

# MATLAB EXPO 2019

## Simplifying Requirements Based Verification with Model-Based Design

Vamshi Kumbham  
Pilot Engineering



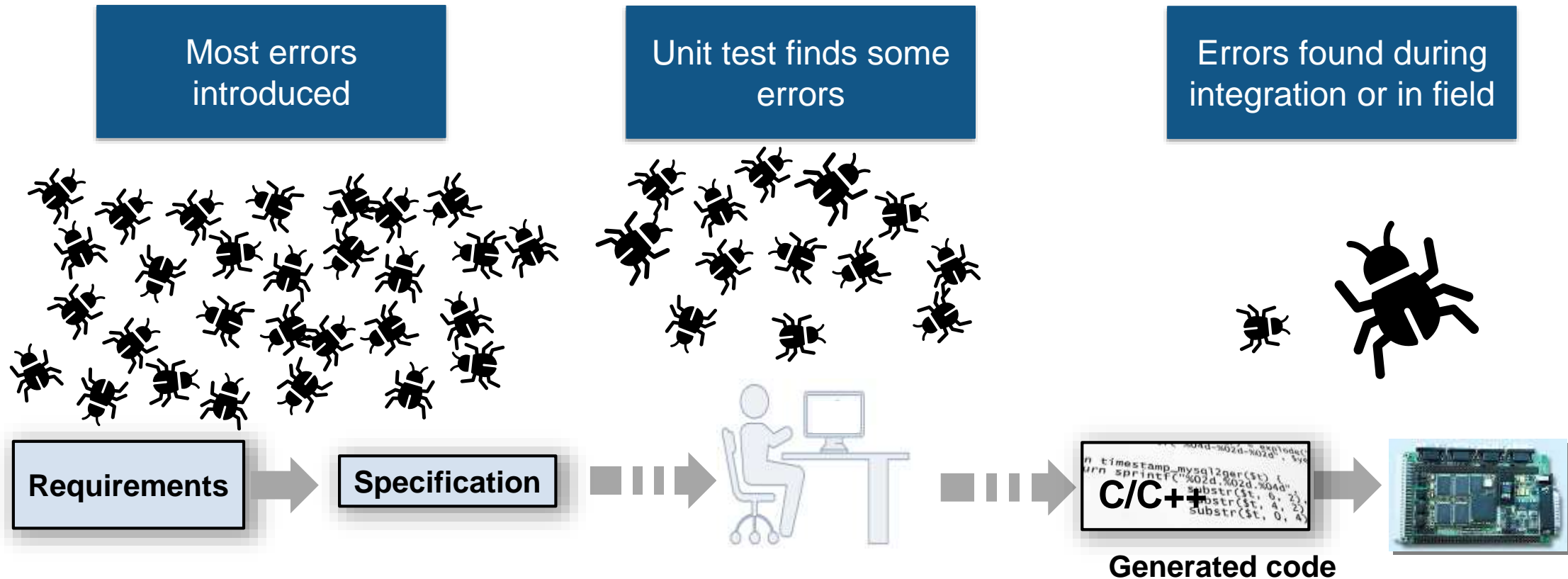
## Key takeaways

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

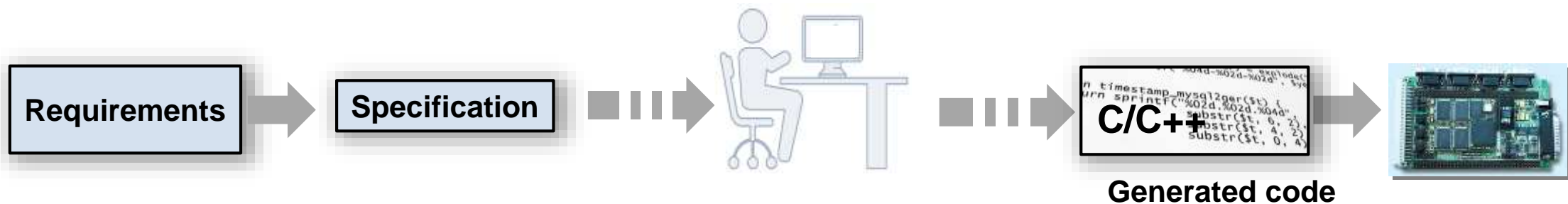
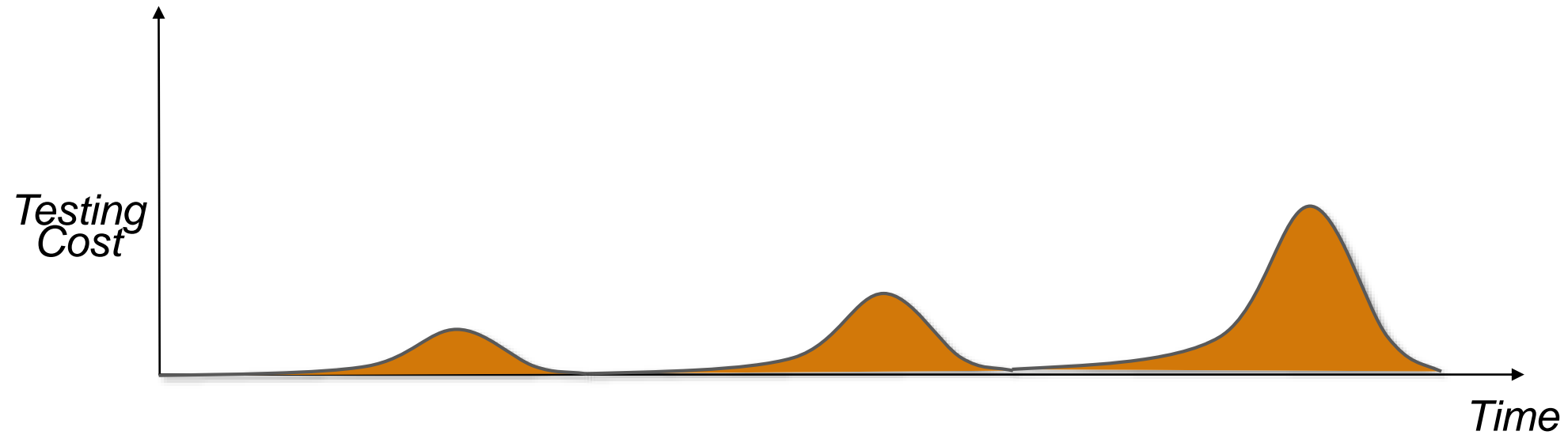
*“By enabling us to analyze requirements quickly, reuse designs from previous products, and eliminate manual coding errors, Model-Based Design has reduced development times and enabled us to shorten schedules to meet the needs of our customers.”*

*- MyoungSuk Ko, LS Automotive*

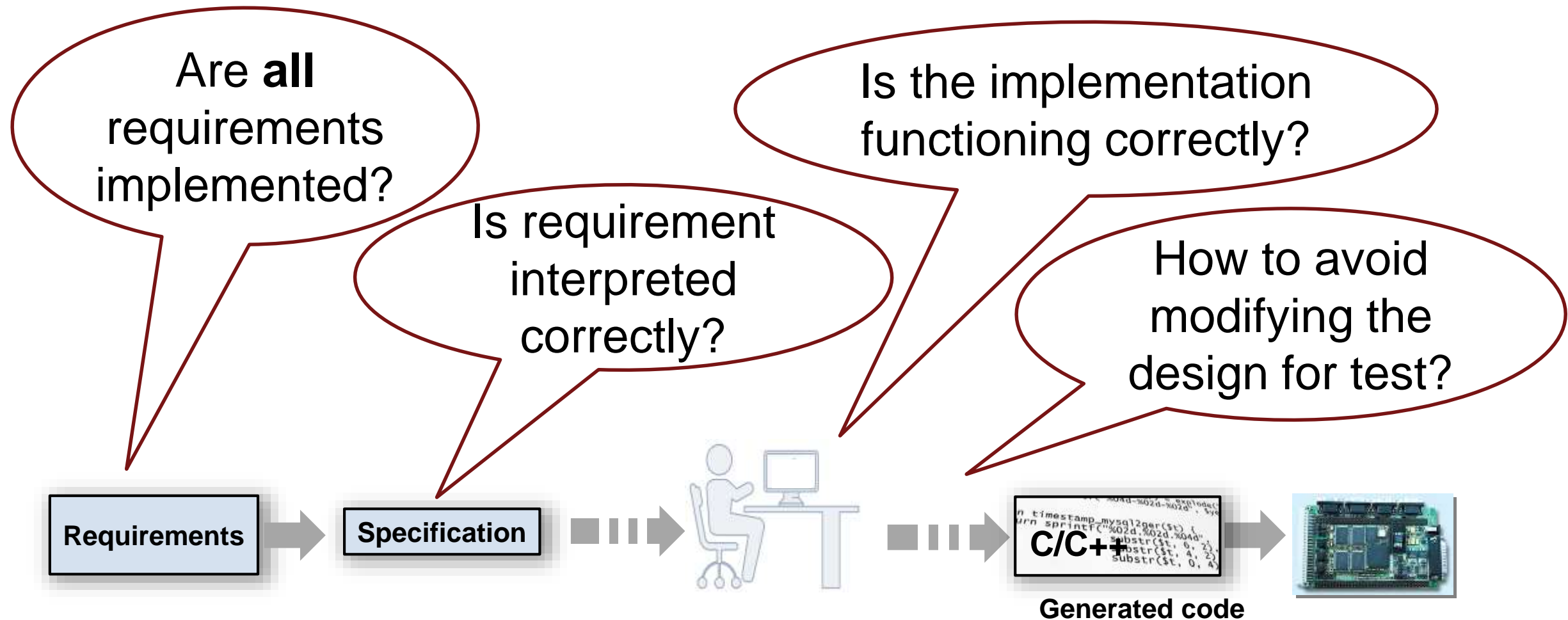
# Challenge: Errors introduced early but found late



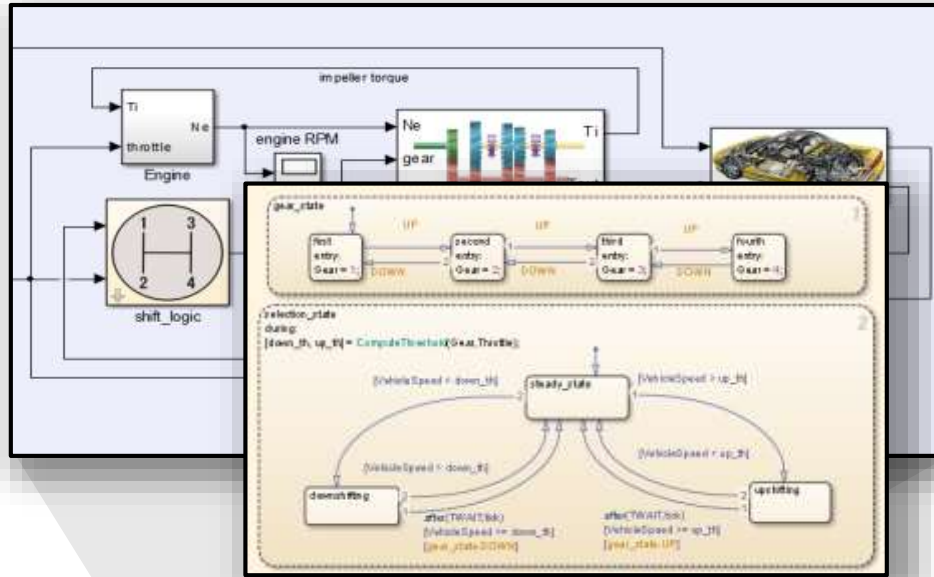
# Cost of finding errors increases over time



# 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*

Requirements

Design Model

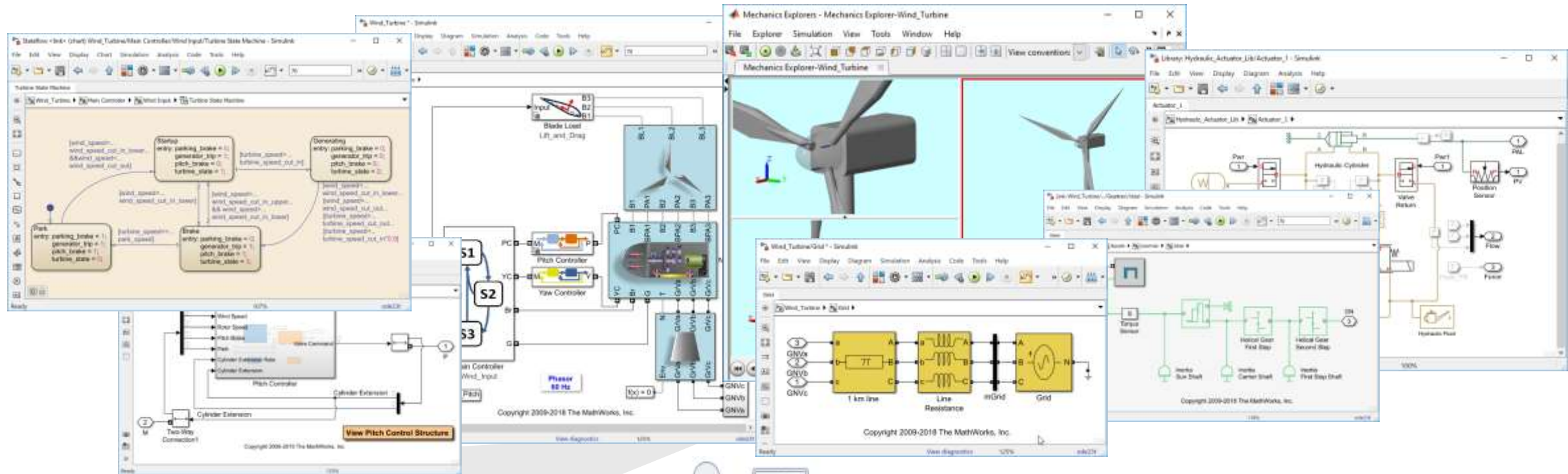


```
n timestamp_mysql2ger(st) {\n  printf("%02d.%02d.%04d",\n    substr(st, 6, 2),\n    substr(st, 4, 2),\n    substr(st, 0, 4)\n  )\n}
```

Generated code



# Multiple languages to describe complex systems



Requirements

Design Model



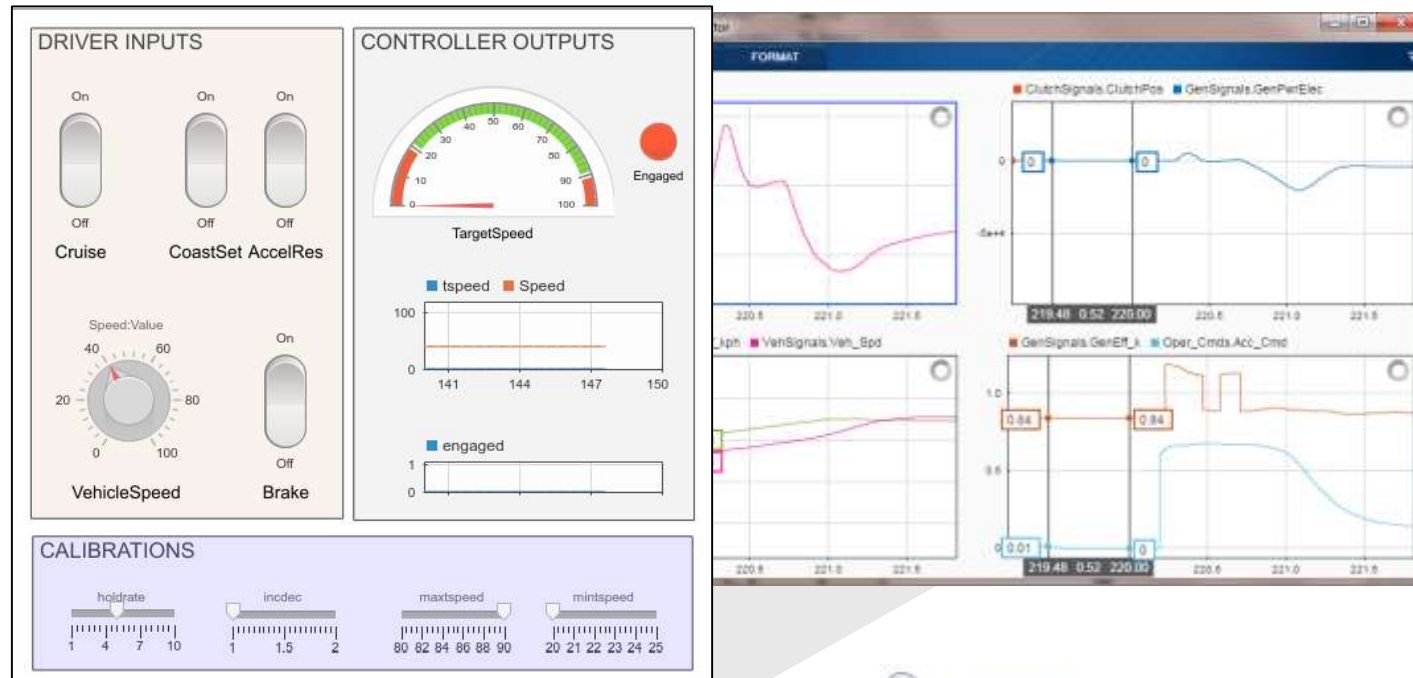
C/C++

Generated code





# Ad-Hoc Testing: Explore behavior and design alternatives



Requirements

Design  
Model



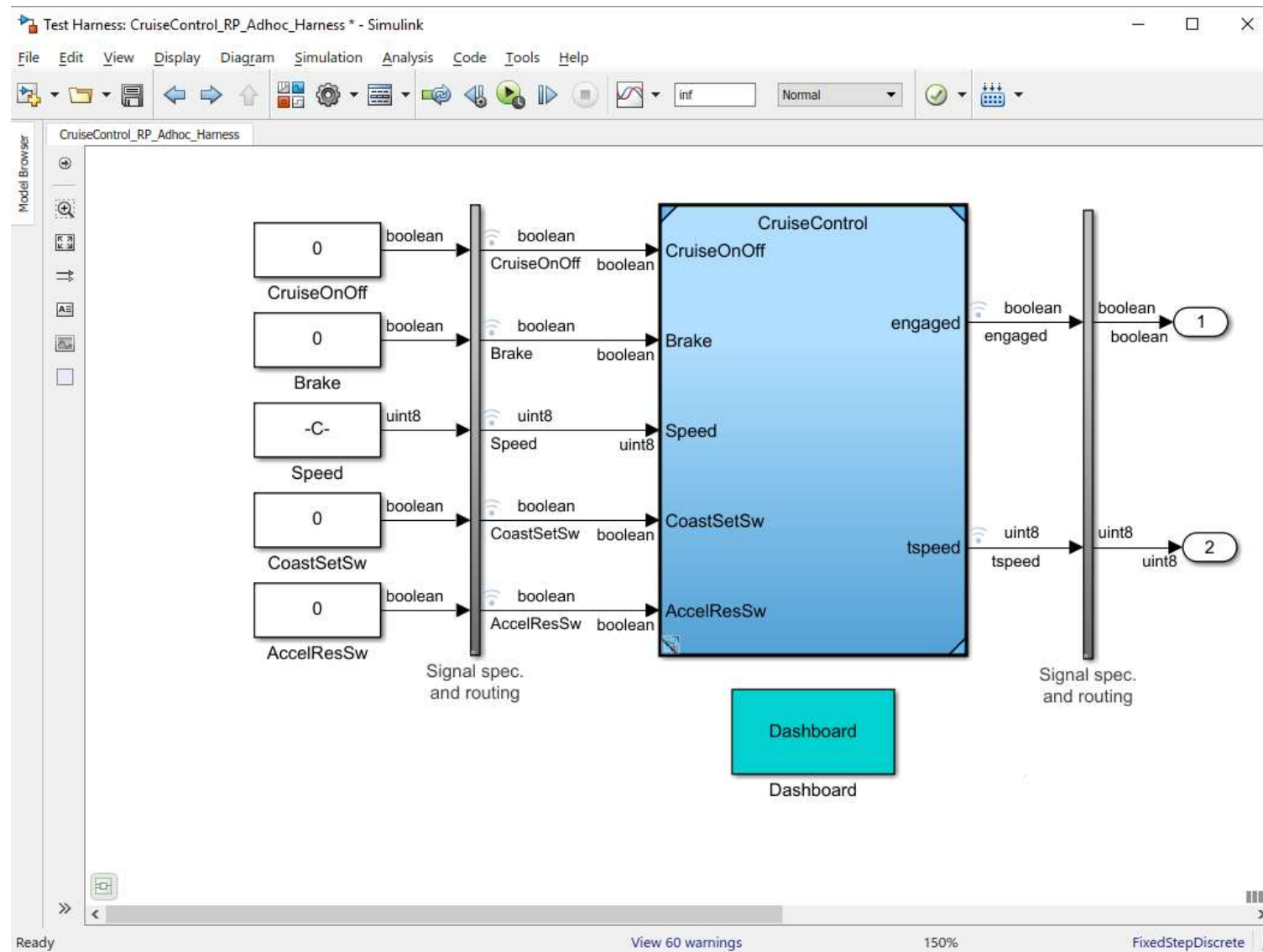
**C/C++**

Generated code

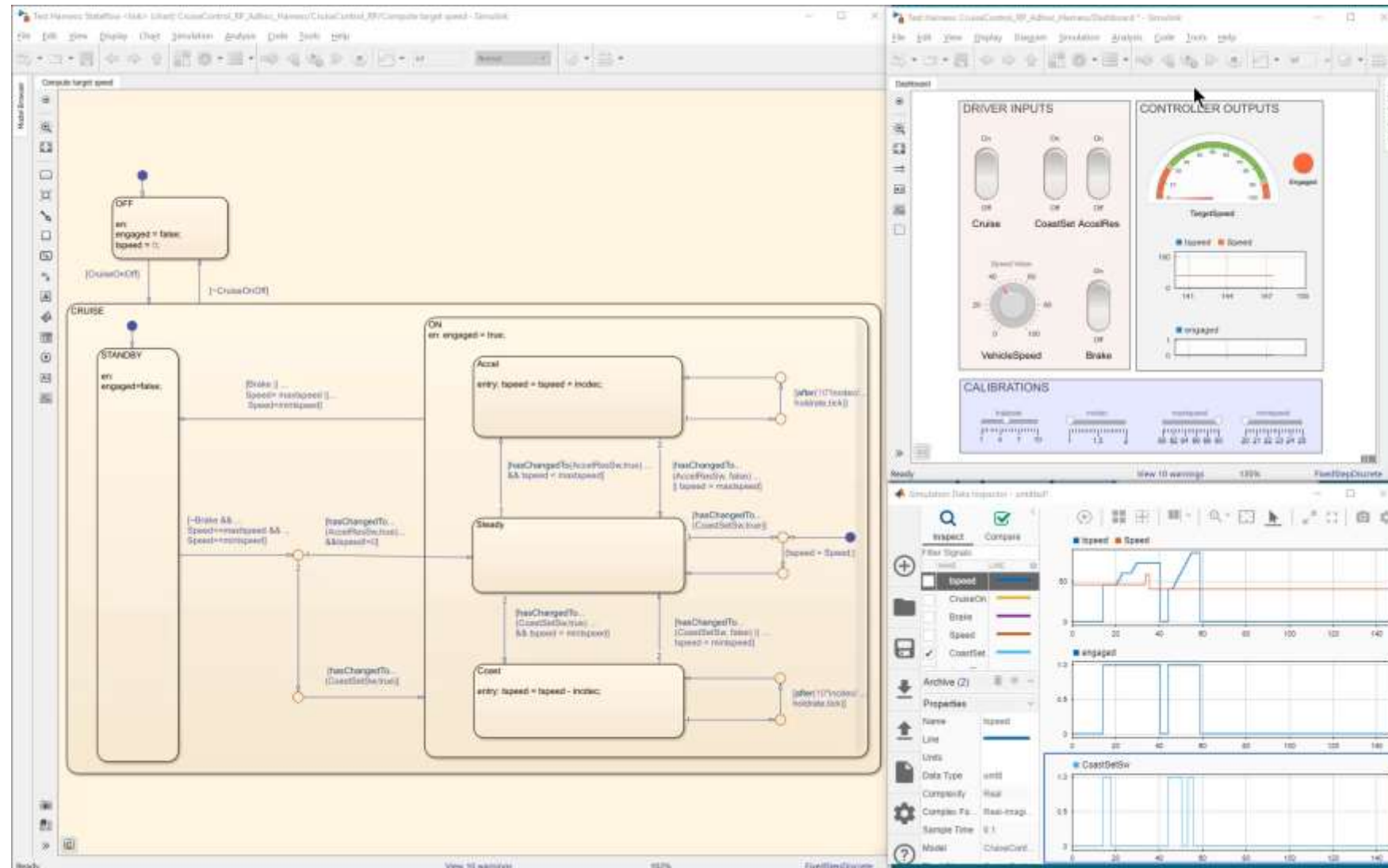




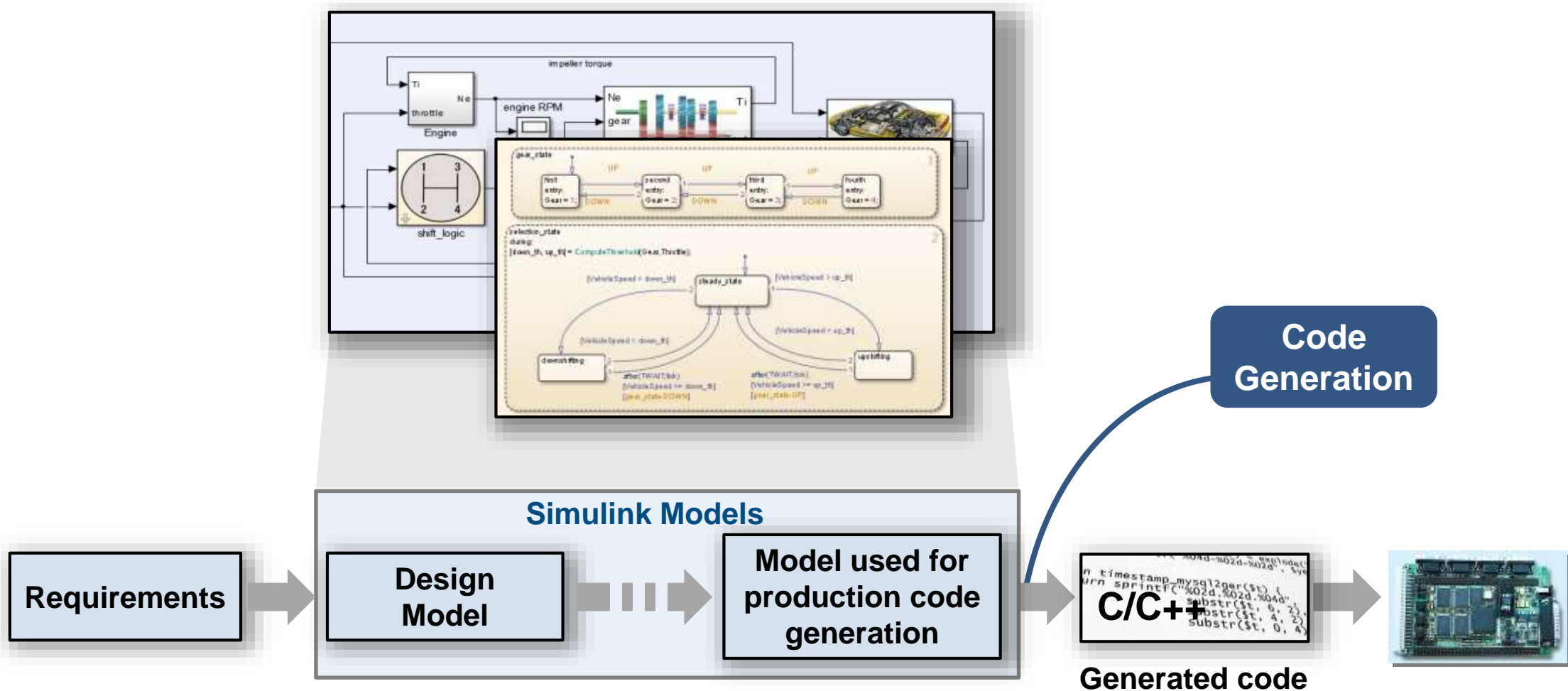
# Validate behavior earlier with simulation



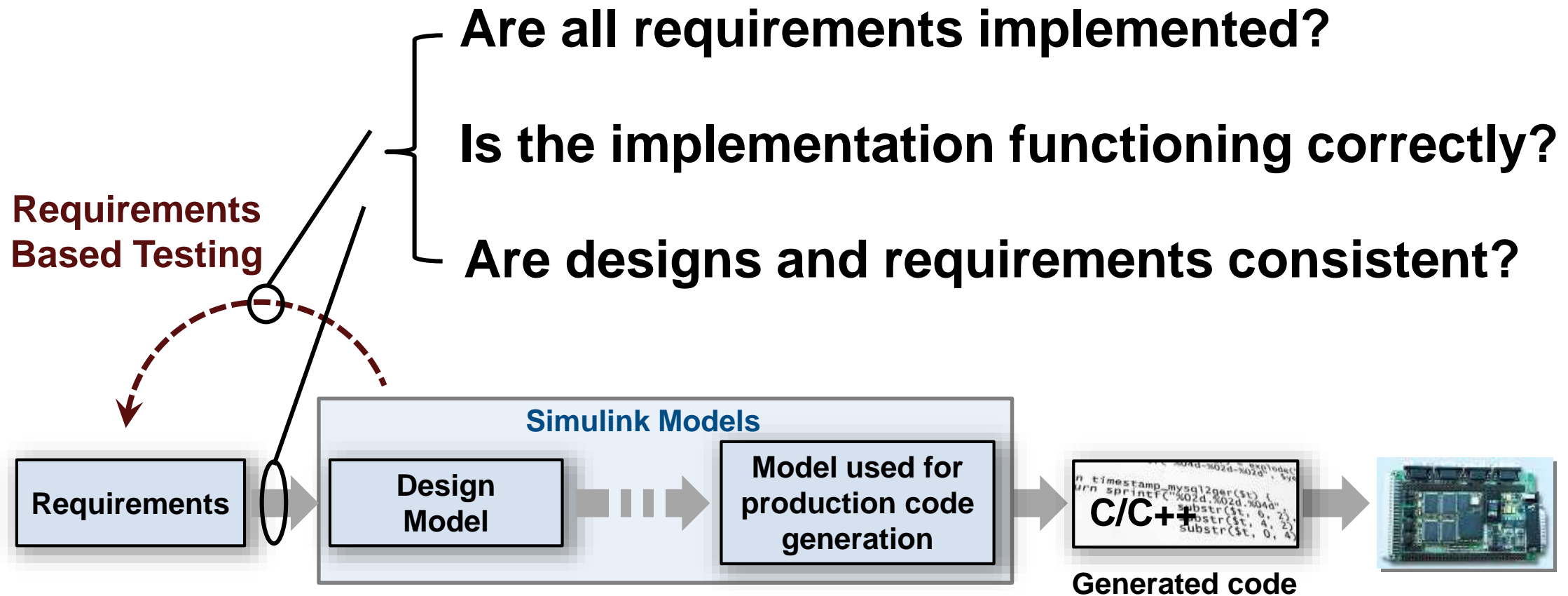
# Validate Behavior Earlier with Simulation



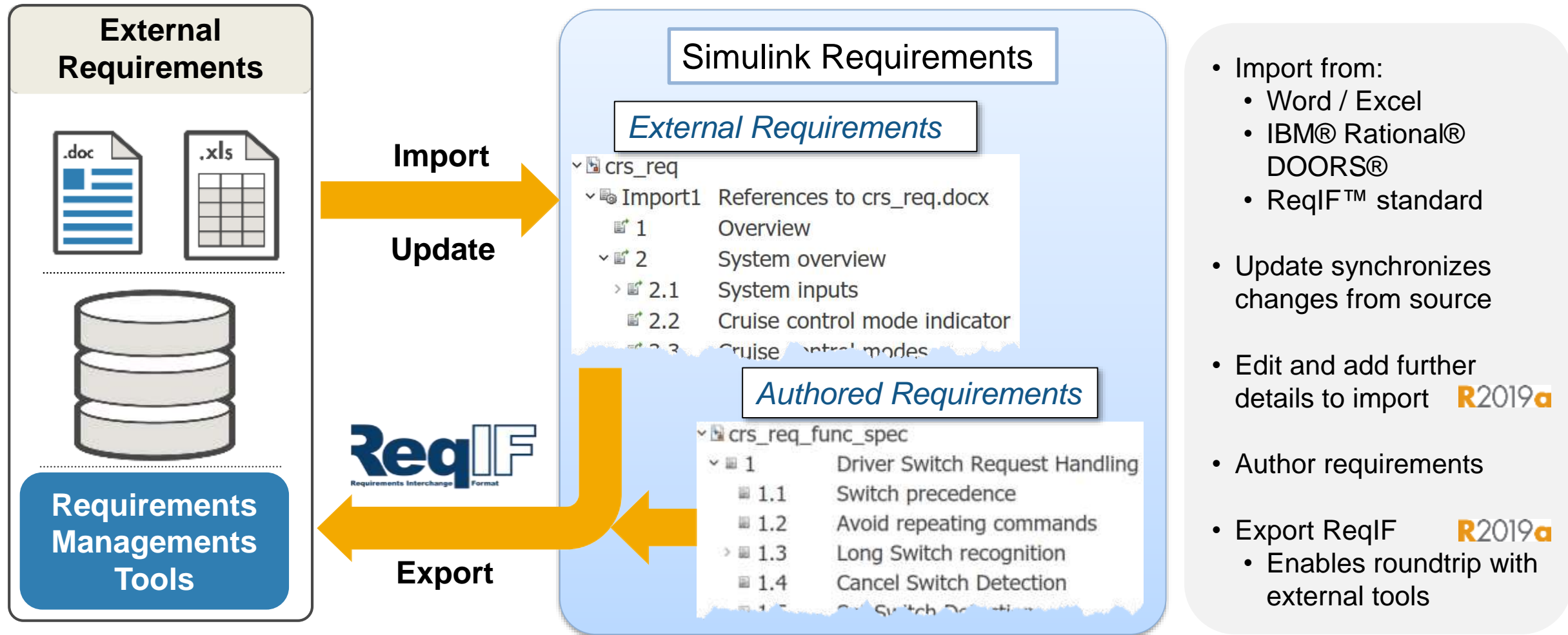
# Complete Model Based Design



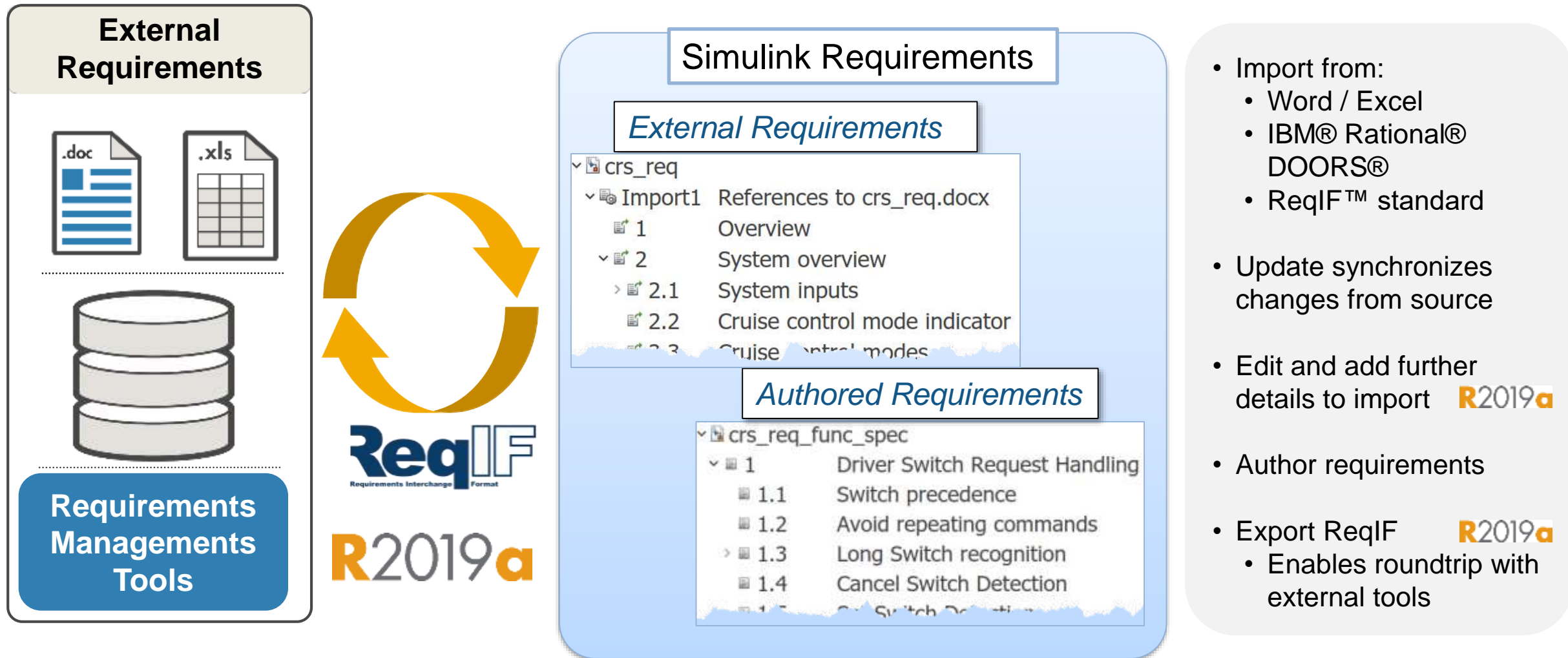
# Systematically verify requirements



# Integrate with requirements tools and author requirements



# Roundtrip workflow with external tools thru ReqIF





# Requirements Verification with Simulink

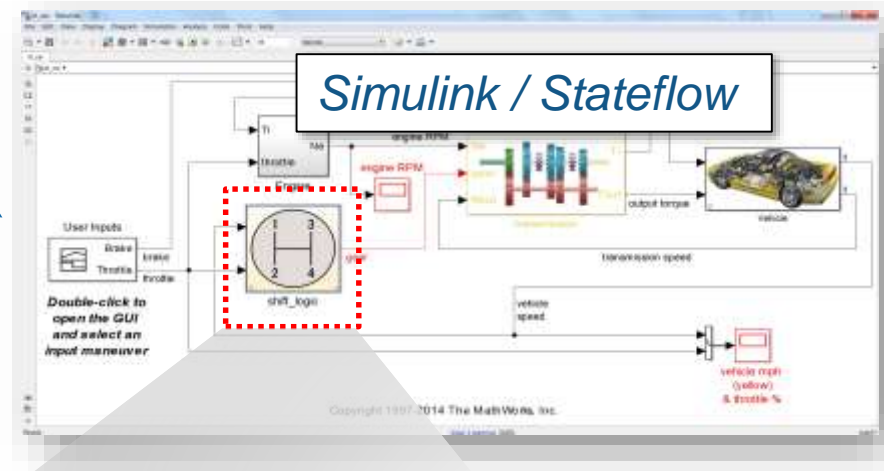
## Requirements

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



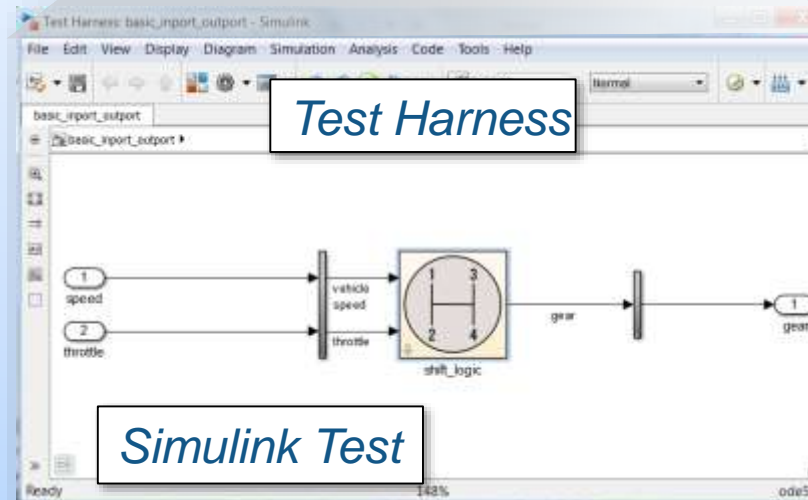
MAT / Excel  
file (input)



Signal Editor



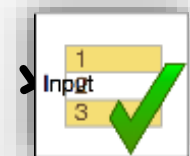
Test Sequence



## Assessments



MAT / Excel  
File (baseline)



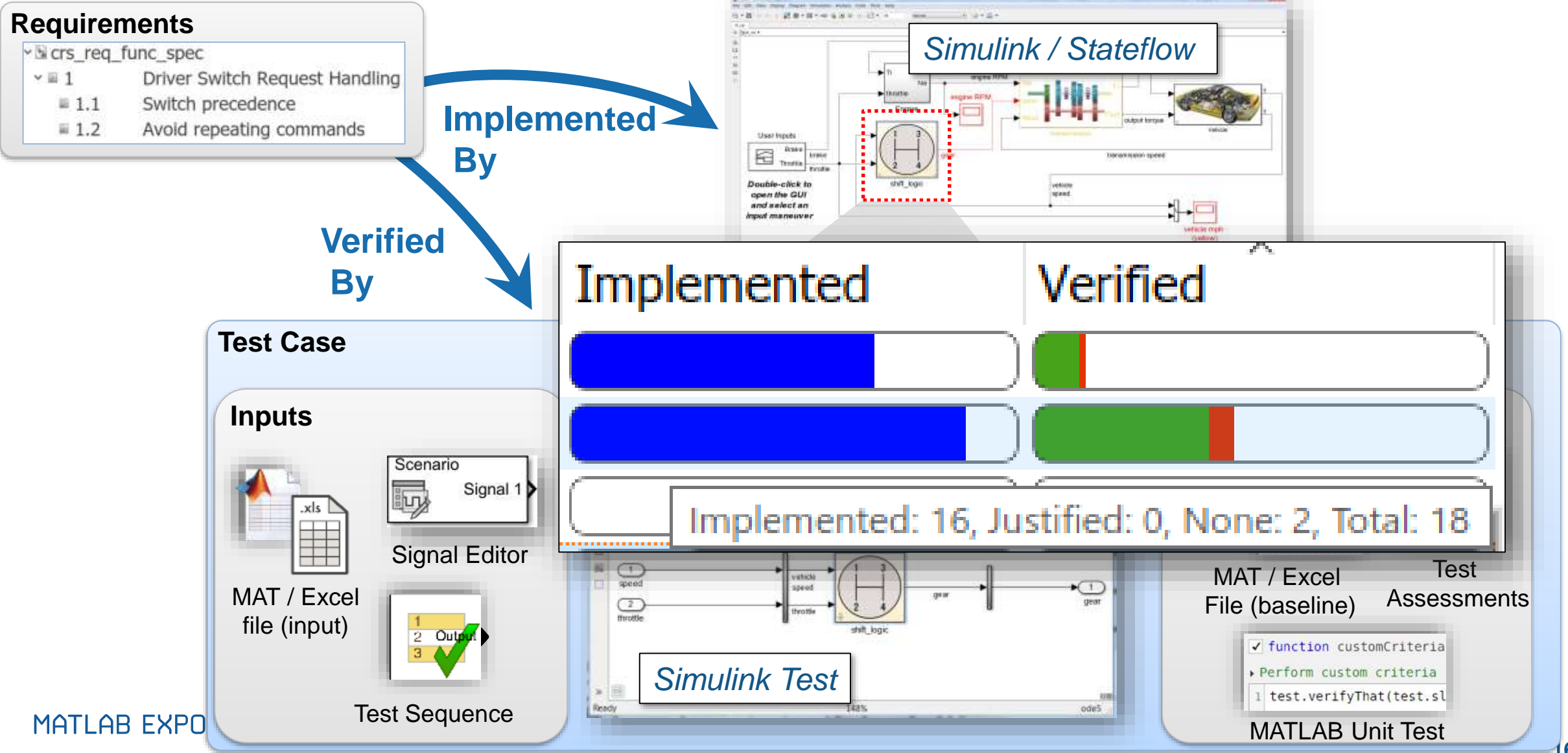
Test  
Assessments

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

MATLAB Unit Test



# Requirements Verification with Simulink



# 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.

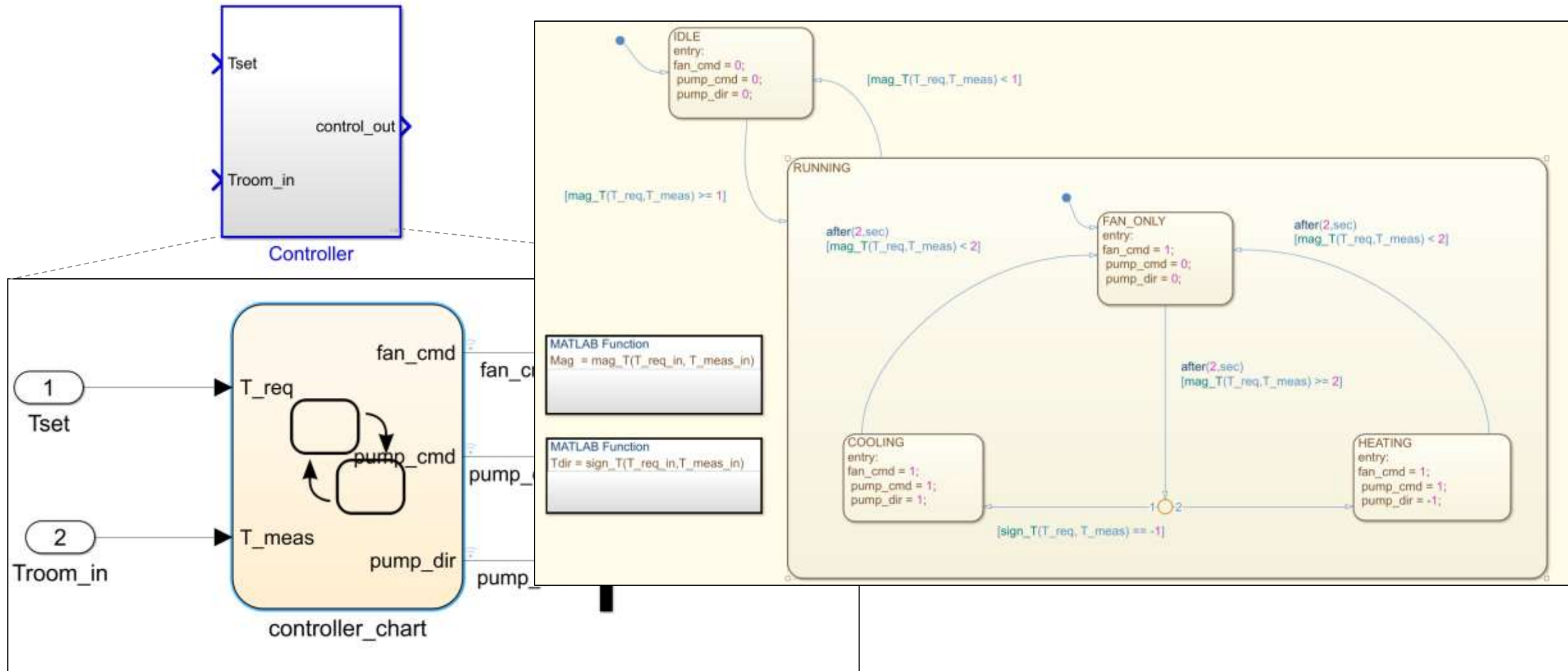
**1.3.1 Cool Mode**  
If the room temperature is greater than the set temperature, the system shall cool the space.

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

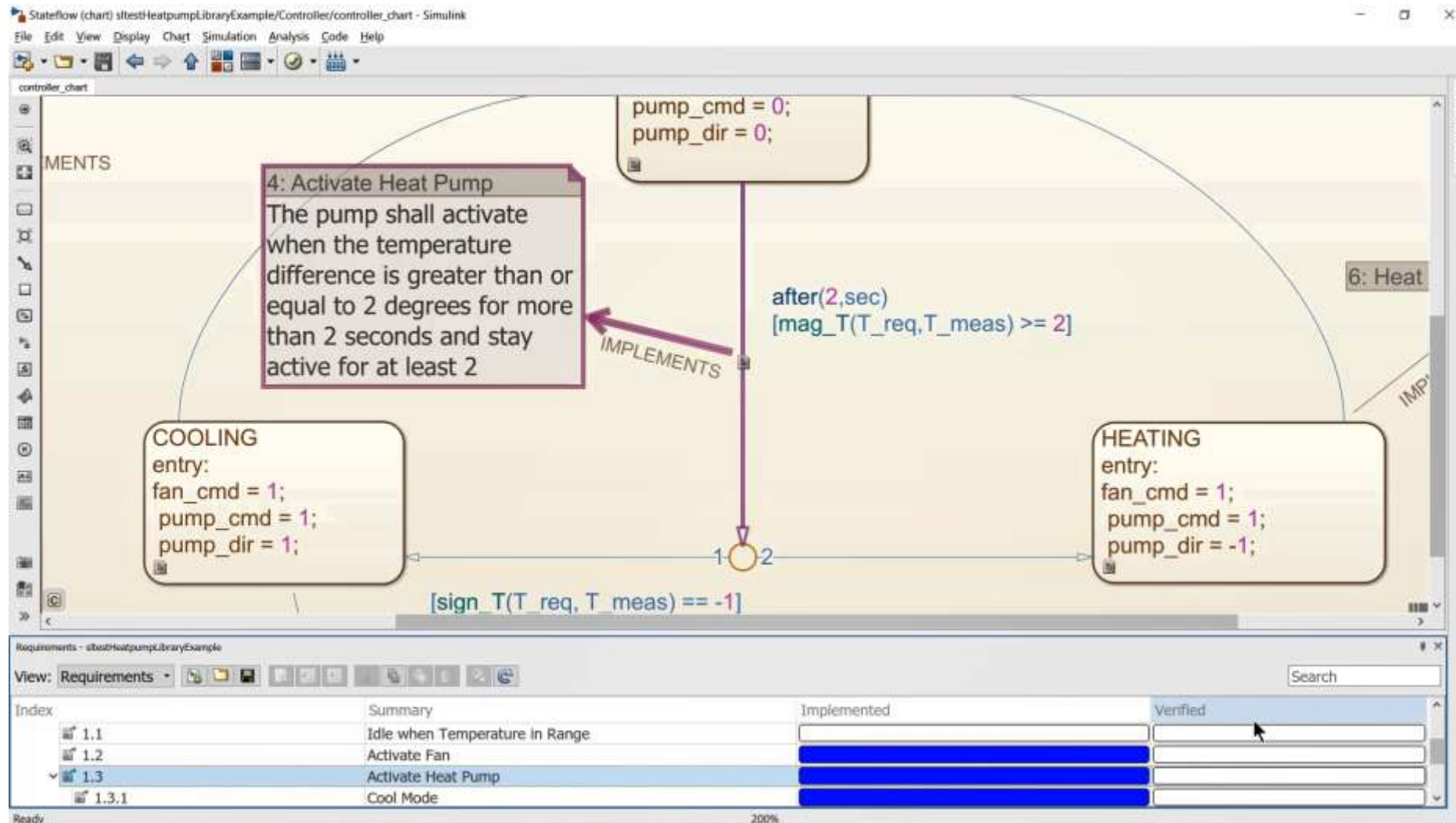
**1.4 Max Temperature Difference**  
The difference between the room temperature and the set temperature should never exceed 6 degrees.

*Requirements in DOORS*

# Example: Heat Pump Controller Implementation

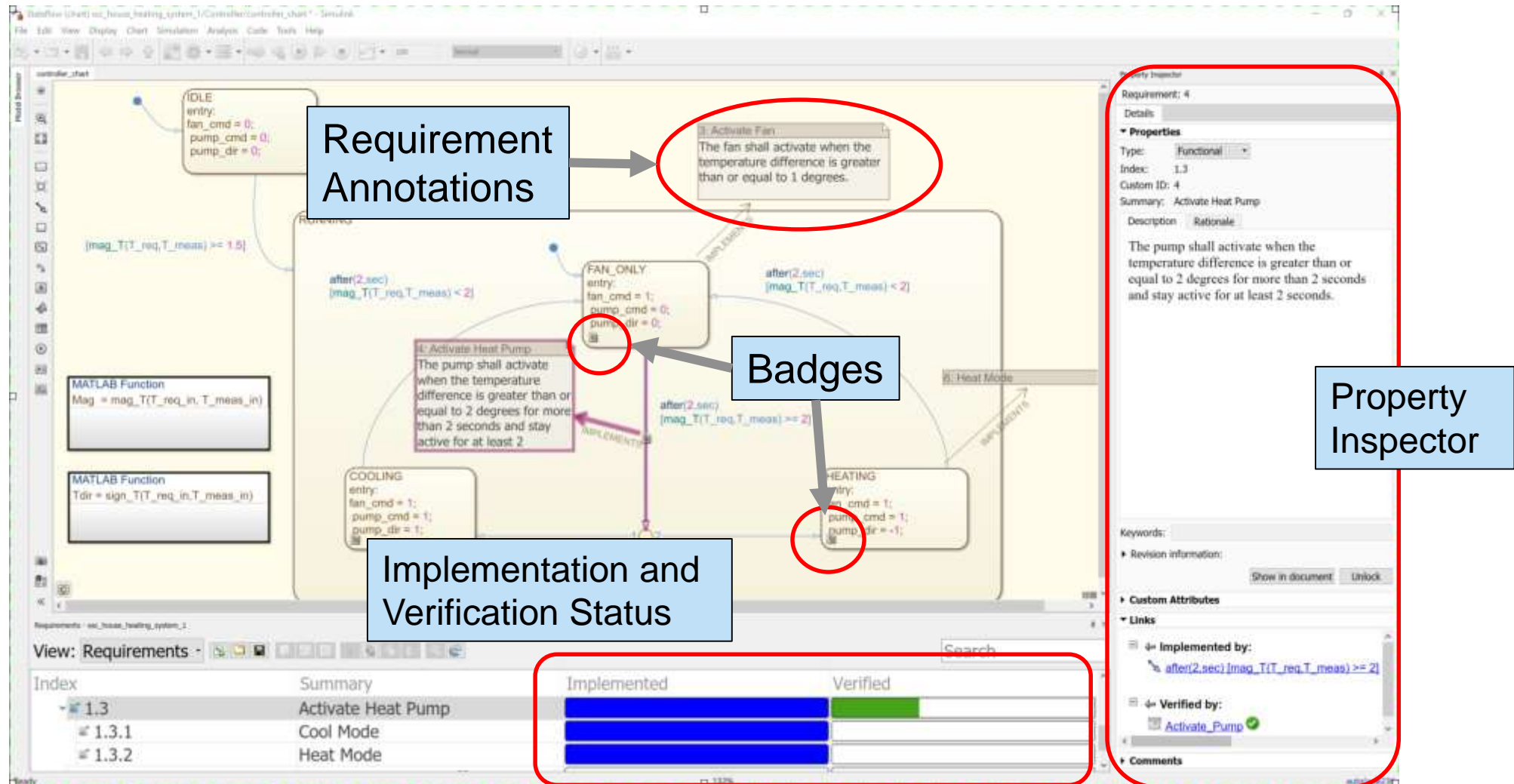


# Link requirements to implementation in model

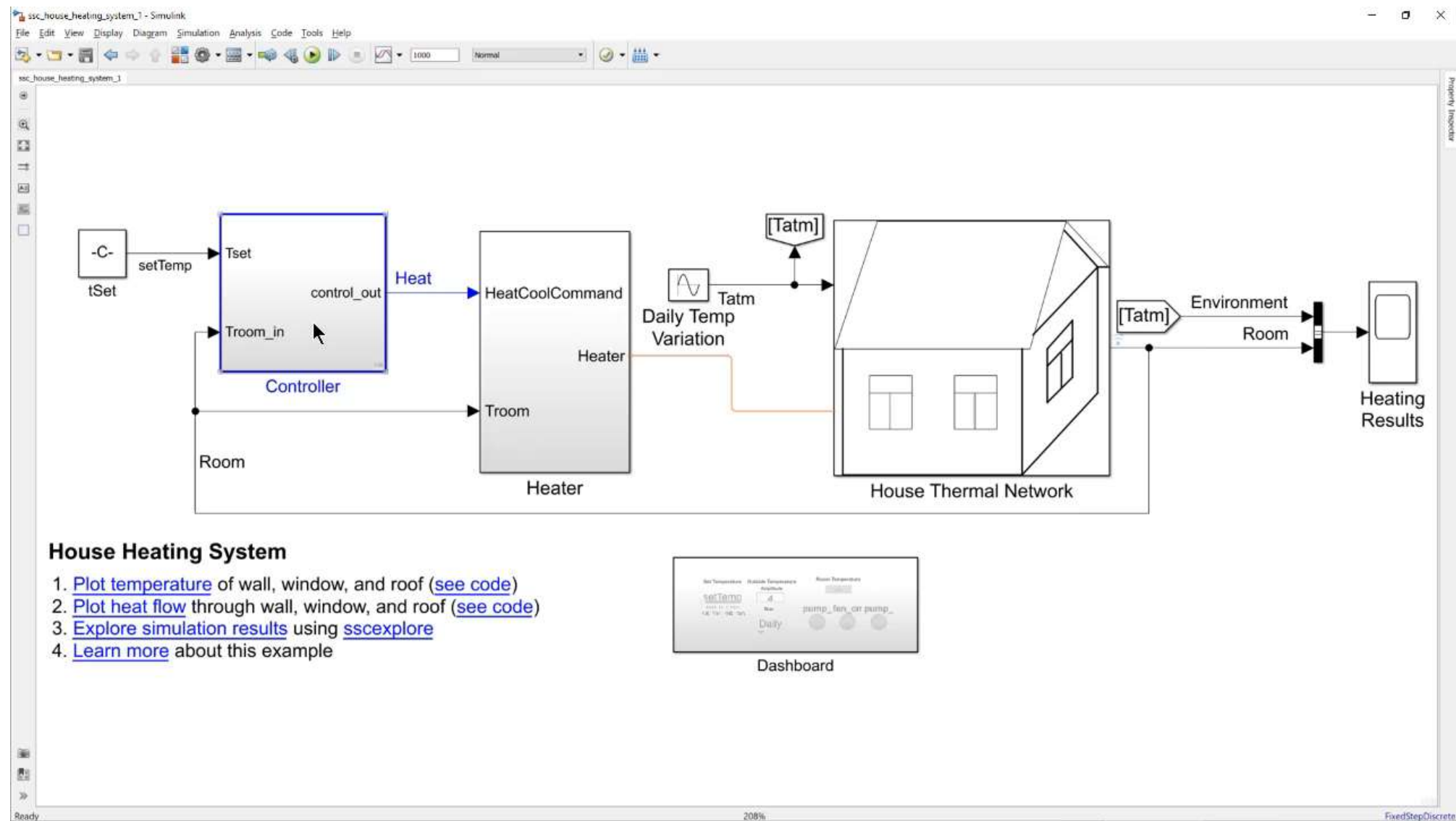




# Work with Model and Requirements with Requirements Perspective



# Isolate Component Under Test with Test Harness



# Test Sequence Block: Step-based and temporal test sequences

The screenshot displays the MATLAB Test Sequence Editor interface. The left sidebar contains a 'Symbols' panel with categories: Input (control\_out), Output (Tset, Troom\_in), Local, Constant, Parameter, and Data Store Memory. Below this is a 'Step Hierarchy' panel showing a tree view with 'Initialize', 'Cold\_Outside', and 'Hot\_Outside'. The main workspace is a table with three columns: 'Step', 'Transition', and 'Next Step'.

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

The screenshot displays the MATLAB Simulink form-based editor for creating temporal assessments. The interface is divided into several sections:

- Simulation Test:** Includes a dropdown for "Select releases for simulation" (set to "Current") and a checkbox for "Create Test Case from External File".
- REQUIREMENTS:** A section for defining requirements.
- SYSTEM UNDER TEST:** Contains fields for "Model" (set to "ssc\_house\_heating\_system\_1") and "Harness" (set to "ssc\_house\_heating\_system\_1\_Harness1").
- TEST HARNESS:** A section for defining the test harness.
- SIMULATION SETTINGS OVERRIDES:** A section for defining simulation settings overrides.
- INPUTS:** A section for defining inputs.
- SIMULATION OUTPUTS:** A section for defining simulation outputs.
- LOGICAL AND TEMPORAL ASSESSMENTS:** A table for defining assessments.

The "LOGICAL AND TEMPORAL ASSESSMENTS" table has the following columns: EN, NAME, ASSESSMENT, REQUIREMENTS, and VISUAL REPRESENTATION. The first row is selected:

EN	NAME	ASSESSMENT	REQUIREMENTS	VISUAL REPRESENTATION
✓	Activate_Pump	At any point of time, if $\text{abs}(\text{roomTemperature} - \text{setTemperature}) \geq \text{threshold}$ becomes true and stays true for at least 2 seconds then, starting from rising edge of trigger, with a delay of at most 2 seconds, $\text{pumpCmd}$ must stay true for at least 2 seconds	None	

The "VISUAL REPRESENTATION" section shows a timing diagram with two signals: TRIGGER and RESPONSE. The TRIGGER signal is true for a duration of min-time to max-time. The RESPONSE signal is true for a duration of min-time, starting after a delay of at most 2 seconds from the rising edge of the TRIGGER signal.

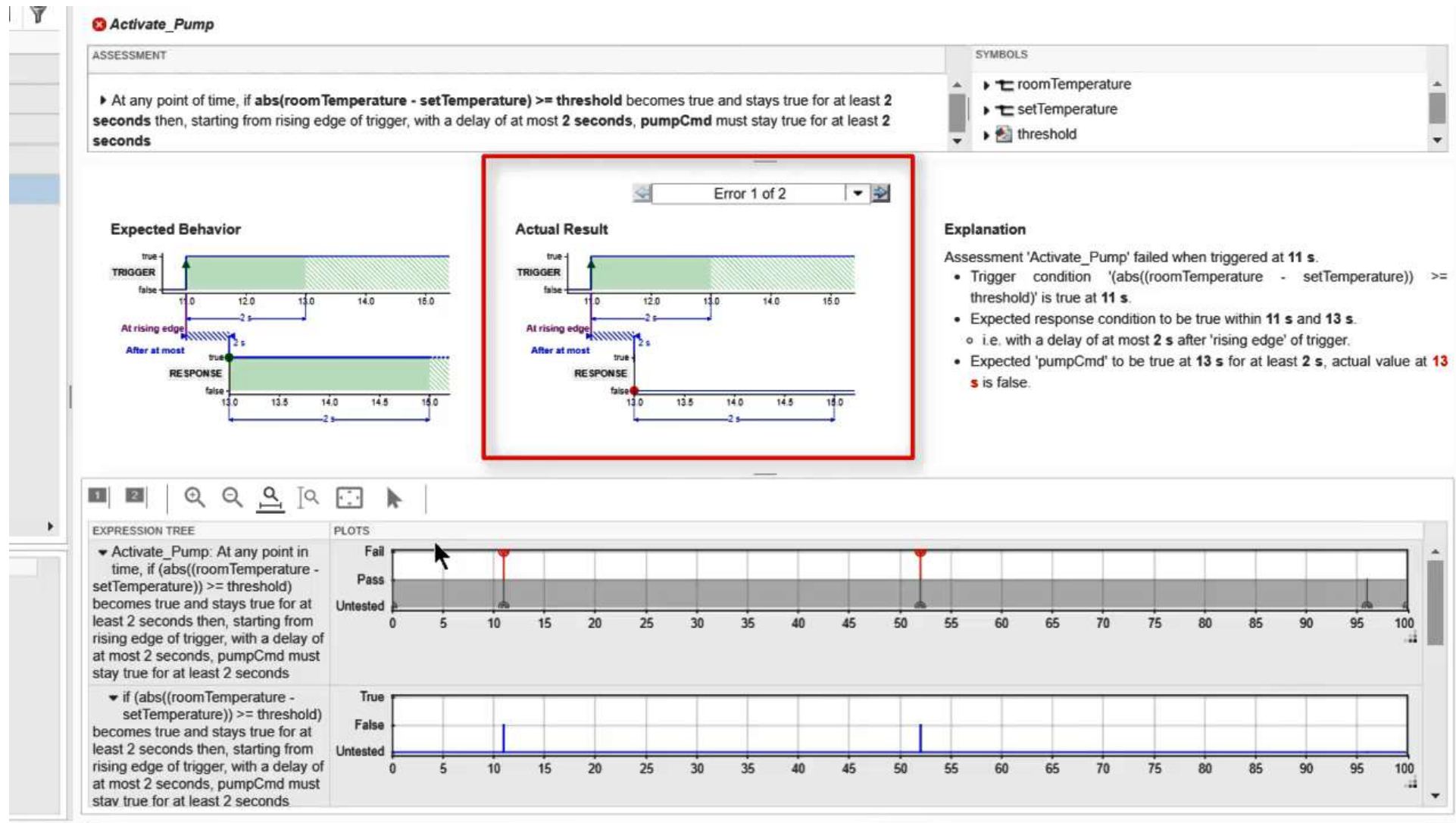
The "SYMBOLS" section lists the variables used in the assessment:

- 1 roomTemperature
- 2 setTemperature
- 3 threshold
- 4 pumpCmd

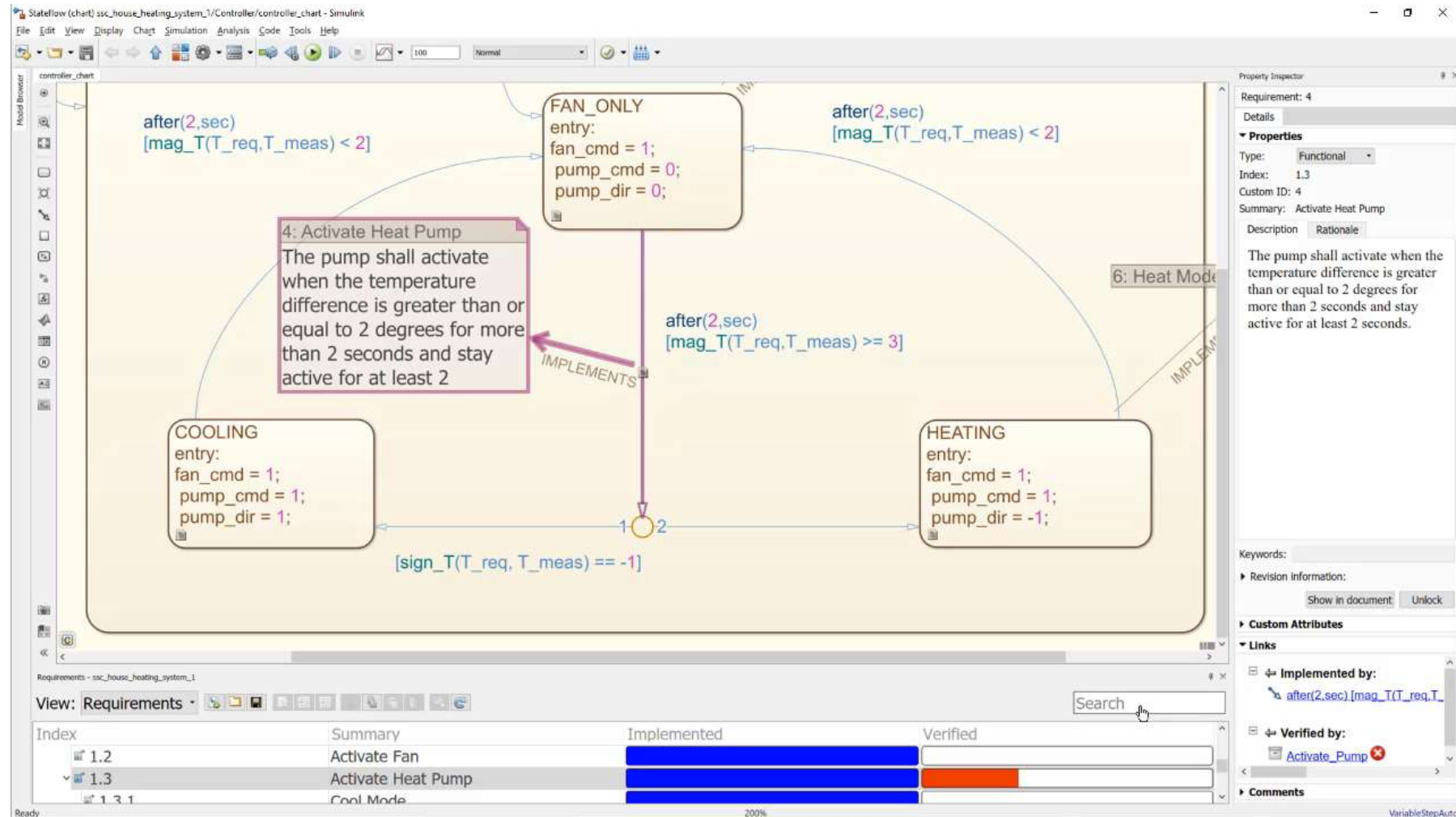
At the bottom of the editor, there are buttons for "Add Assessment", "Delete", "Add Symbol", and "Delete".

# Execute assessments to verify requirements

R2019a

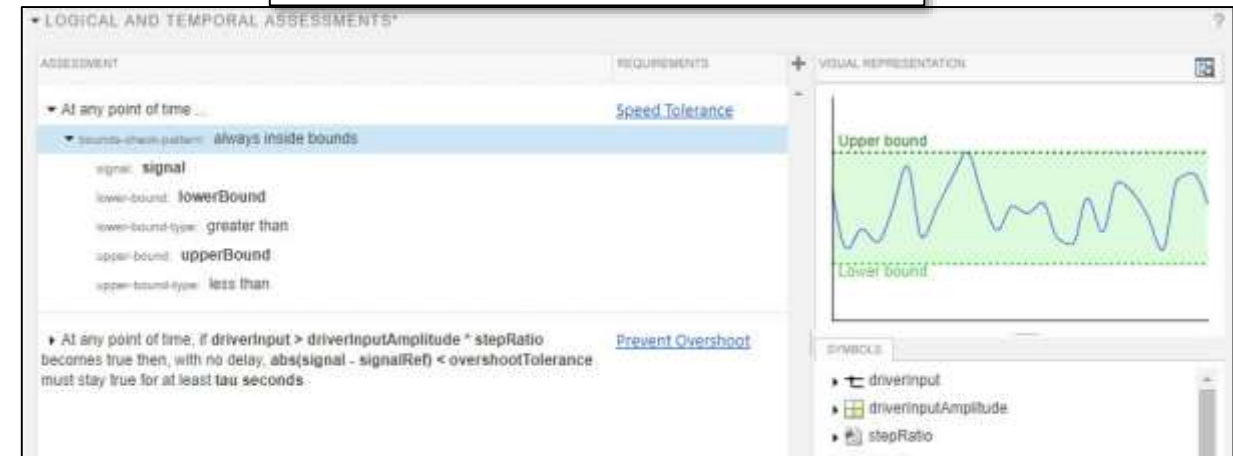


# Locate implementation of requirement using link

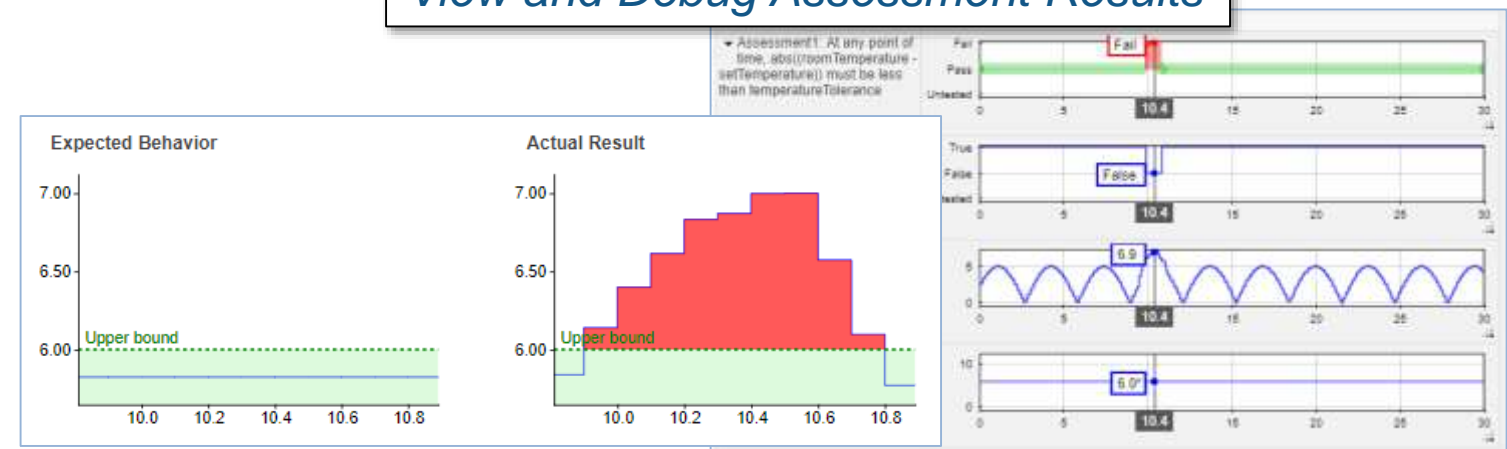


# Translate textual requirements into unambiguous Temporal Assessments

## Temporal Assessment Editor



## View and Debug Assessment Results



# 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

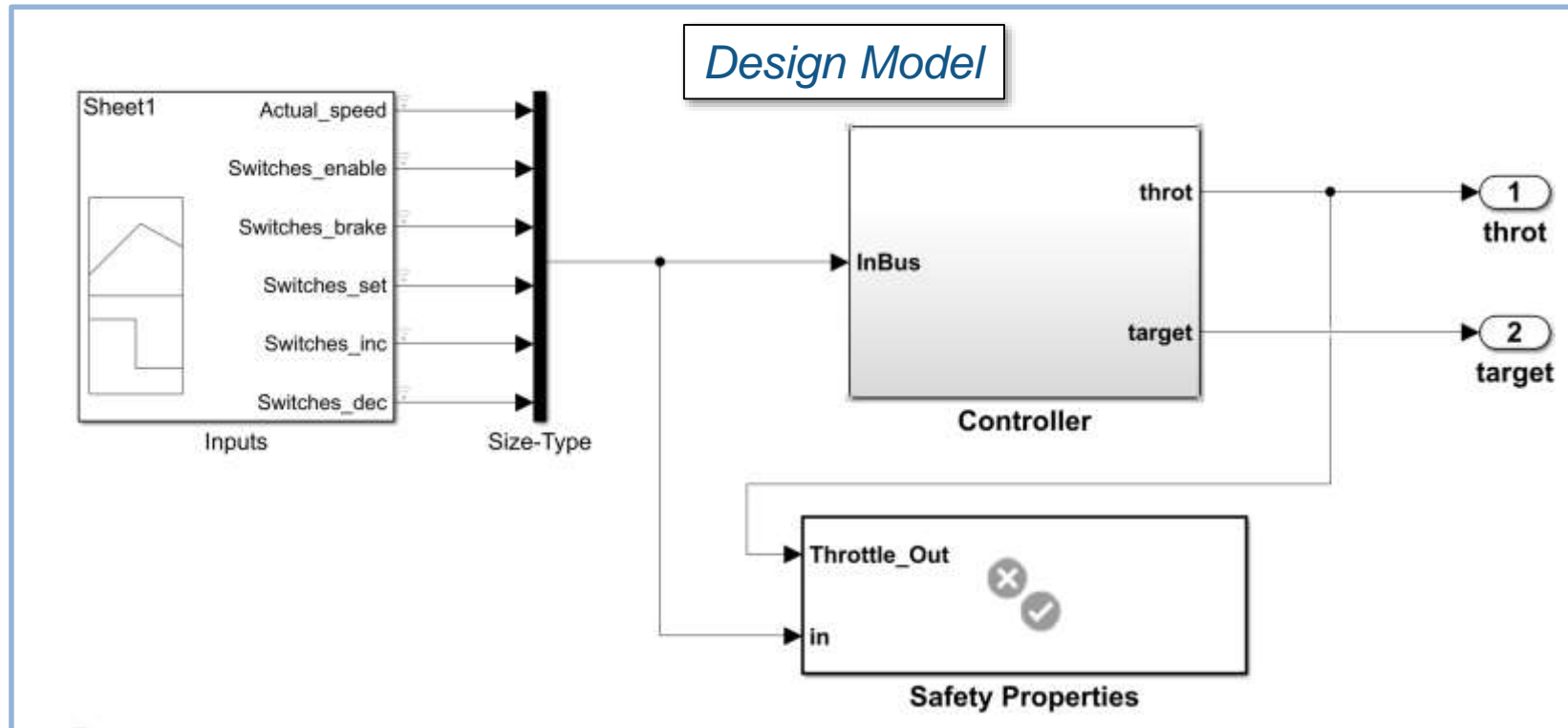
- Implemented
- Justified
- Missing

## Verification Status

- Passed
- Failed
- Unexecuted
- Missing

# Observers: Separate test/verification logic from design

R2019a

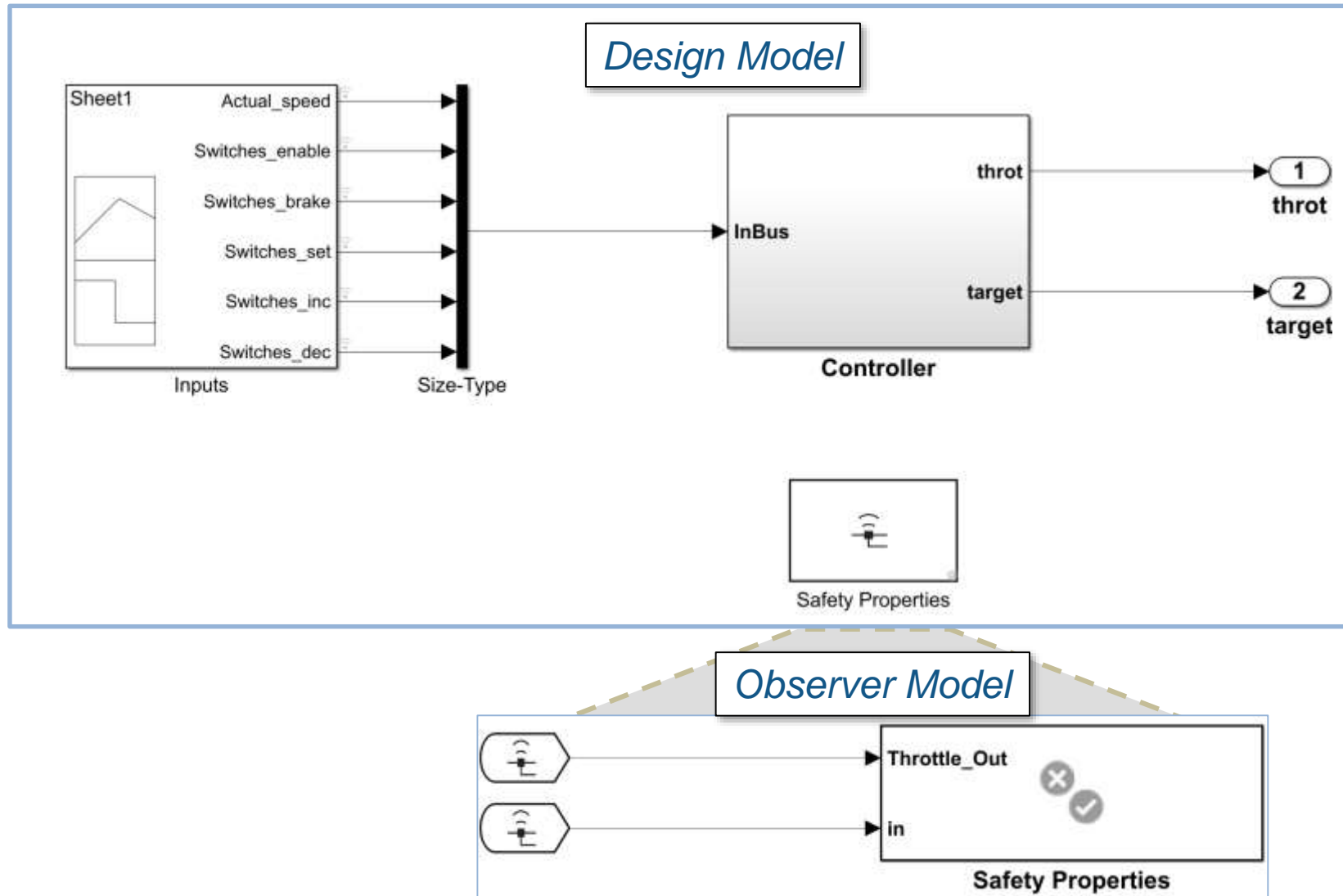


- Access nested signals without signal lines or changing dynamic response
- Avoid modifying interface for testing
- Simplify design and test by avoiding additional signal lines



# 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

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

## Challenge

Shorten development times for embedded control software used in automotive switches and components

## Solution

Use Model-Based Design to model controller designs, run simulations, verify customer specifications, and generate error-free production code

## Results

- Specification errors detected early
- Proven development approach established
- 80% Coding errors eliminated

[Link to user story](#)

MATLAB EXPO 2019



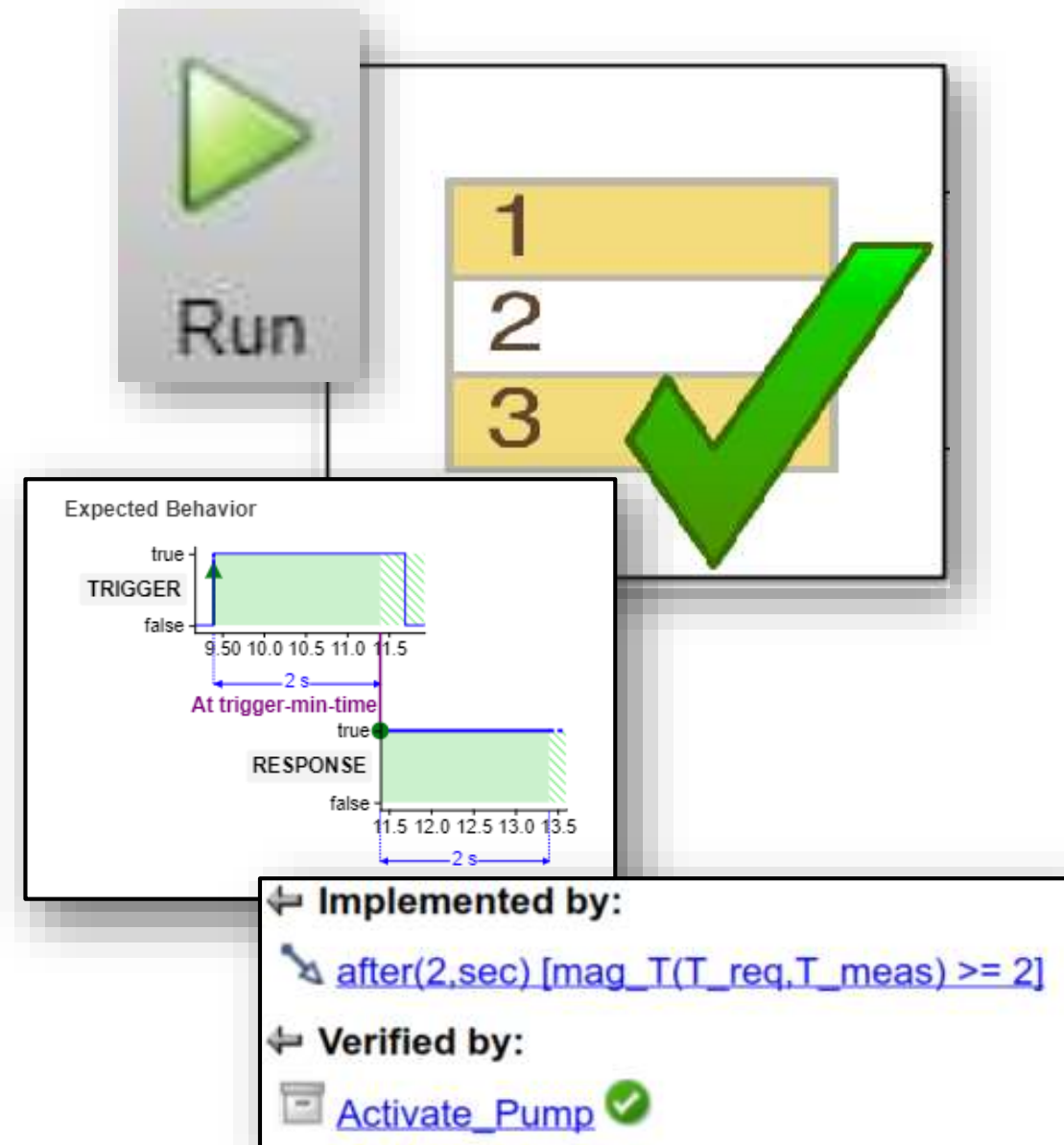
An LS Automotive door area unit.

*“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*

# Summary

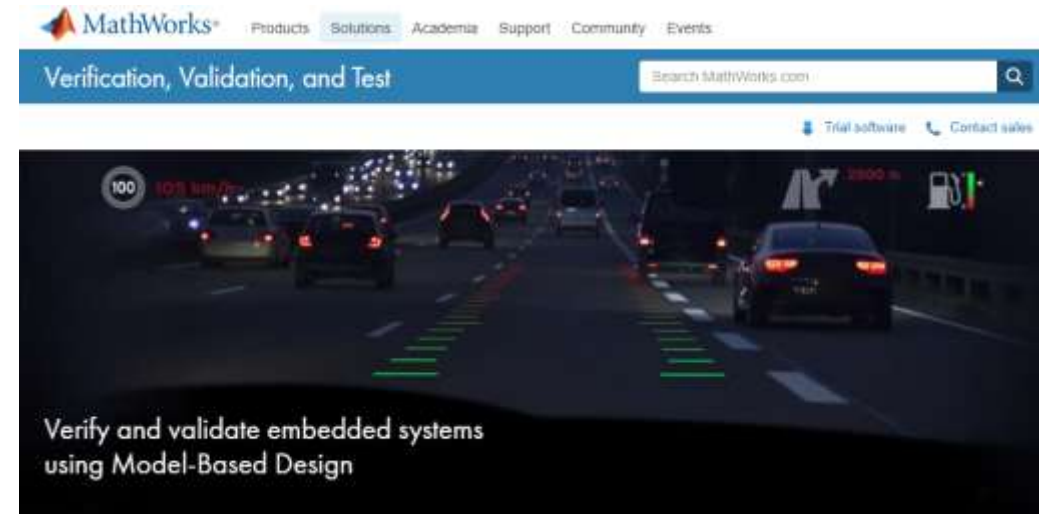
- Verify and validate requirements earlier
- Identify inconsistencies in requirements by using unambiguous assessments
- Traceability from requirements to design and test



# Learn More

Key products covered in this presentation:

- [Simulink Requirements](#)
- [Simulink Test](#)



Learn more at Verification, Validation and Test Solution Page:  
[mathworks.com/solutions/verification-validation.html](https://mathworks.com/solutions/verification-validation.html)