

MATLAB EXPO 2019

Simplifying Requirements Based Verification with Model-Based Design

Dr. Tjorben Groß, Senior Application Engineer



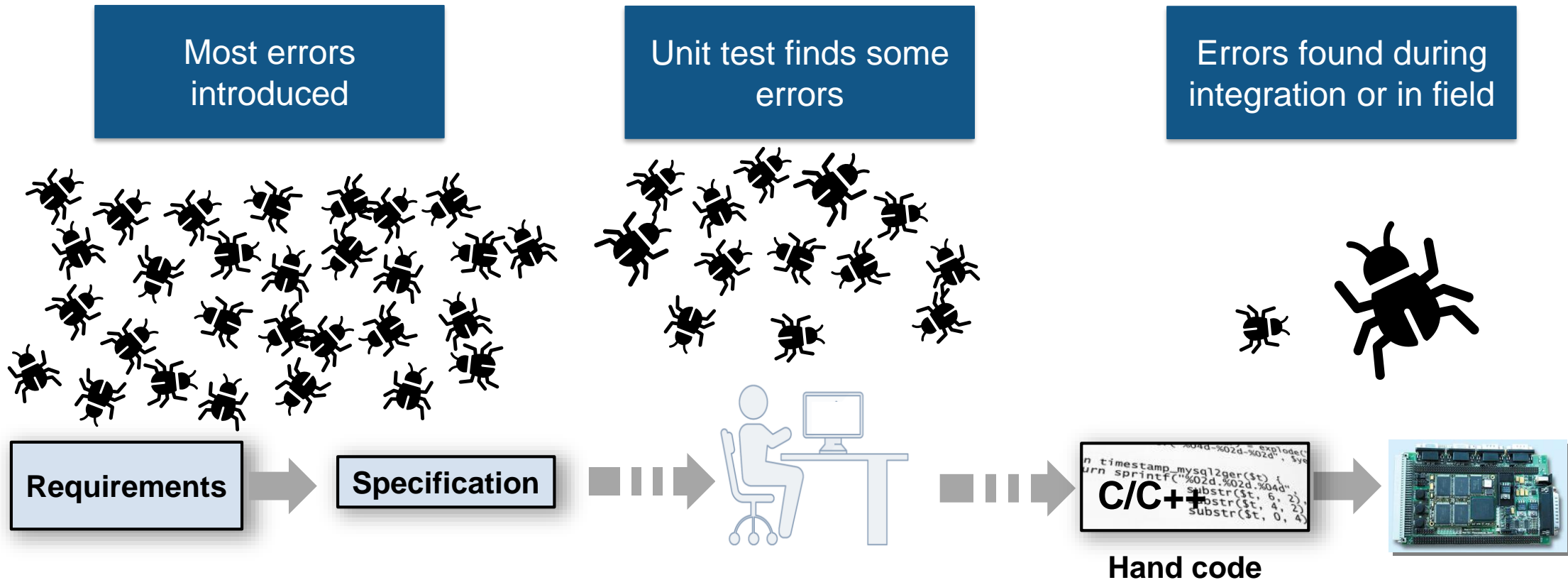
Key takeaways

- Verify and validate requirements earlier
- Identify inconsistencies in requirements by using unambiguous assessments
- Traceability from requirements to design and test

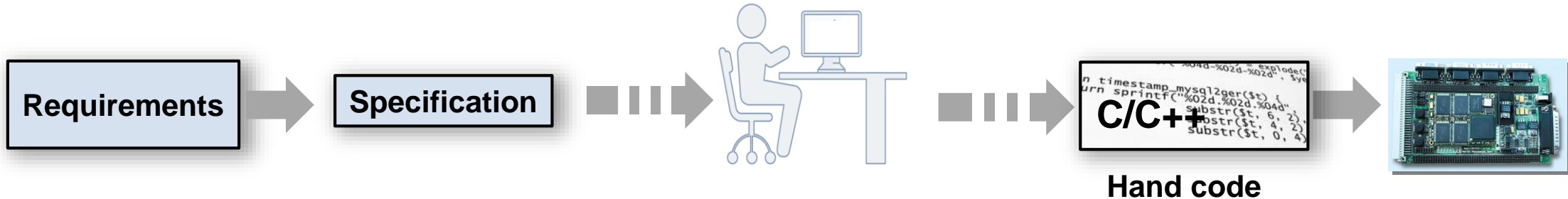
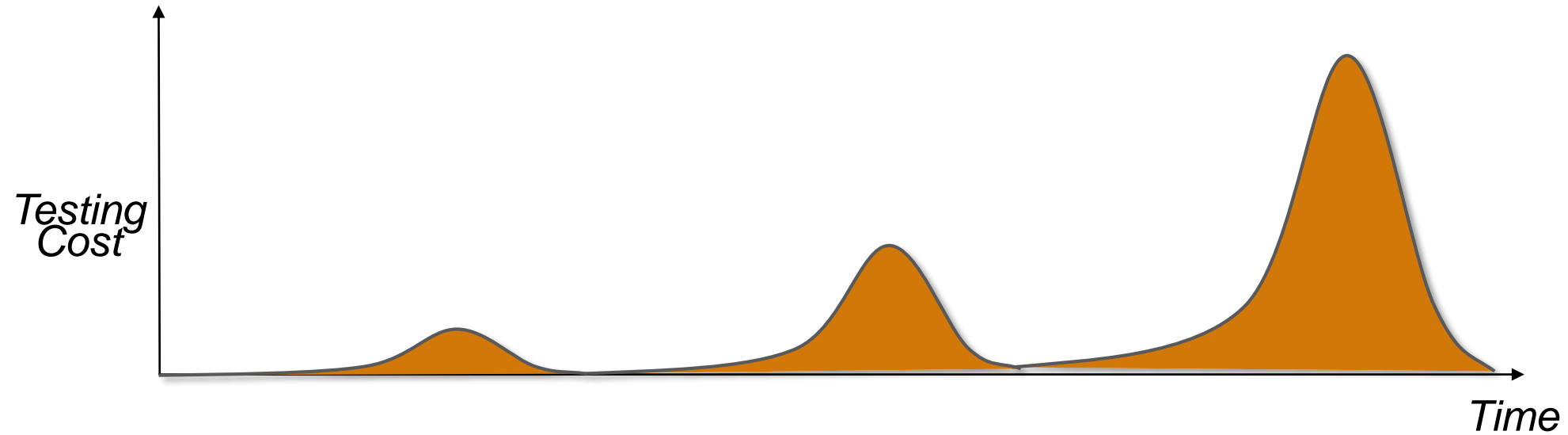
“By enabling us to analyze requirements quickly, reuse designs from previous products, and eliminate manual coding errors, Model-Based Design has reduced development times and enabled us to shorten schedules to meet the needs of our customers.”

- MyoungSuk Ko, LS Automotive

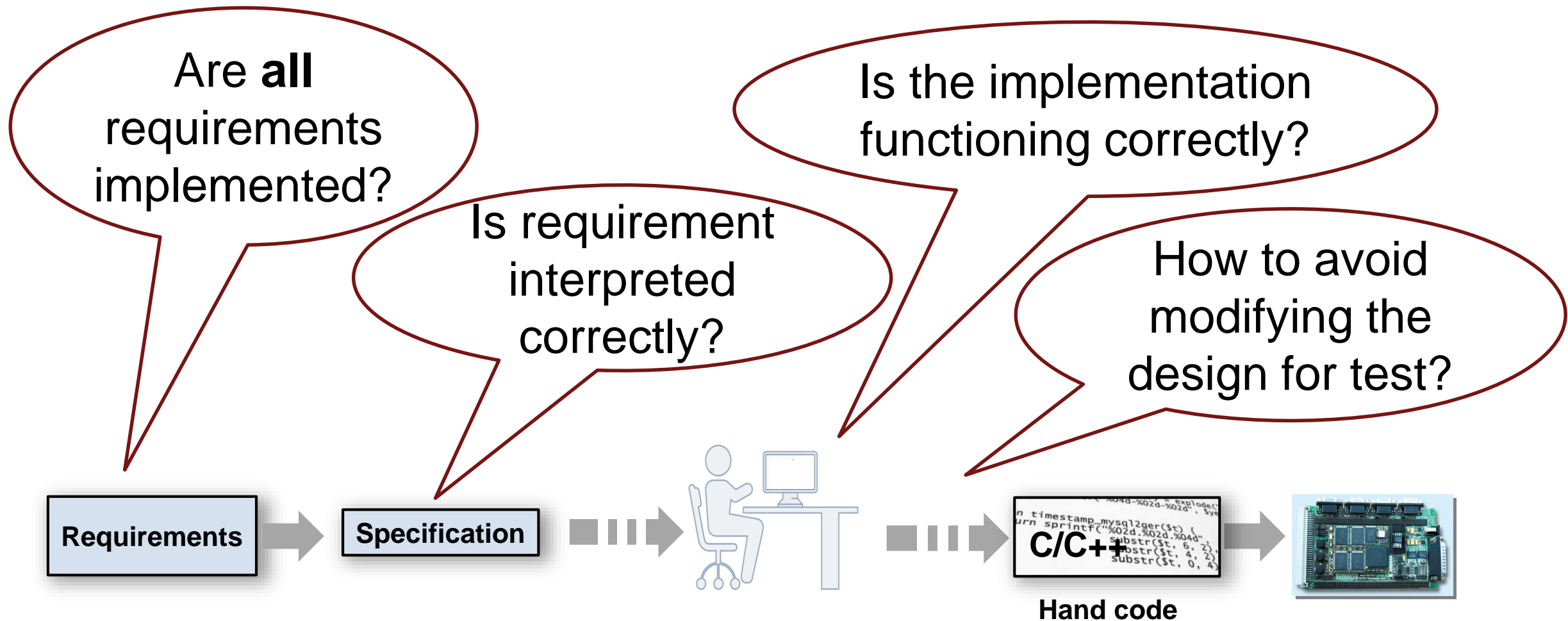
Challenge: Errors introduced early but found late



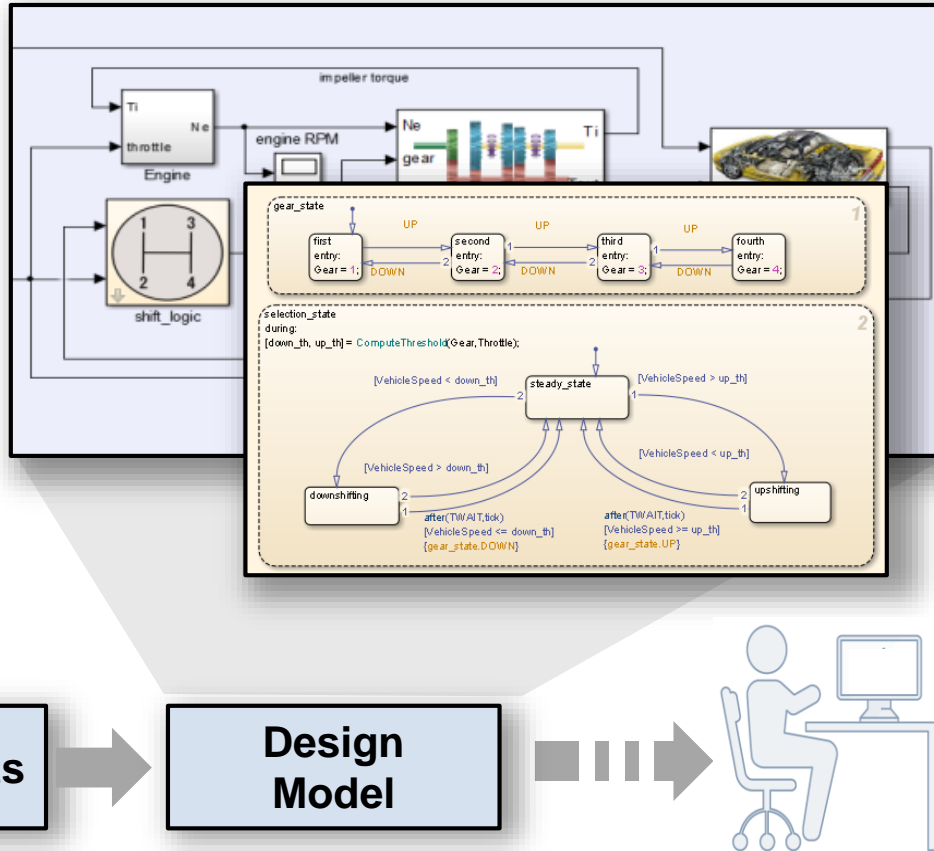
Cost of finding errors increases over time



Challenges with requirements-based verification



Simulink models for specification

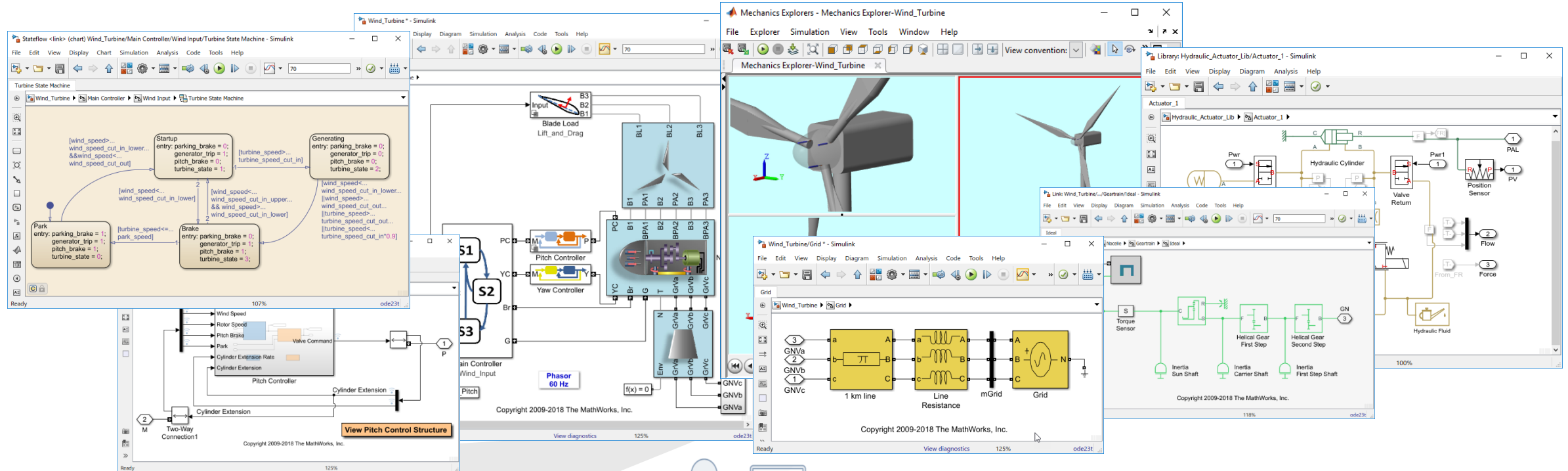


Model-Based Design enables:

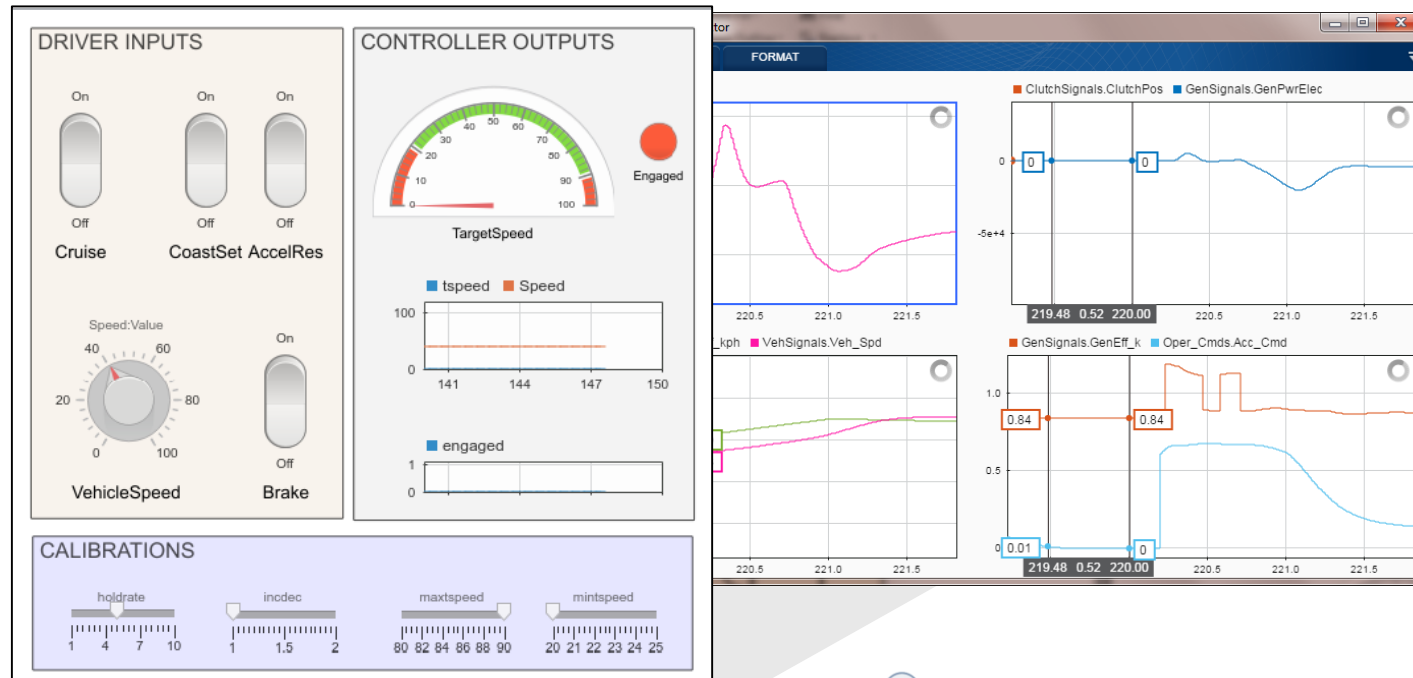
- *Early testing to increase confidence in your design*
- *Delivery of higher quality software throughout the workflow*



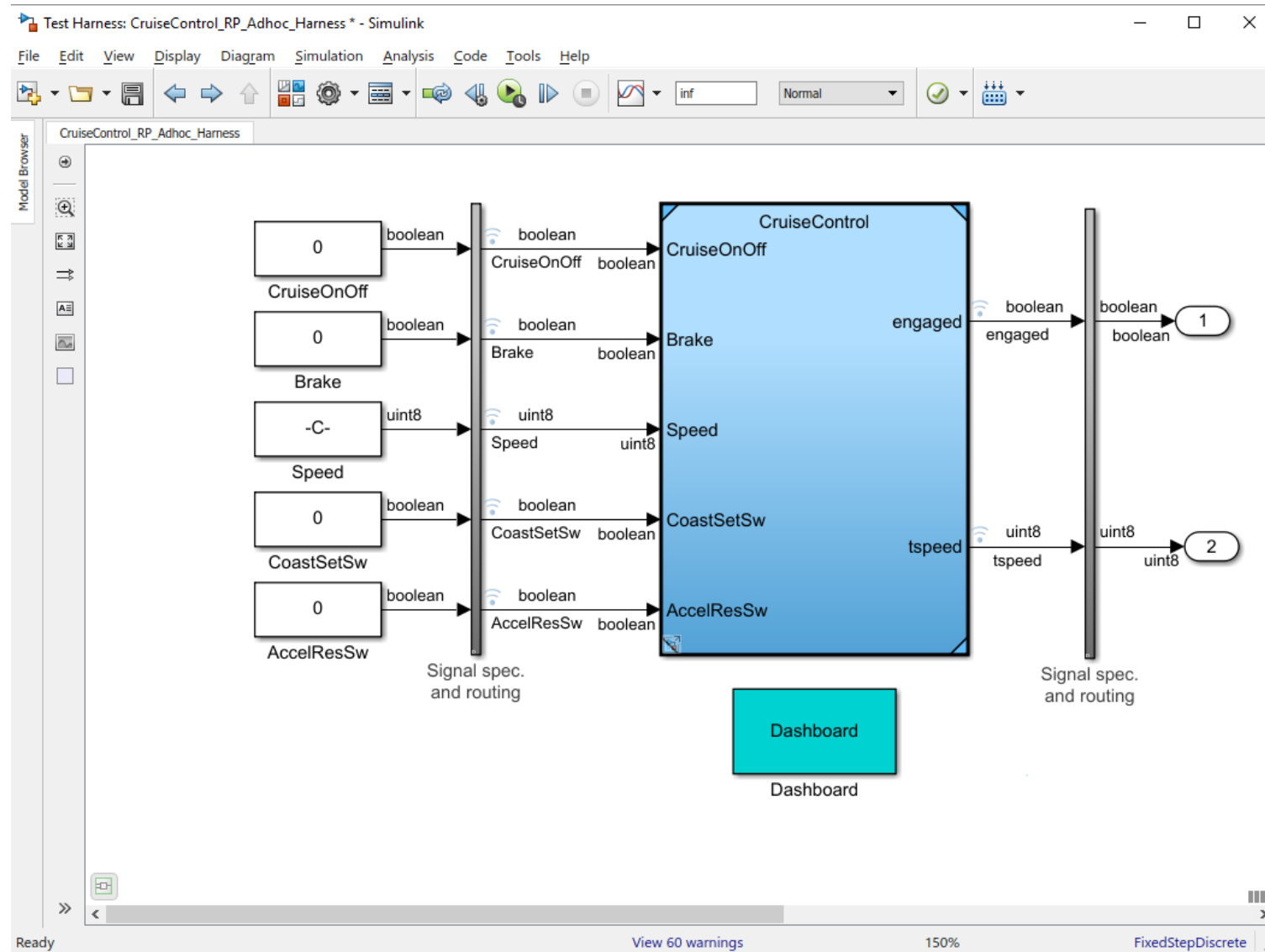
Multiple languages to describe complex systems



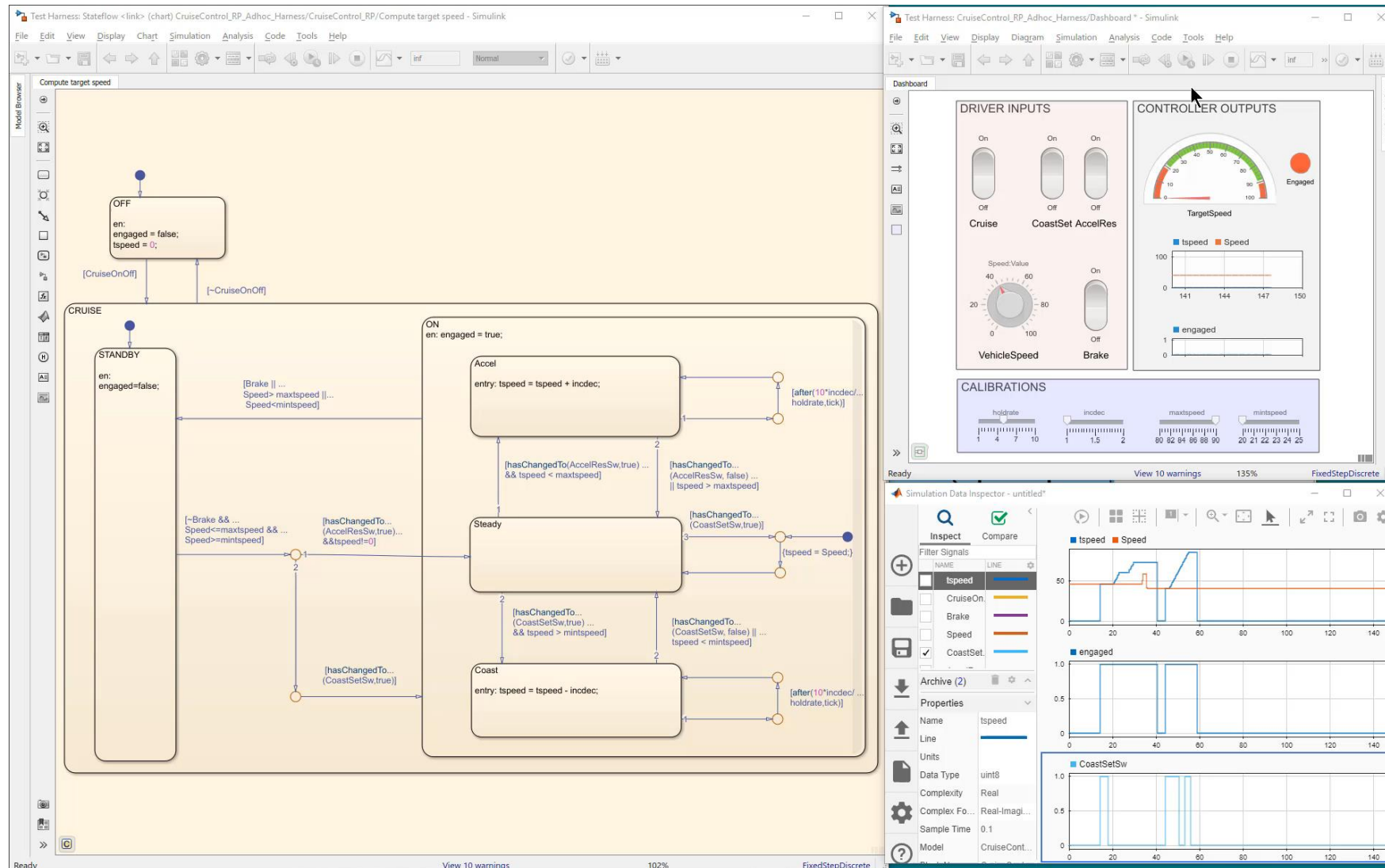
Ad-Hoc Testing: Explore behavior and design alternatives



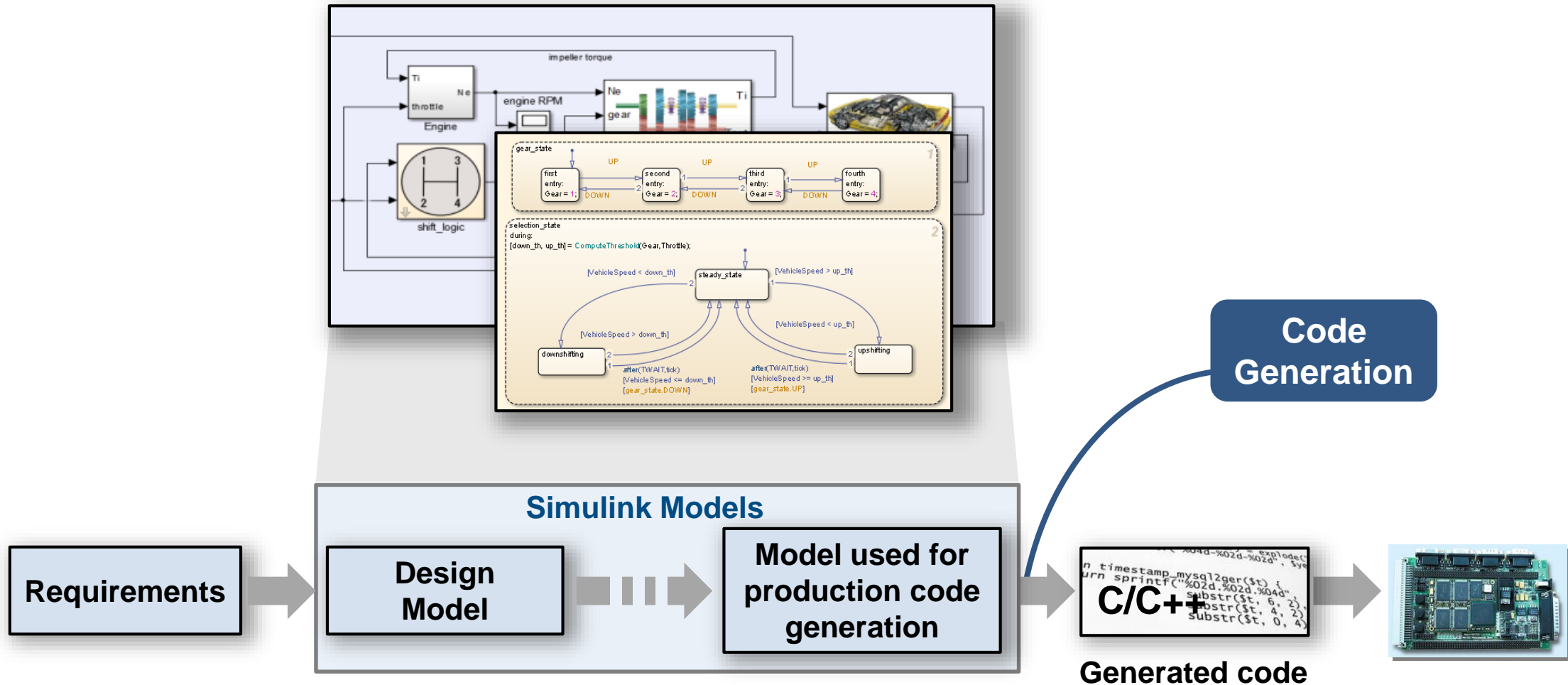
Validate behavior earlier with simulation



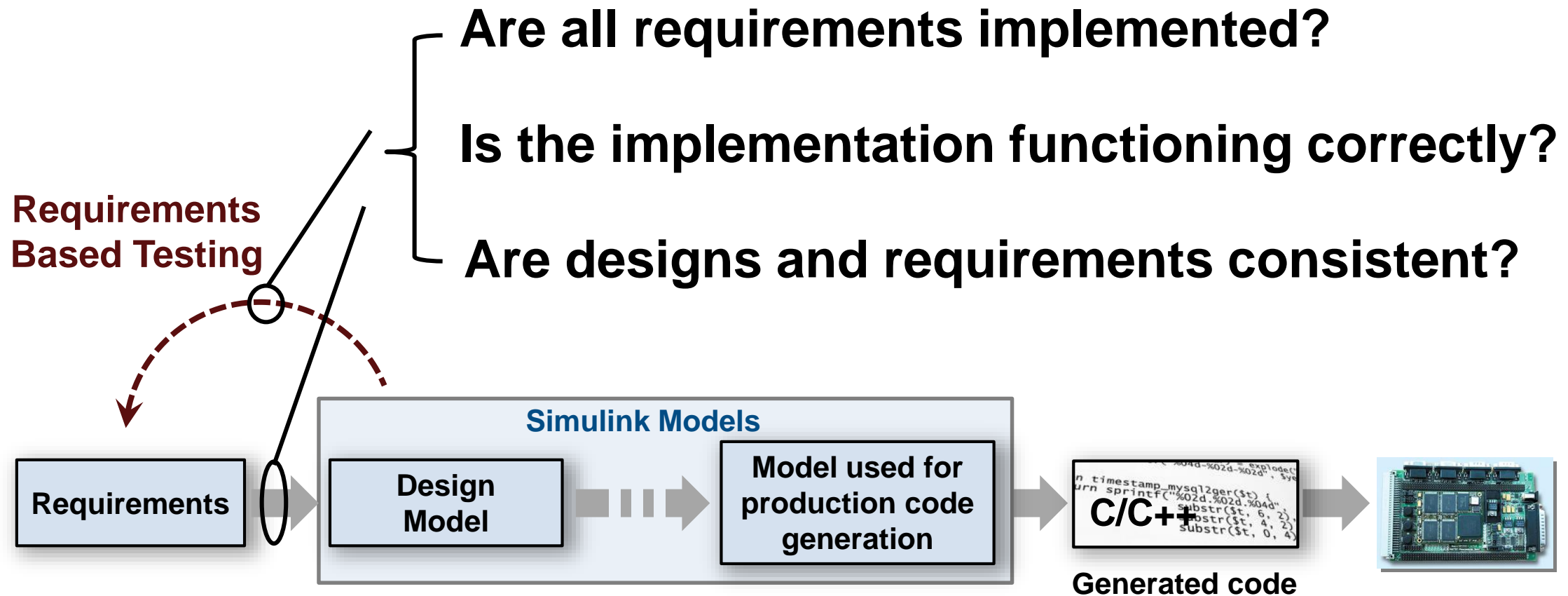
Validate Behavior Earlier with Simulation



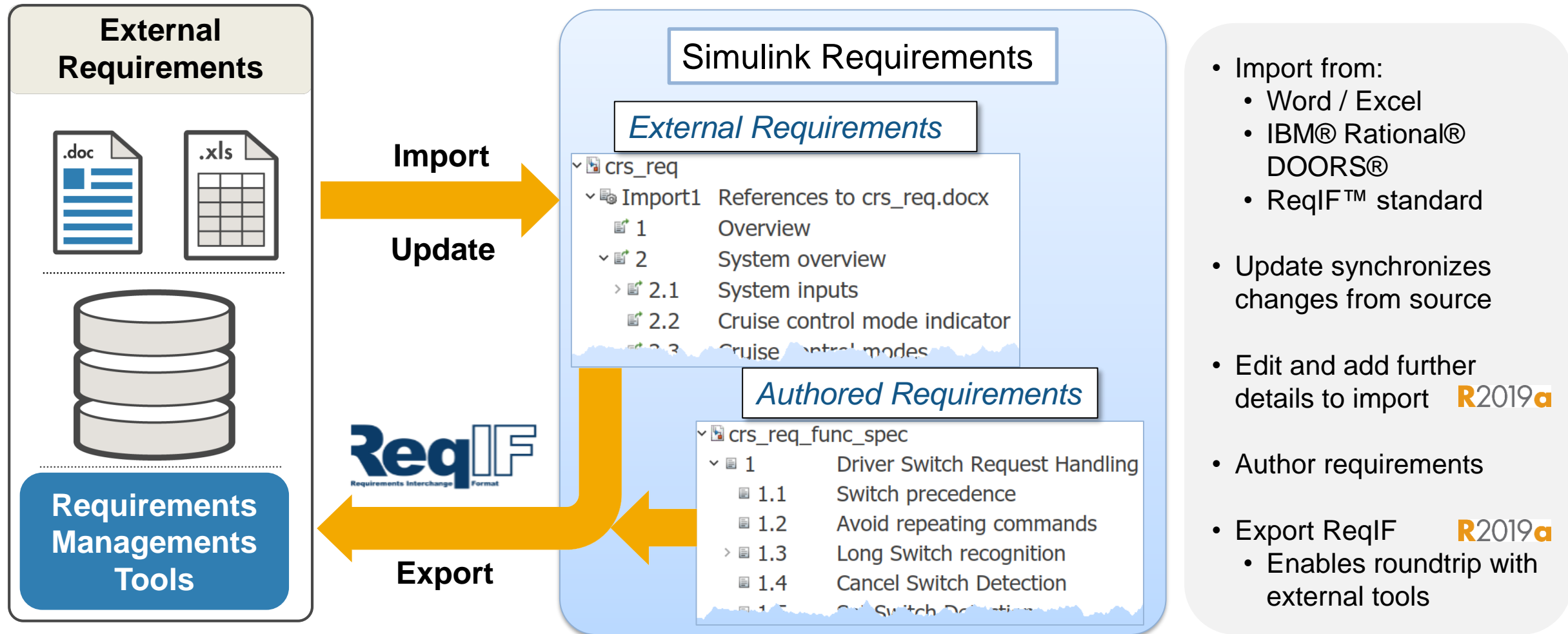
Complete Model Based Design



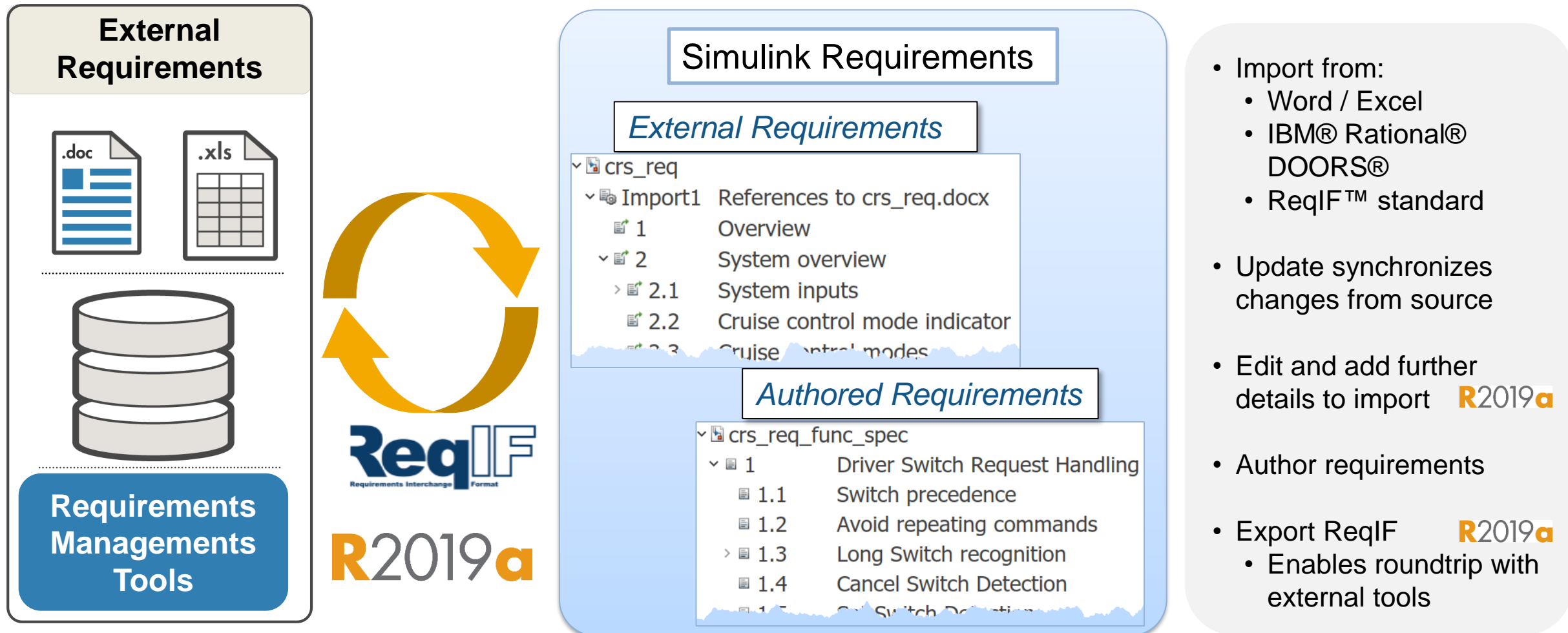
Systematically verify requirements



Integrate with requirements tools and author requirements



Roundtrip workflow with external tools thru ReqIF



- Import from:
 - Word / Excel
 - IBM® Rational® DOORS®
 - ReqIF™ standard
- Update synchronizes changes from source
- Edit and add further details to import **R2019a**
- Author requirements
- Export ReqIF **R2019a**
 - Enables roundtrip with external tools

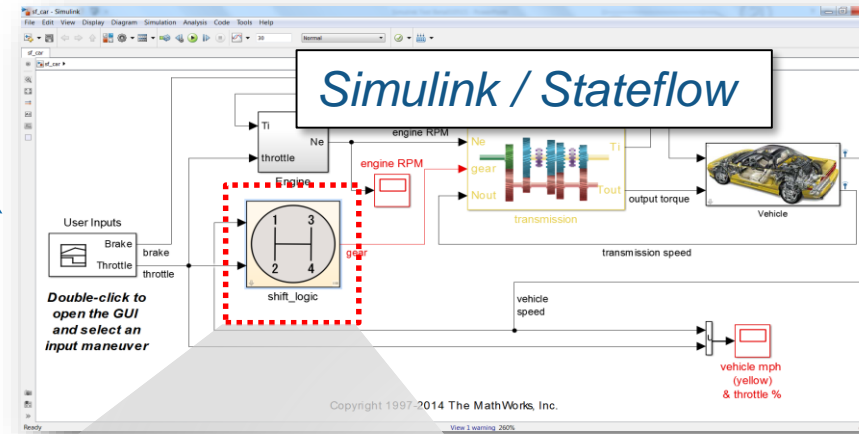
Requirements Verification with Simulink

Requirements

- TransmissionReq
 - 1 Transmission Operating Modes
 - 1.1 Reverse cannot be entered from drive
 - 1.2 Engine only starts in Park

Implemented
By

Verified
By



Test Case

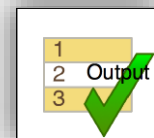
Inputs



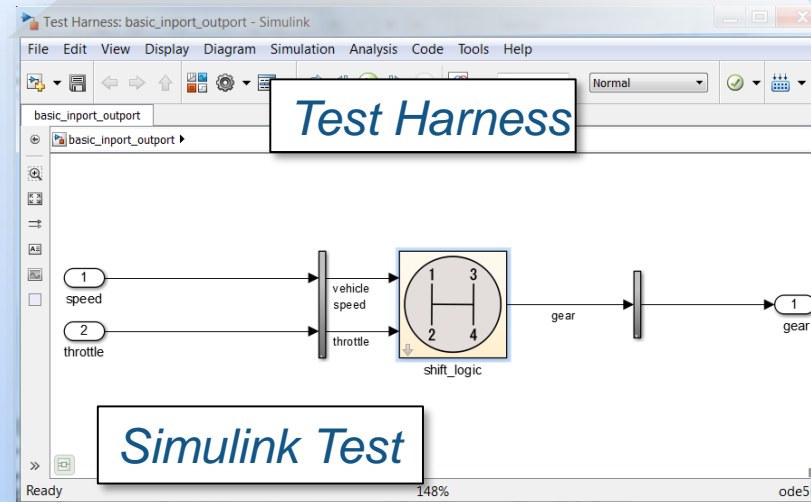
MAT / Excel
file (input)



Signal Editor

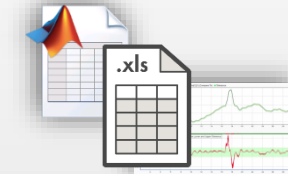


Test Sequence

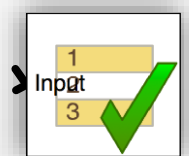


Simulink Test

Assessments



MAT / Excel
File (baseline)



Test
Assessments

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

MATLAB Unit Test

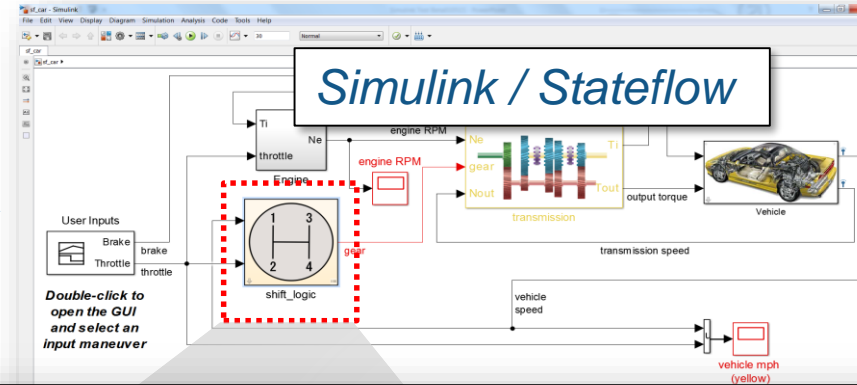
Requirements Verification with Simulink

Requirements

- ▼ crs_req_func_spec
 - ▼ 1 Driver Switch Request Handling
 - 1.1 Switch precedence
 - 1.2 Avoid repeating commands

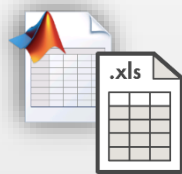
Implemented
By

Verified
By

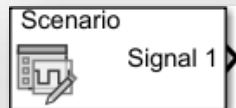


Test Case

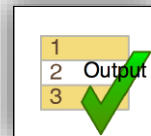
Inputs



MAT / Excel
file (input)



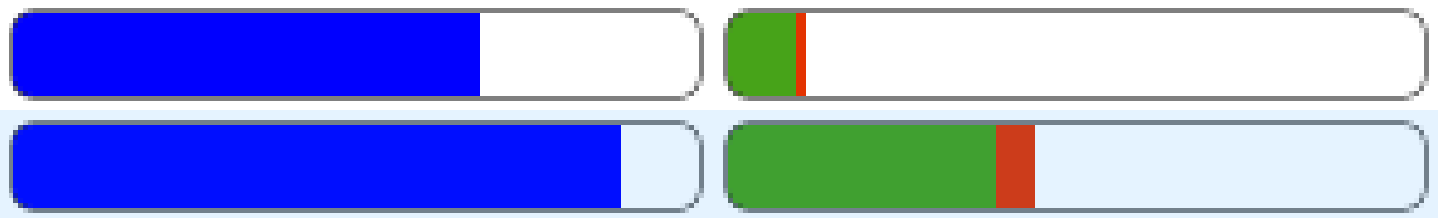
Signal Editor



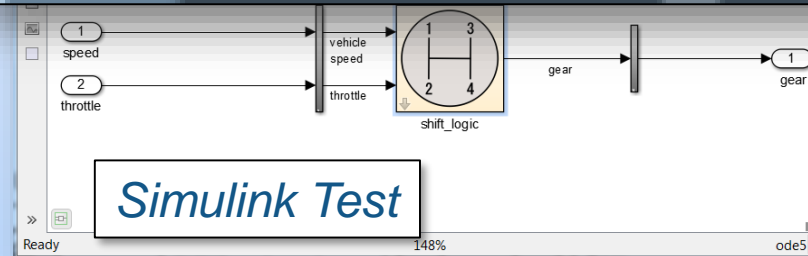
Test Sequence

Implemented

Verified



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



MAT / Excel
File (baseline)

Test
Assessments

```
function customCriteria
Perform custom criteria
1 test.verifyThat(test.sl
```

MATLAB Unit Test

Example: Verifying Heat Pump Controller Requirements

1 Requirements for the basic Heatpump Controller
Temperature difference is defined as the difference between the room and the set temperature. The controller shall turn the fan on when the temperature difference has reached a certain level, to circulate the air. The controller shall turn the heatpump on when the temperature difference has reached another level, to heat or cool the space.

1.1 Idle when Temperature in Range
If the temperature difference is less than 1 degrees, the system shall be idle with all signals off.

1.2 Activate Fan
The fan shall activate when the temperature difference is greater than or equal to 1 degrees.

1.3 Activate Heat Pump
The pump shall activate when the temperature difference is greater than or equal to 2 degrees for more than 2 seconds and stay active for at least 2 seconds.

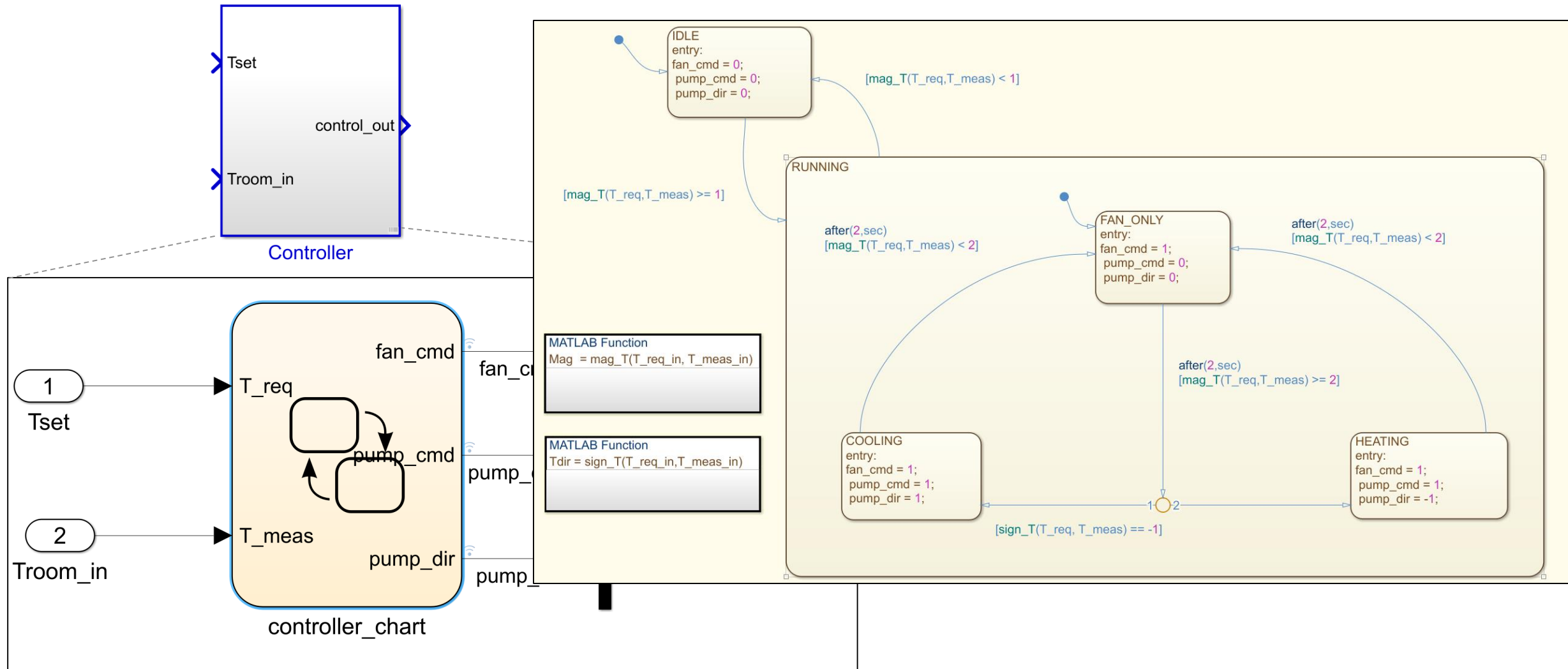
1.3.1 Cool Mode
If the room temperature is greater than the set temperature, the system shall cool the space.

1.3.2 Heat Mode
If the room temperature is less than the set temperature, the system shall heat the space.

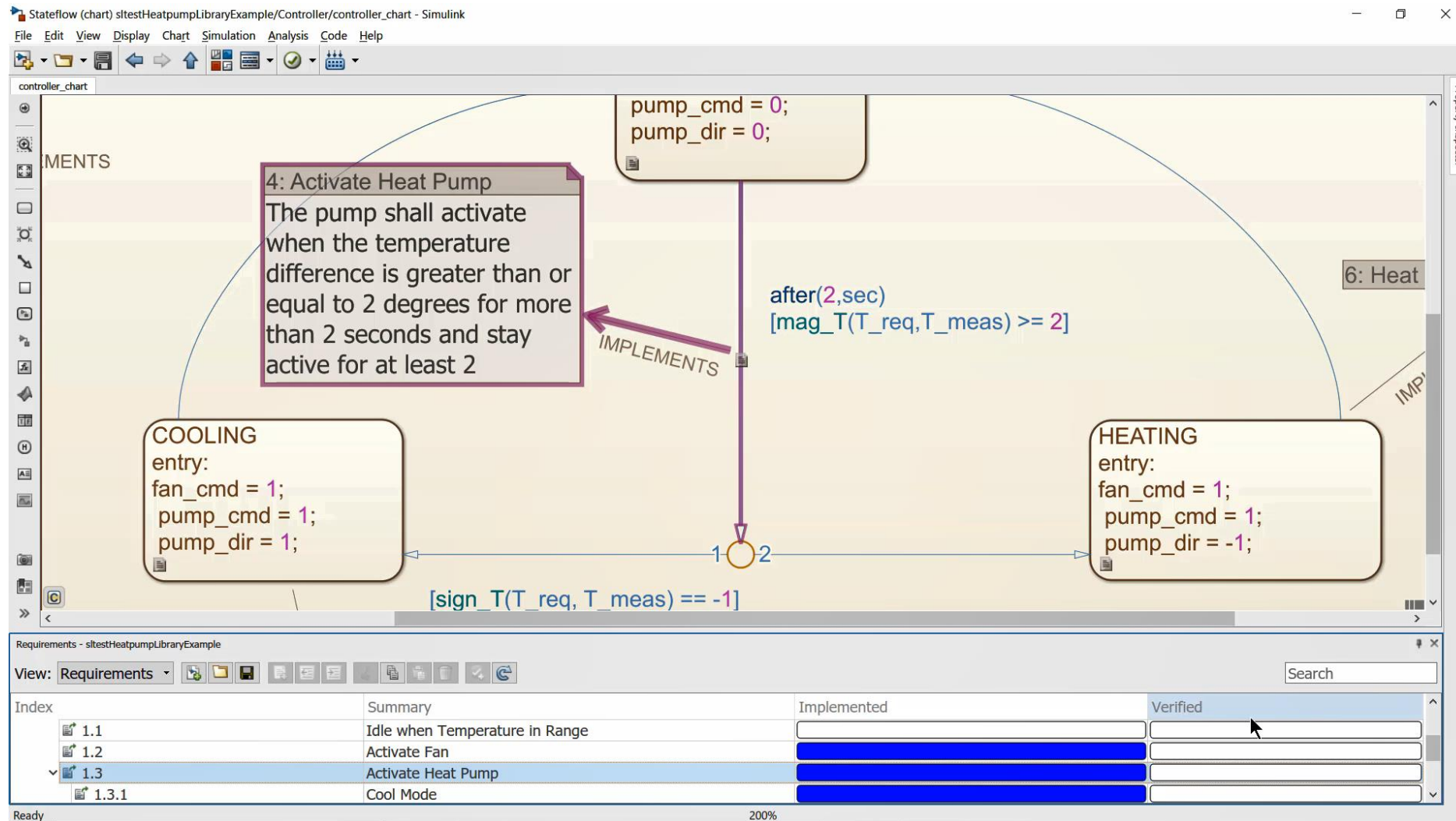
1.4 Max Temperature
The difference between the room temperature and the set temperature should never exceed 6 degrees

Requirements in DOORS

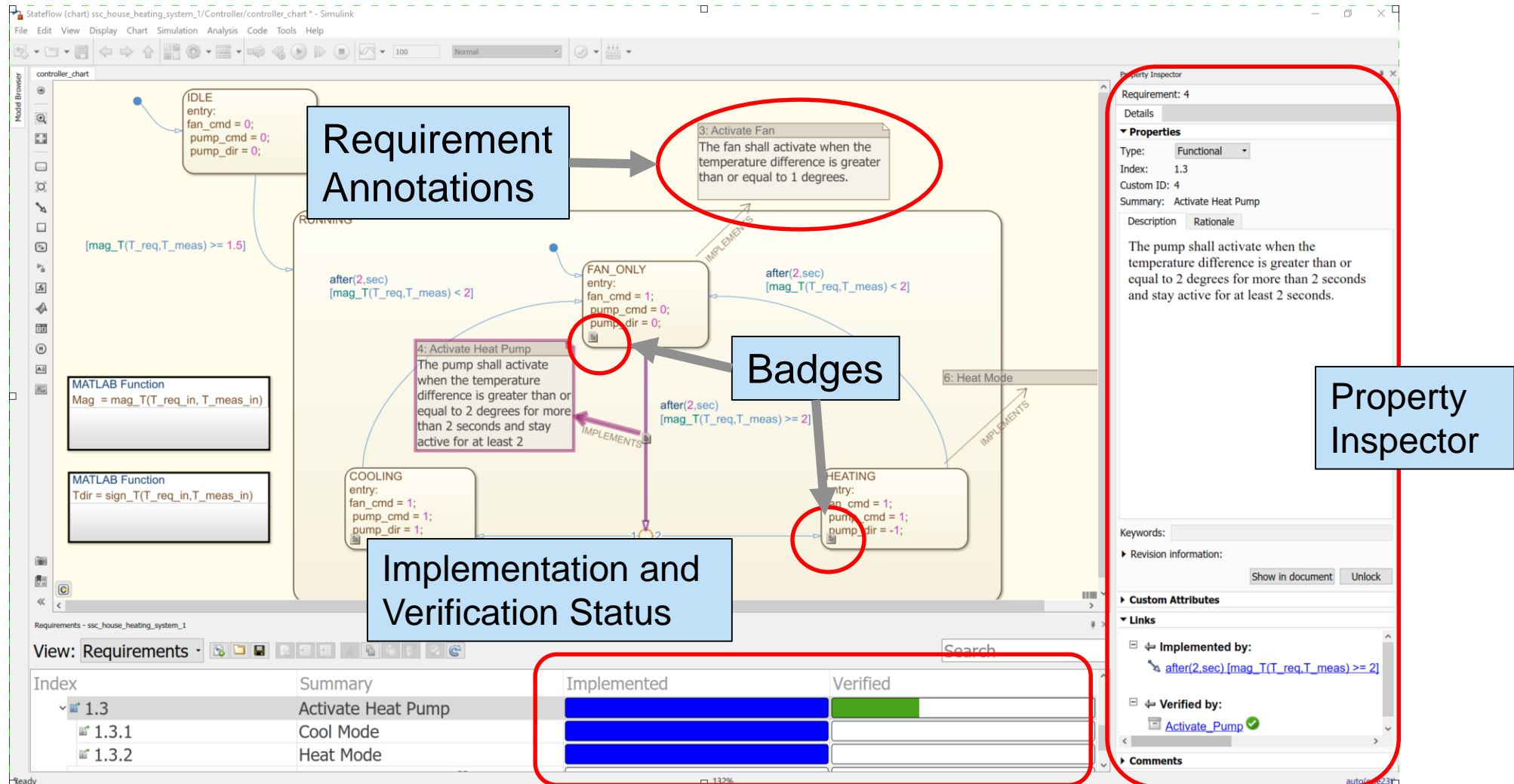
Example: Heat Pump Controller Implementation



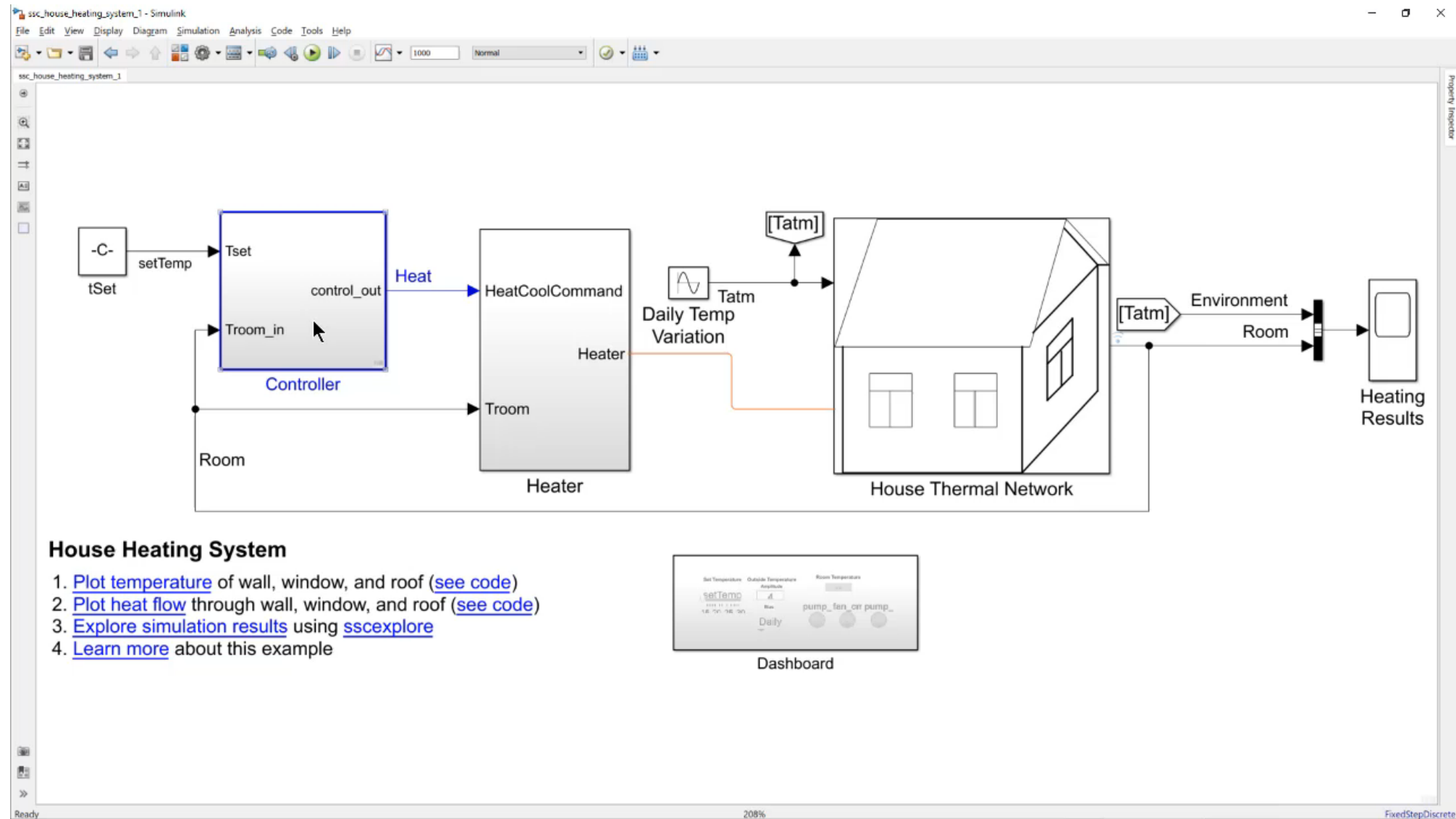
Link requirements to implementation in model



Work with Model and Requirements with Requirements Perspective



Isolate Component Under Test with Test Harness



Test Sequence Block: Step-based and temporal test sequences

ssc_house_heating_system_1_Harness1/Test Sequence - Test Sequence Editor

Symbols

Input

1. control_out

Output

1. Tset
2. Troom_in

Local

Constant

Parameter

Data Store Memory

Step Hierarchy

- Initialize
- Cold_Outside
- Hot_Outside

Step	Transition	Next Step
Initialize %% Initialize data inputs. Tset = 23; Troom_in = 23;	1. true	Cold_Outside ▼
Cold_Outside %% Check heating mode Troom_in = 23 - ramp(et*0.2);	1. Troom_in <= 15	Hot_Outside ▼
Hot_Outside %% Check cooling mode Troom_in = 23 + ramp(et*0.2);	1. Troom_in >= 27	Return_Idle ▼
Return_Idle %% Return to idle mode Troom_in = Troom_in - ramp(et*0.2);	1. Troom_in <= 22	End ▼
End Troom_in = 22		

Test Assessments: Formalize and execute requirements

R2019a

Activate Heat Pump

If the temperature difference exceeds 2 degrees for more than 2 seconds, then the pump shall activate for at least 2 seconds

When <condition 1> is true,
Then <condition 2> must be true for some time

Simple concept

$$(|x_1 - x_2| \geq x_3)^{\varepsilon} \wedge \square_{[0, t_1)} (|x_1 - x_2| \geq x_3) \rightarrow \square_{[0, t_2)} x_4$$

Hard to formalize

MTL logic

Author temporal assessments using form based editor

R2019a

use_heating_system_1_Harness1

SETTINGS OVERRIDES

S

TEMPORAL ASSESSMENTS*

ASSESSMENT	REQUIREMENTS	VISUAL REPRESENTATION
<p>▼ At any point of time ...</p> <p>▼ trigger: becomes true and stays true for at least</p> <p>condition: $\text{abs}(\text{roomTemperature} - \text{setTemperature}) \geq \text{threshold}$</p> <p>min-time (sec): <empty></p> <p>time-reference: <empty></p> <p>delay: with no delay ...</p> <p>response: <empty></p>	None	<p>⚠ Select a time reference</p> <p>⚠ Select a response</p>

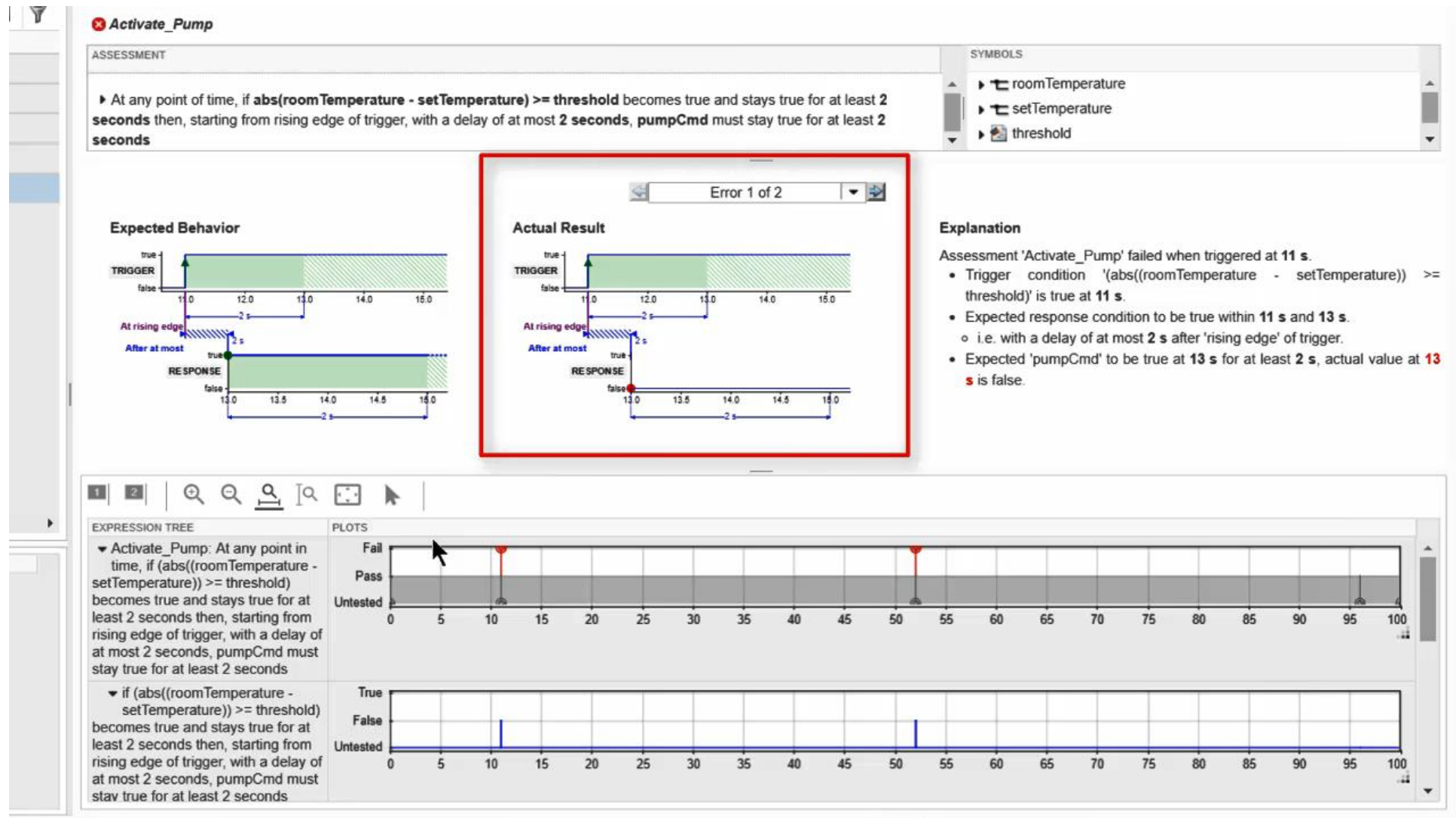
SYMBOLS

- roomTemperature
- setTemperature
- threshold

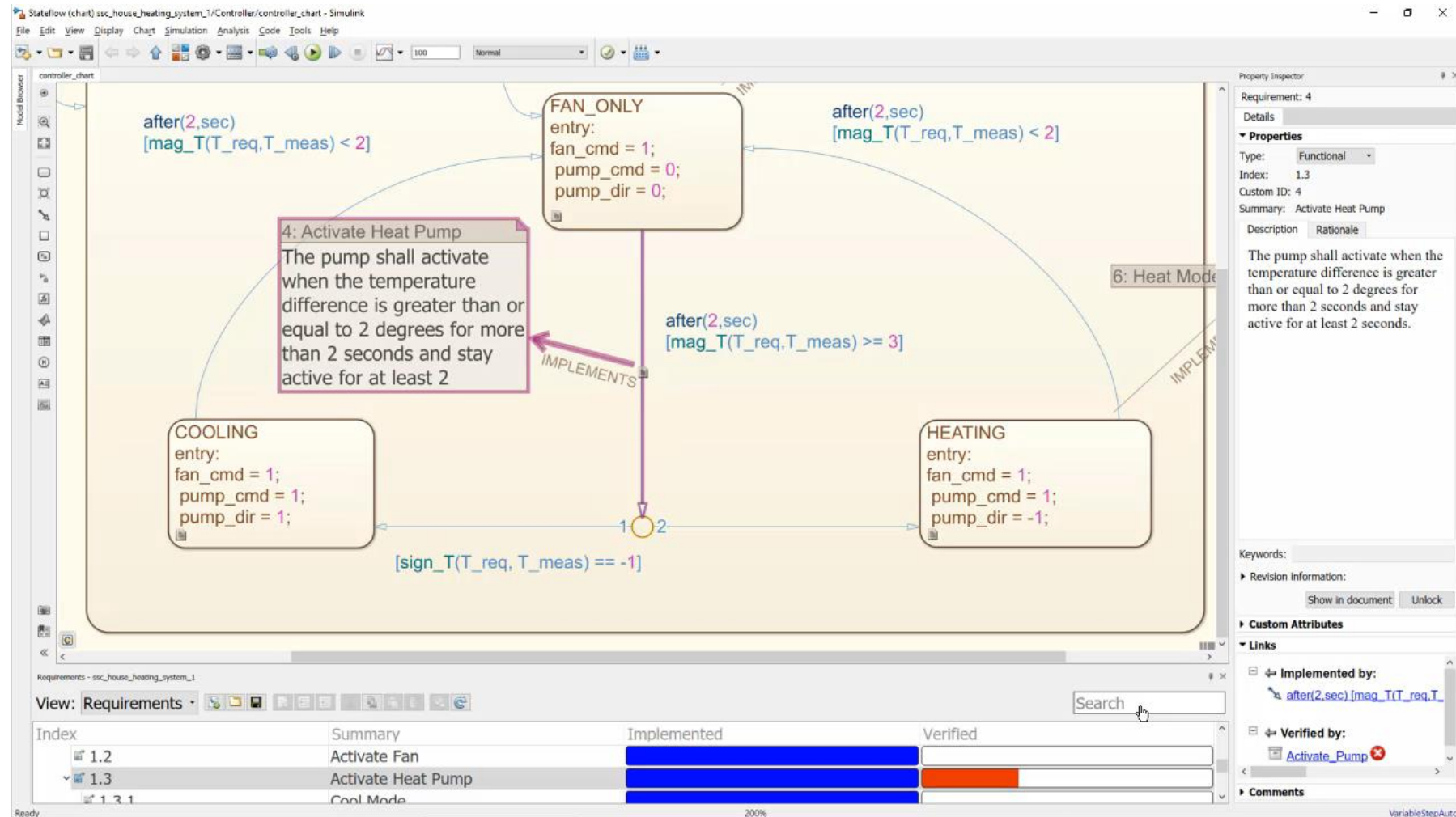
Delete + Add Symbol

Execute assessments to verify requirements

R2019a

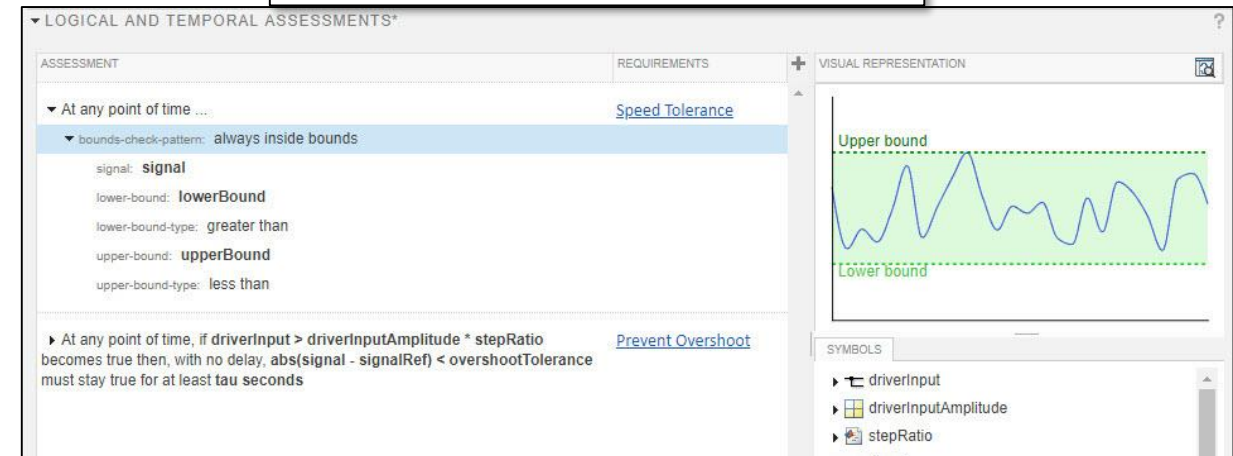


Locate implementation of requirement using link

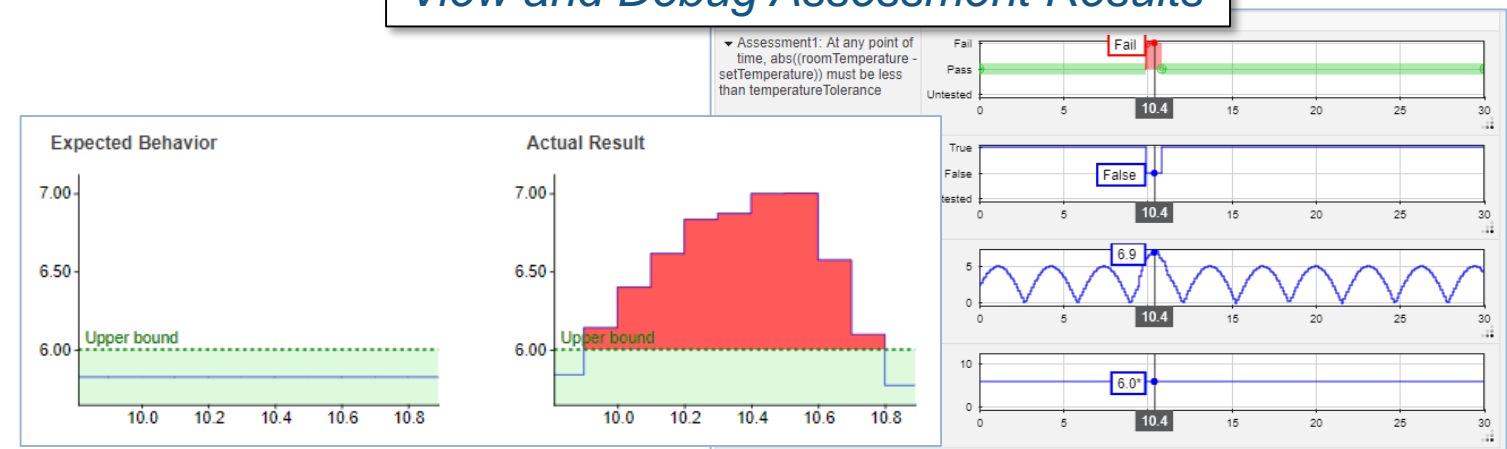


Translate textual requirements into unambiguous Temporal Assessments

Temporal Assessment Editor



View and Debug Assessment Results





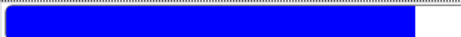

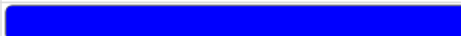



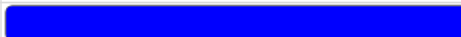

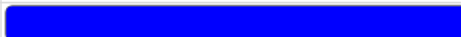

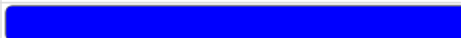



- Compose assessments using form based editor
- View assessments as English-like sentence
- Review and debug temporal assessment results
- Link to requirements

Track Implementation and Verification

Requirements - crs_controller

View: Requirements





Search

Index	ID	Summary	Implemented	Verified
crs_req_func_spec	-	-		
1	#1	Driver Switch Request Handling		
1.1	#2	Switch precedence		
1.2	#3	Avoid repeating commands		
1.3	#4	Long Switch recognition		
1.4	#7	Cancel Switch Detection		
1.5	#8	Set Switch Detection		
1.6	#9	Enable Switch Detection		

Implementation Status

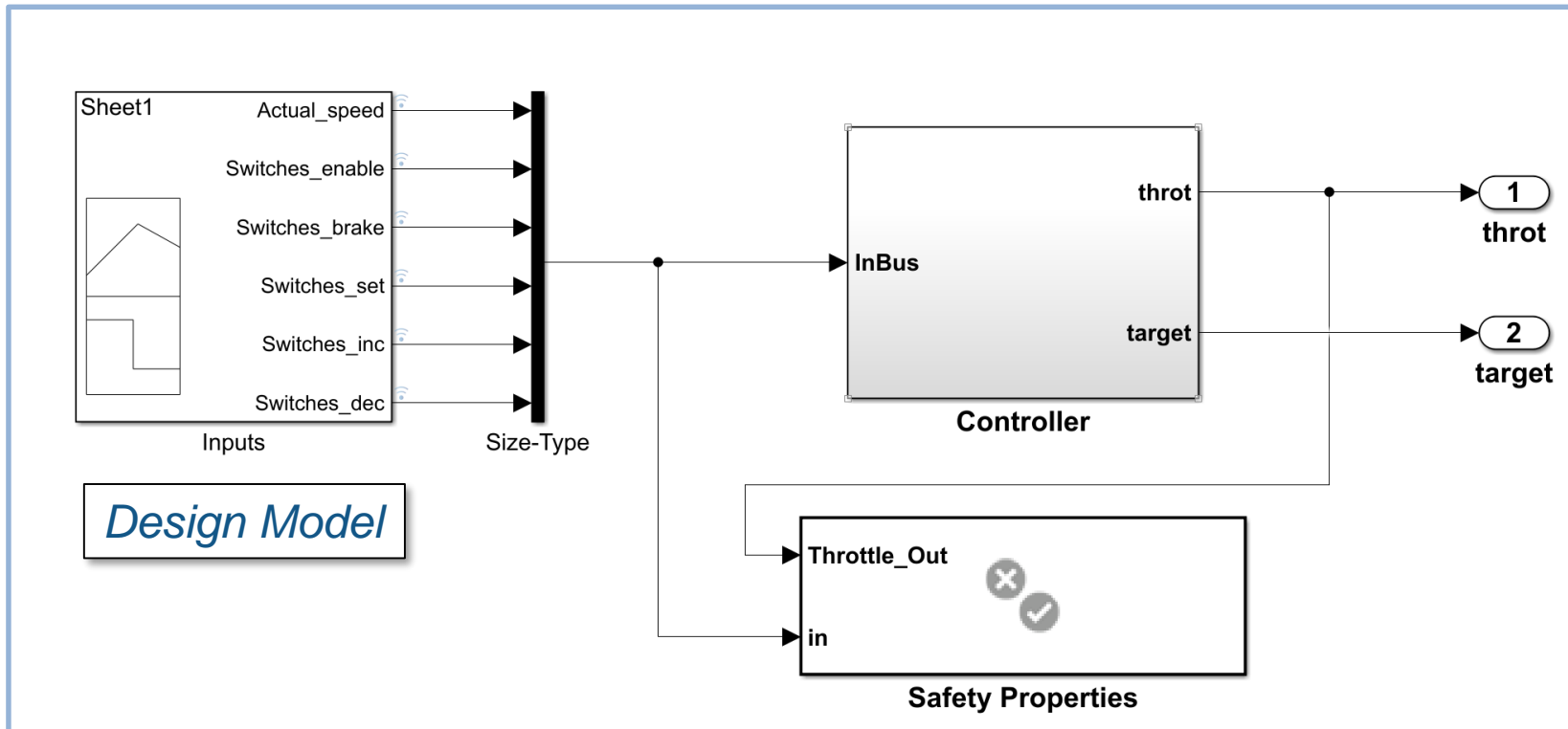
-  Implemented
-  Justified
-  Missing

Verification Status

-  Passed
-  Failed
-  Unexecuted
-  Missing

Observers: Separate test/verification logic from design

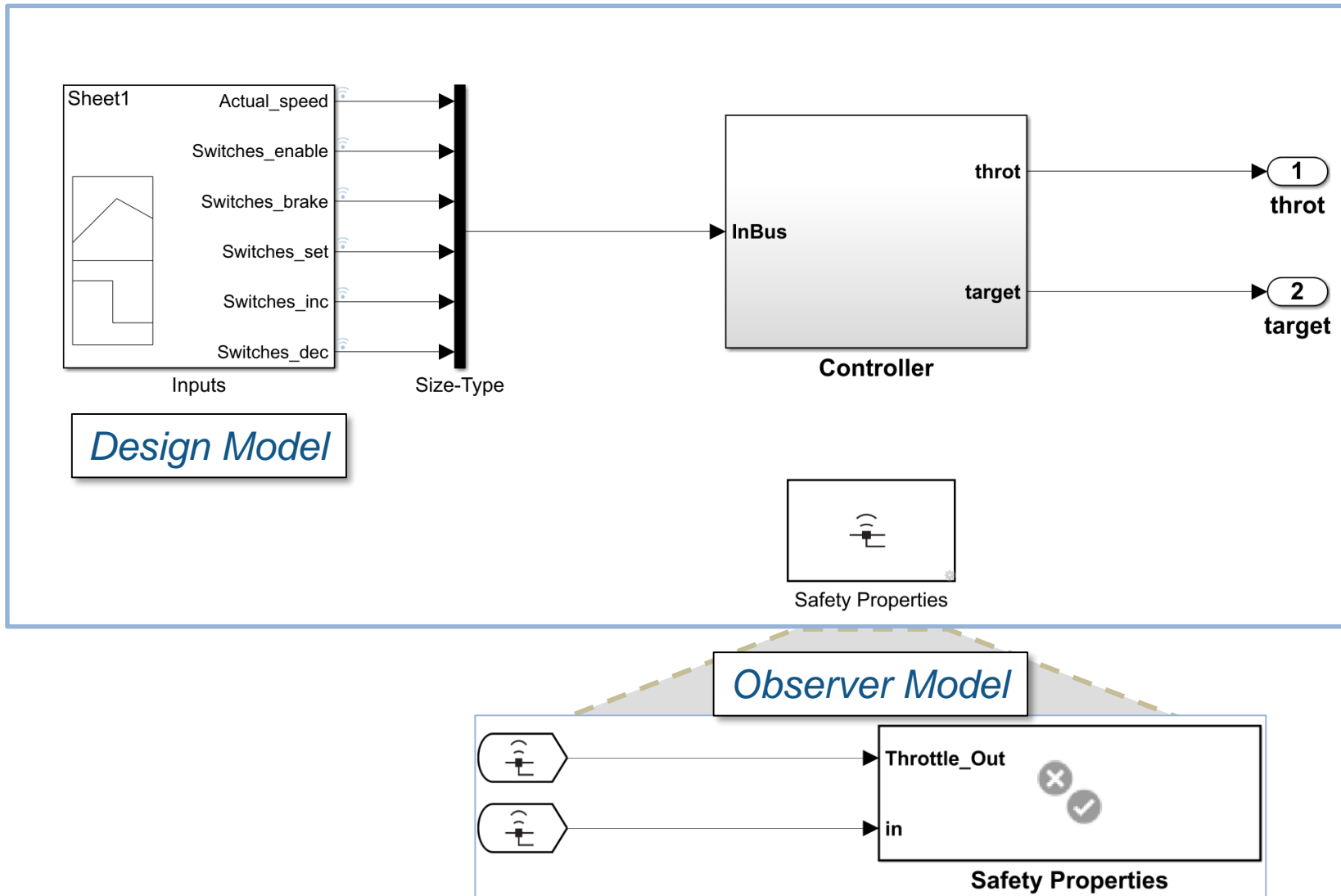
R2019a



- Access nested signals without signal lines or changing dynamic response
- Avoid modifying interface for testing
- Simplify design and test by avoiding additional signal lines

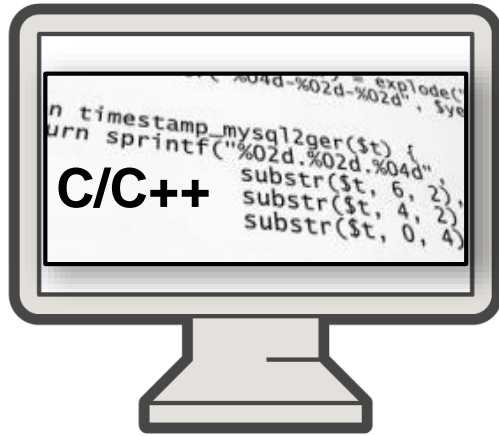
Observers: Separate test/verification logic from design

R2019a



- Access nested signals without signal lines or changing dynamic response
- Avoid modifying interface for testing
- Simplify design and test by avoiding additional signal lines

Re-use tests developed for model to test code



Software in the Loop (SIL)

- Show functional equivalence, model to code
- Execute on desktop



Processor in the Loop (PIL)

- Numerical equivalence, model to target code
- Execute on target board



Hardware in the Loop (HIL)

- Check real-time behavior of the design and code.
- Execute on Speedgoat target computer using Simulink Real-Time

IDNEO Accelerates Development of AUTOSAR Software Components and Complex Device Drivers with Model-Based Design

Challenge

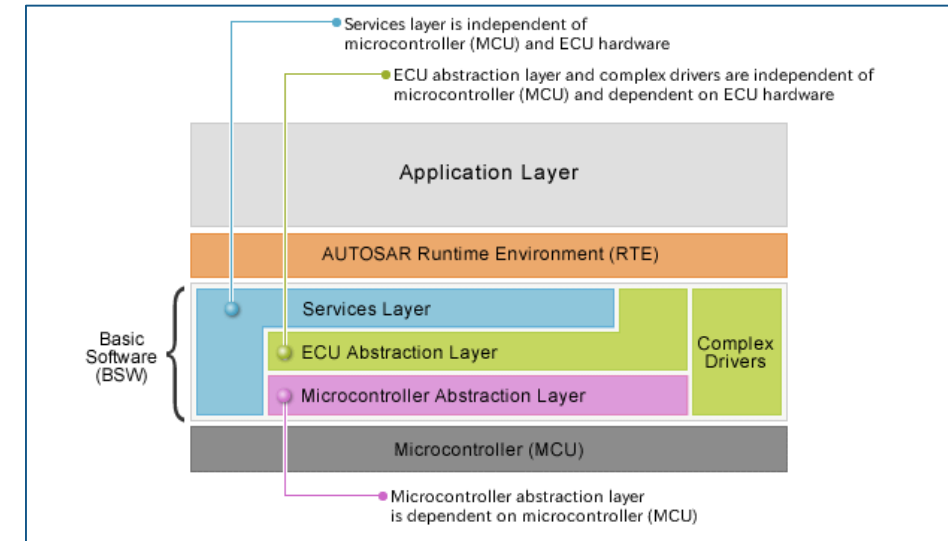
Reduce development time for embedded software for automotive applications

Solution

Use MATLAB and Simulink to model AUTOSAR software components and complex device drivers, run simulation-based tests, and generate embedded C code

Results

- Development time cut by at least 50%
- 80% of errors detected before hardware testing
- Test harnesses and MISRA-compliant C code generated from models



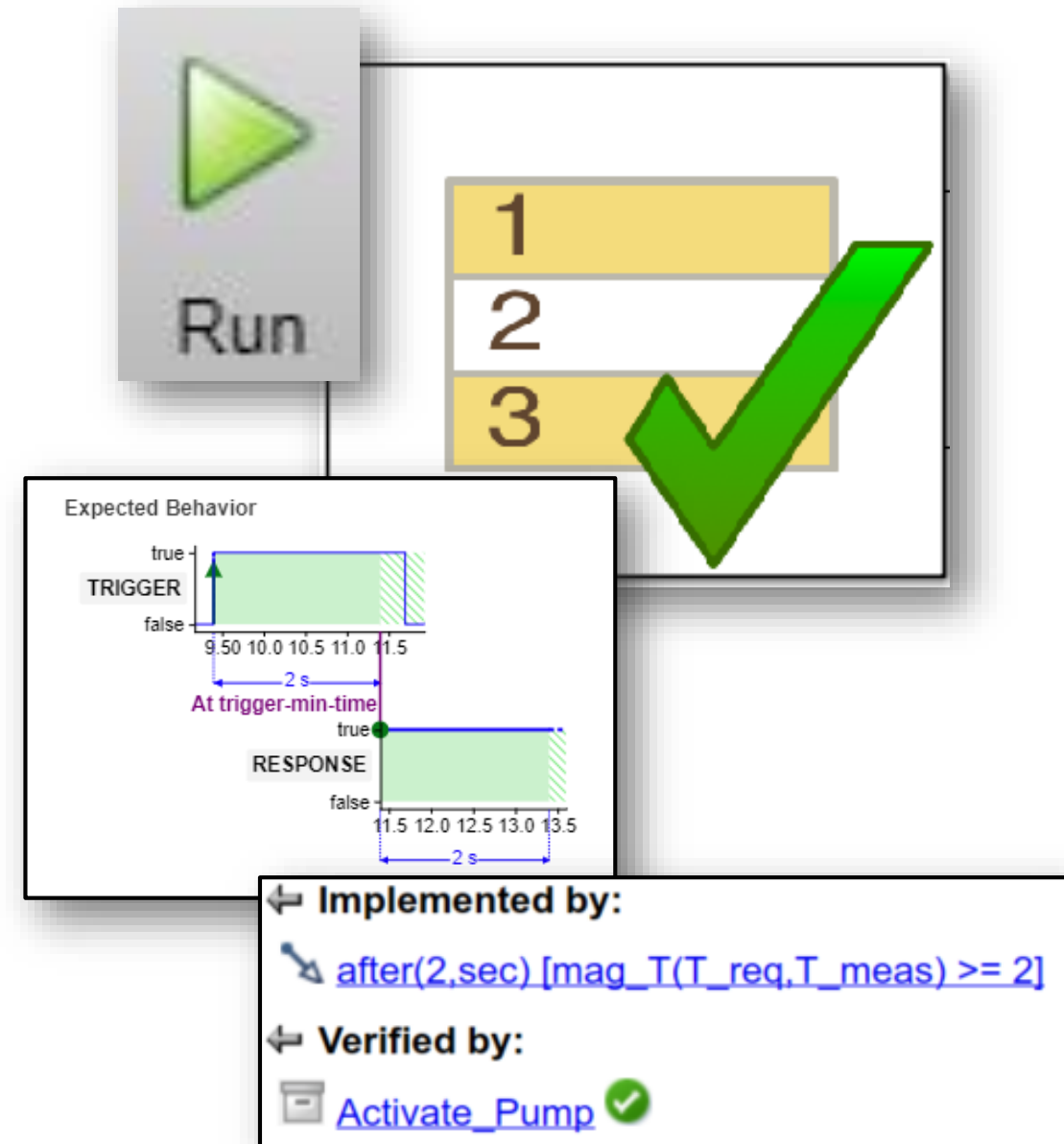
AUTOSAR software architecture.

“By using Model-Based Design for all AUTOSAR projects, we have cut development time by at least 50 percent while increasing the number of defects identified early in the design phase and reducing the number of defects found in hardware tests and beyond.”

- Joan Albesa, IDNEO

Summary

- Verify and validate requirements earlier
- Identify inconsistencies in requirements by using unambiguous assessments
- Traceability from requirements to design and test



Learn More

Key products covered in this presentation:

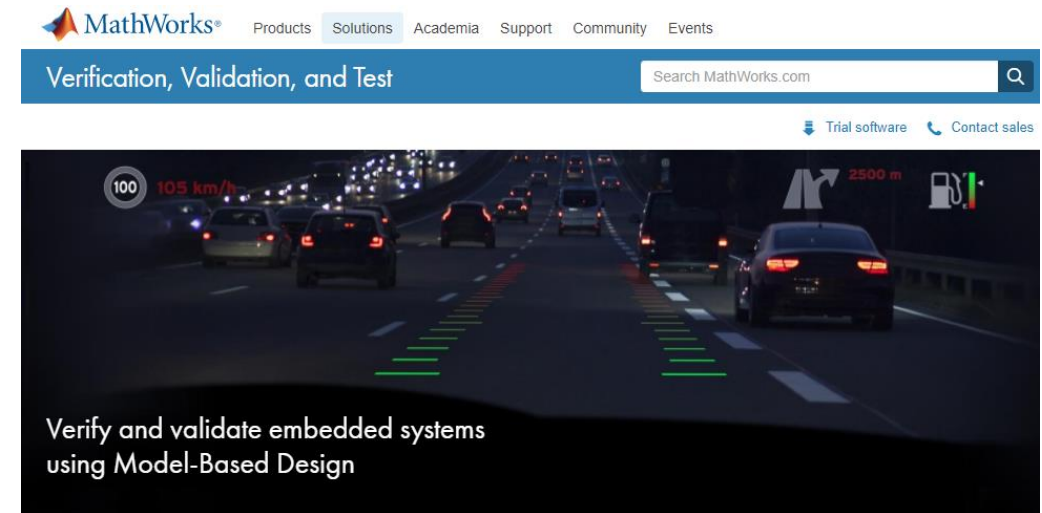
- [Simulink Requirements](#)
- [Simulink Test](#)
- [Embedded Coder](#)
- [Simulink Real-Time](#)

Learn more at Verification, Validation and Test Solution Page:

mathworks.com/solutions/verification-validation.html

MathWorks Training Services:

mathworks.com/training-schedule/simulation-based-testing-with-Simulink



Visit our Demo Booth