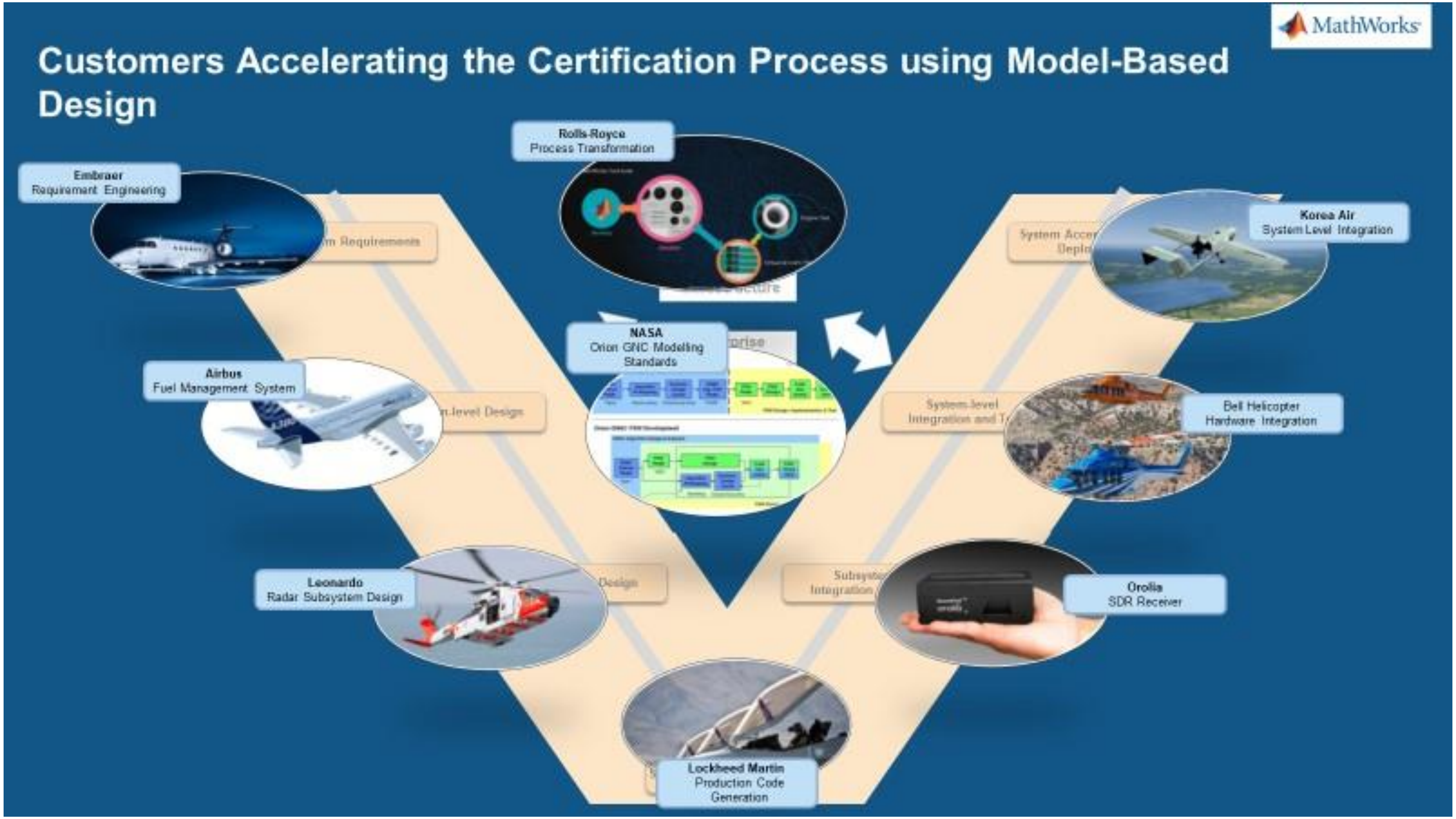
"Front-loaded development with Model-Based Design enables us to **shorten development cycles and minimize rework**, which allows us to **deliver products earlier than our competitors**."
Dr. Hisahiro Ito, Asst. GM.



System models reused across 54 products worldwide. "Once we had moved to Model-Based Design, we were able to use the same core system in many different vehicles by simply calibrating parameters such as the vehicle dimensions **and then re-generating production code.**"
Johan Hägnander, GM Engineering Europe

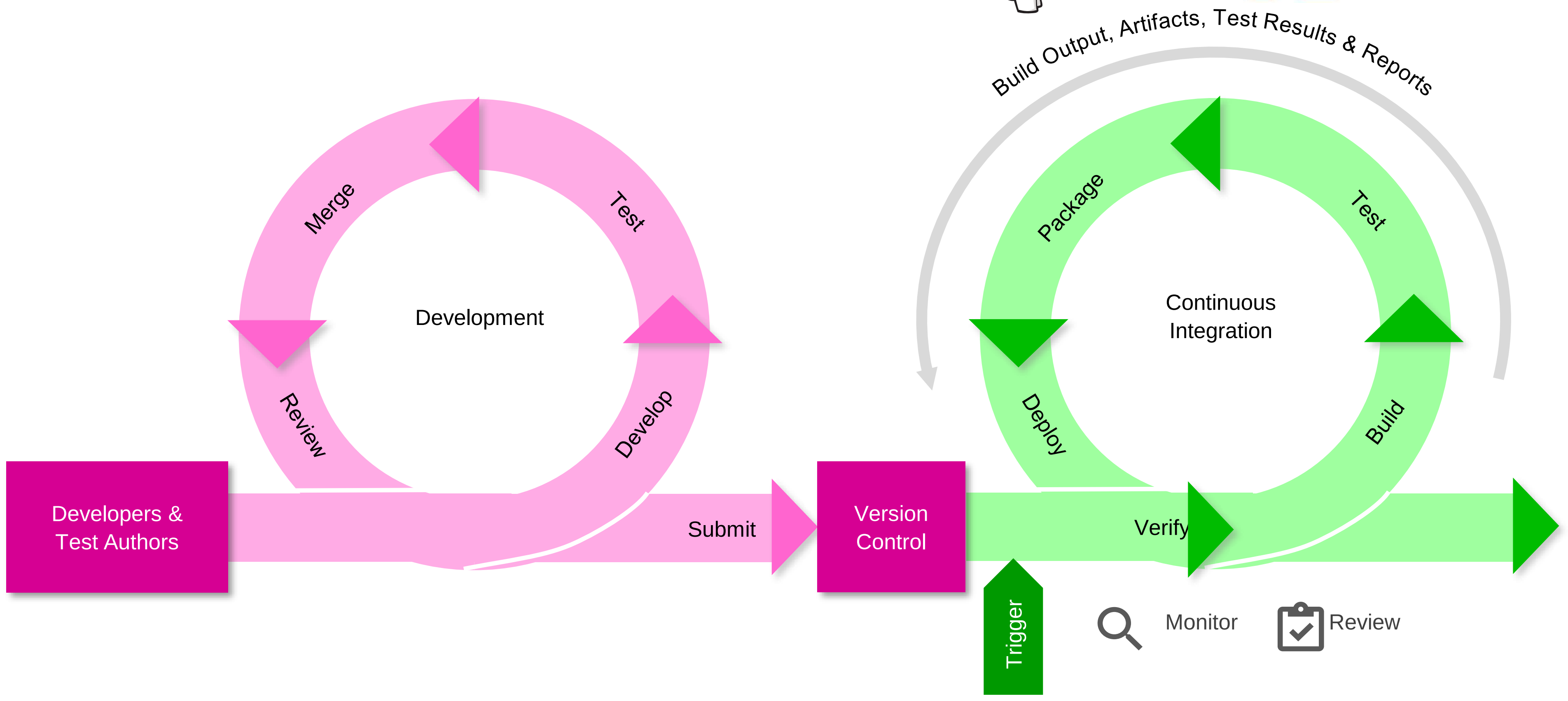
"We use our system design model in Simulink for ARP4754 to establish stable, objective requirements. **We save time by using the model as the basis for our software design model for DO-178**—**from which we generate flight code**—and reusing validation tests for software verification."
Ronald Blanrue, Airbus Helicopters

Customers Accelerating the Certification Process using Model-Based Design



Model Based Design and DevOps

& many more…

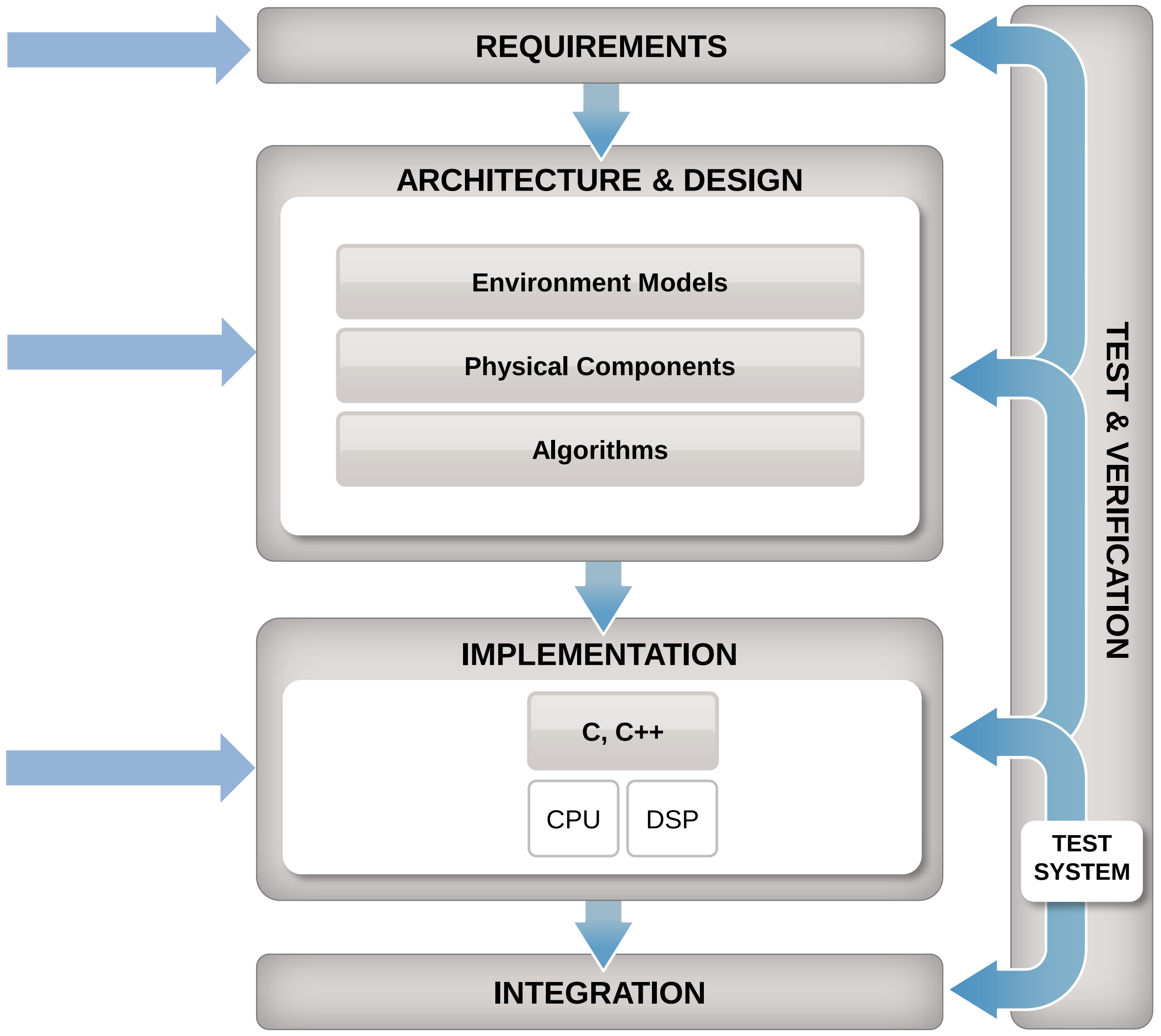


Model-Based Design Integrated Process

- Requirements Toolbox

- System Composer
- Simulink
- Stateflow
- MATLAB

- Embedded Coder



- Simulink Check
- Simulink Test
- Simulink Coverage
- Simulink Design Verifier
- Simulink Report Generator

- Simulink Code Inspector
- Polyspace Bug Finder
- Polyspace Code Prover

- Simulink Real-Time

Model Based Design Verification Workflow

Requirements Capture & Traceability
Requirements Toolbox

Verify Conformance to Standards
Simulink Check

Model and Code Coverage Analysis
Simulink Coverage

Static Code Analysis
Polyspace Bug Finder, Code Prover

Functional Testing
Simulink Test

Formal Verification
Simulink Design Verifier

Test Generation
Simulink Design Verifier

Equivalence Testing
Simulink Test

✓ **Modeling & Simulation**

✓ **Test & Validation**

✓ **Code Generation & Verification**

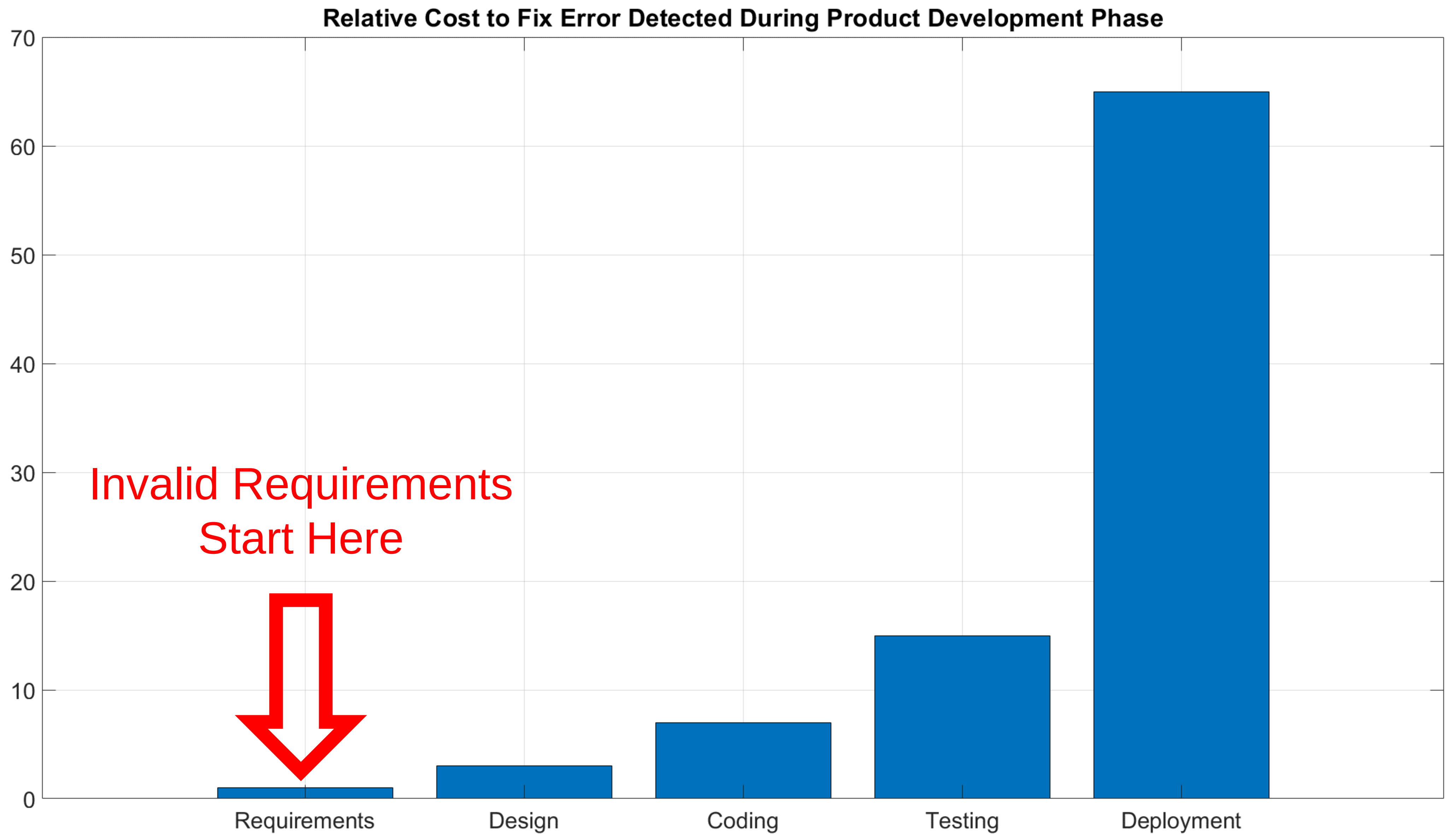
Industry Compliance: Certification
for ISO 26262, IEC 61508, DO and related standards

- Qualify tools, including
  - Embedded Coder
  - Simulink Check
  - Simulink Coverage
  - Simulink Design Verifier
  - Simulink Test
  - Polyspace Bug Finder
  - Polyspace Code Prover
- Support standards, including
  - ISO 26262 (Automotive)
  - DO178C (Aero)
  - IEC 61508 (Industrial)
  - EN 50128 (Rail)
  - IEC 62304 (Medical)

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

Leonardo Accelerates Development and Compliance of Radar Navigation Software to DO-178C

An AW101 long-range helicopter equipped with a Leonardo Osprey 30 active electronically scanned array radar system.

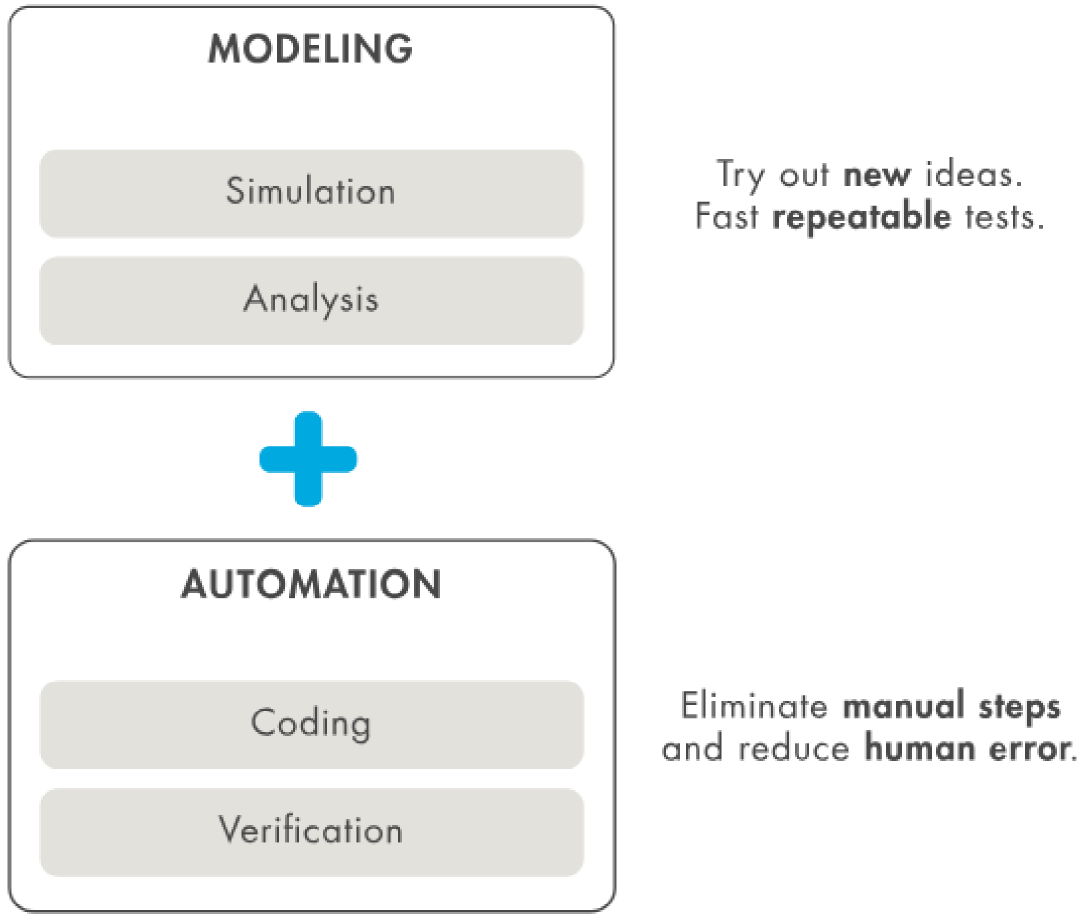


Data gathered by Hewlett Packard referred by XB in 2017
https://xbsoftware.com/blog/why-should-testing-start-early-software-project-development/

Key Takeaways

Accelerating Production of Industry-Compliant Embedded Software Using Model-Based Design

- ✓ Simulate and test your system early and often
- ✓ Validate your design with physical models
- ✓ Generate and deploy directly to your embedded system
- ✓ Verify the generated code for any Run-Time issues and comply to Coding Standards
- ✓ Maintain a digital thread with traceability throughout and comply to industry standards


MODELING
Simulation
Analysis
Try out **new** ideas.
Fast **repeatable** tests.
AUTOMATION
Coding
Verification
Eliminate **manual steps** and reduce **human error**.

Relevant Training Classes

MATLAB and Simulink Training

Advance Your Skills with MATLAB and Simulink Courses

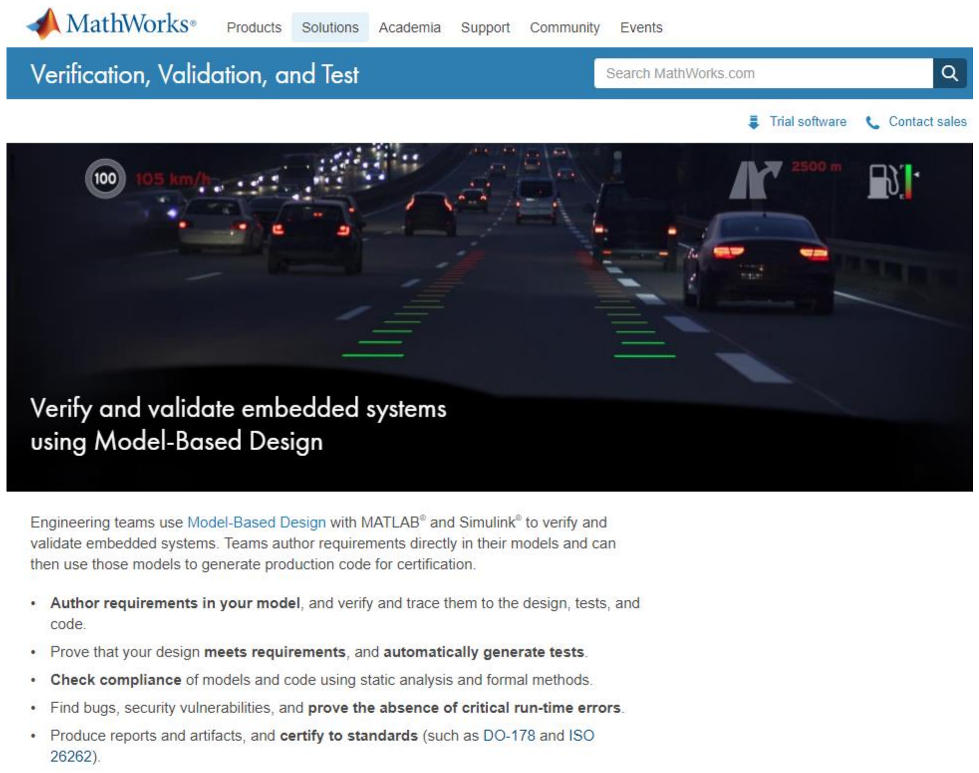
Download course catalog

- Simulink Fundamentals – introduction to designing models using Simulink
- Simulink Model Management and Architecture – Requirements Toolbox, Simulink Projects, Architectural Choices, Data Management, Simulink Report Generator
- Simulation-Based Testing with Simulink – includes Simulink Test
- Design Verification with Simulink – Simulink Design Verifier
- Embedded Coder for Production Code Generation – generating and using code from Simulink models
- Polyspace for C/C++ Code Verification – static analysis of hand code and automatically-generated code
- **Applying Model-Based Design for ISO 26262** (available upon request)

Learn More

Visit MathWorks Verification, Validation and Test Solution Page:

mathworks.com/solutions/verification-validation.html



Verification, Validation, and Test

Verify and validate embedded systems using Model-Based Design

Engineering teams use Model-Based Design with MATLAB® and Simulink® to verify and validate embedded systems. Teams author requirements directly in their models and can then use those models to generate production code for certification.

- **Author requirements in your model**, and verify and trace them to the design, tests, and code.
- Prove that your design **meets requirements**, and **automatically generate tests**.
- **Check compliance** of models and code using static analysis and formal methods.
- Find bugs, security vulnerabilities, and **prove the absence of critical run-time errors**.
- Produce reports and artifacts, and **certify to standards** (such as DO-178 and ISO 26262).

MathWorks
@MathWorks

Share the EXPO experience
#MATLABEXPO

https://www.linkedin.com/in/vamshi-krishna-kumbham-91908622/

https://www.linkedin.com/in/vaishnavi-r-a819497a/

MATLAB EXPO

Thank you

MathWorks®