

MATLAB EXPO

Test Driven Development in Agile Model-Based Design

Paul Urban

Marco Dragic





Marco Dragic

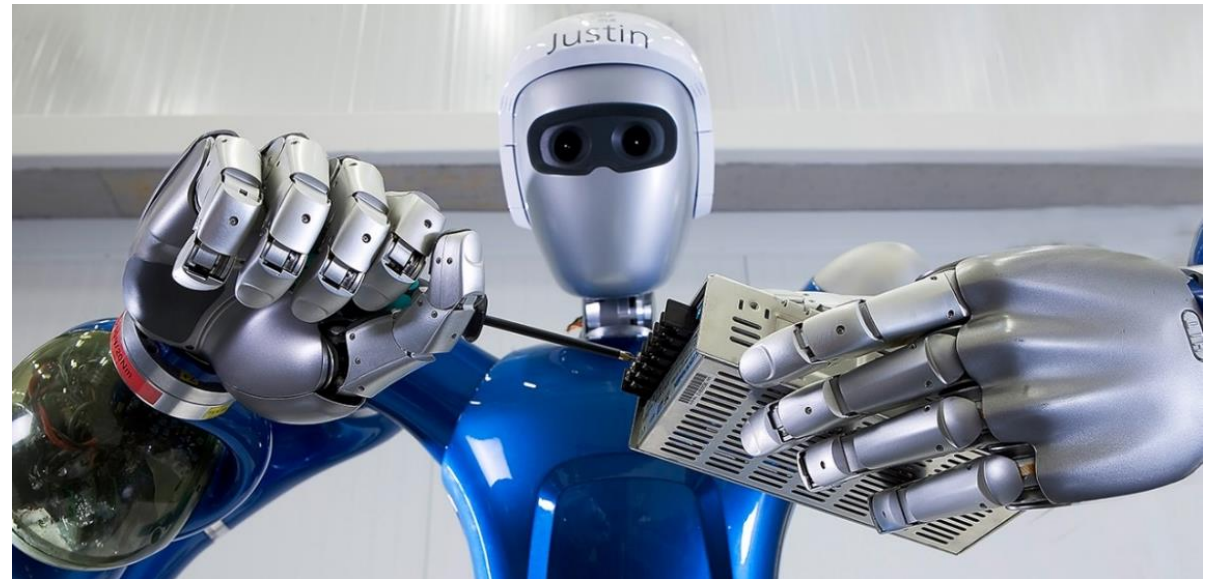
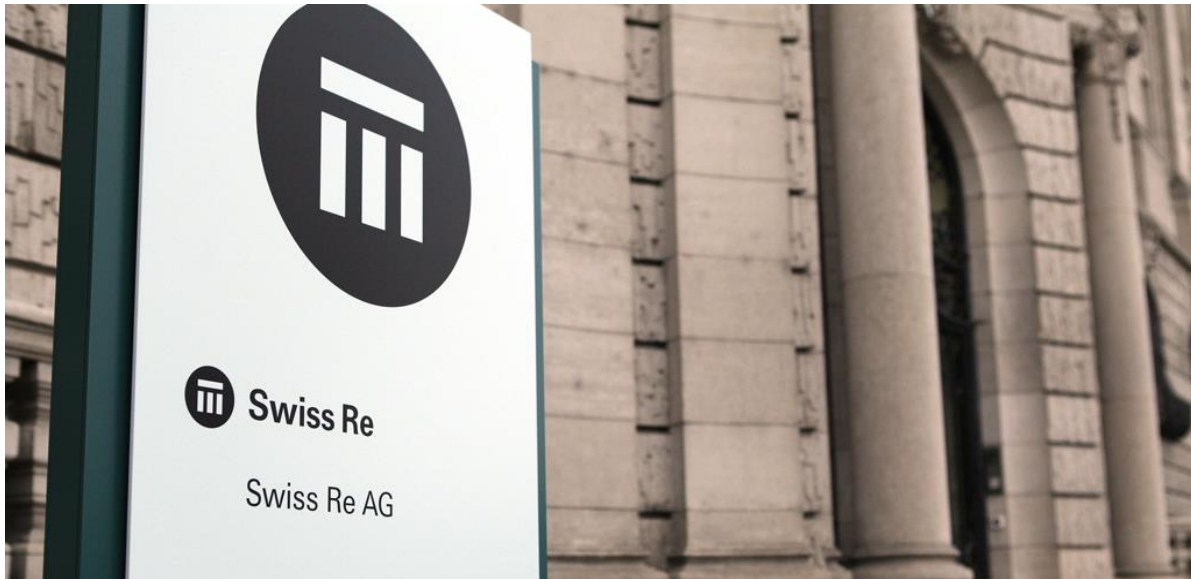
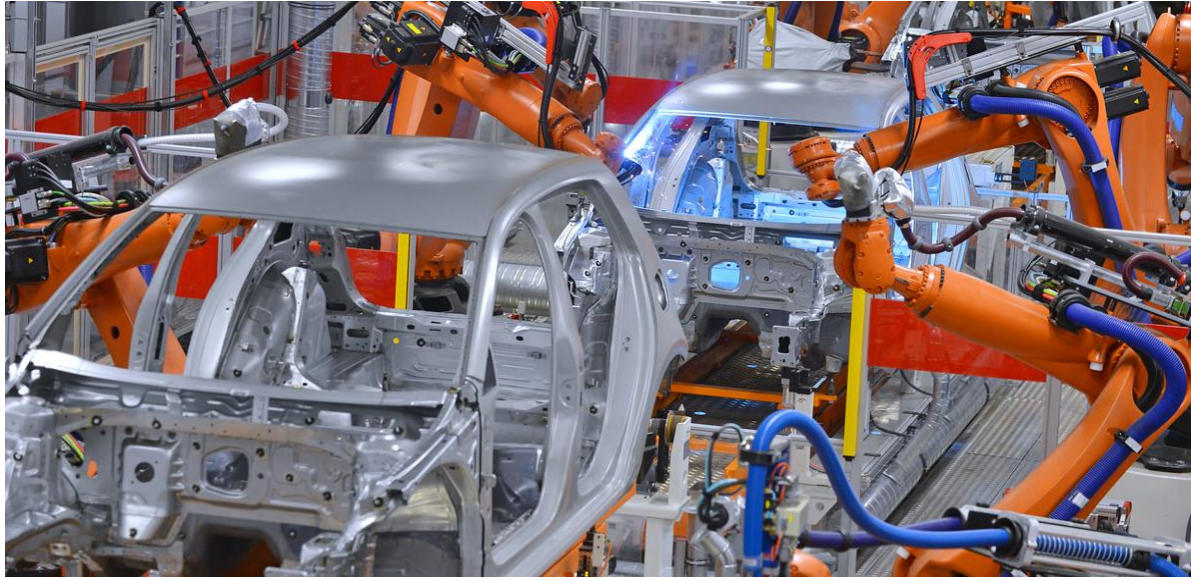
Senior Product Manager
Simulink Platform



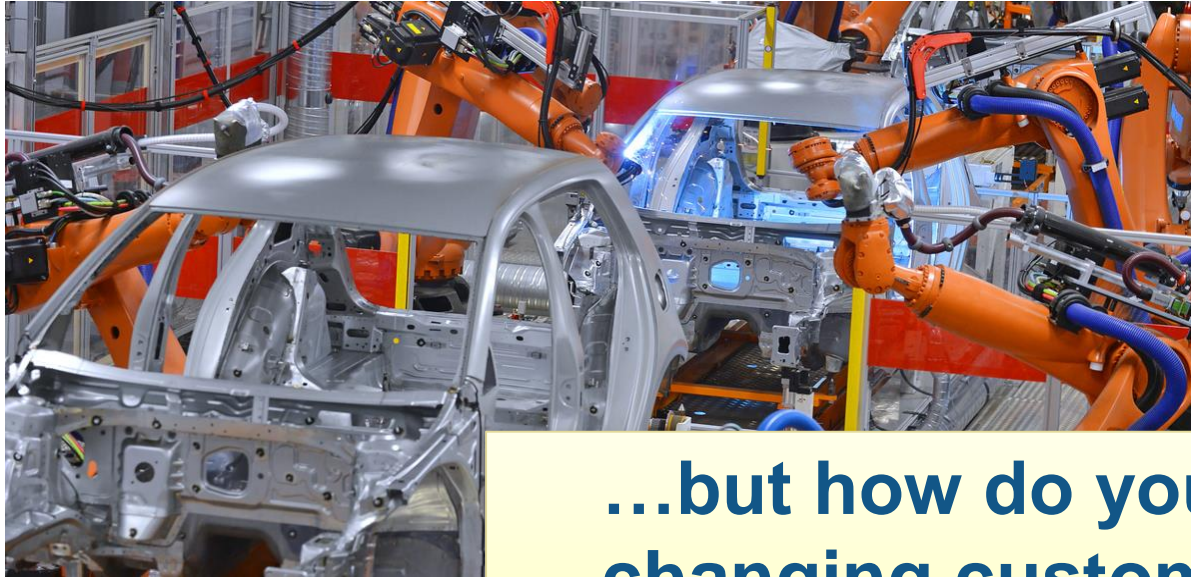
Paul Urban

Senior Product Manager
Simulink Verification and Validation

Building Algorithms in Everything...



Building Algorithms in Everything...

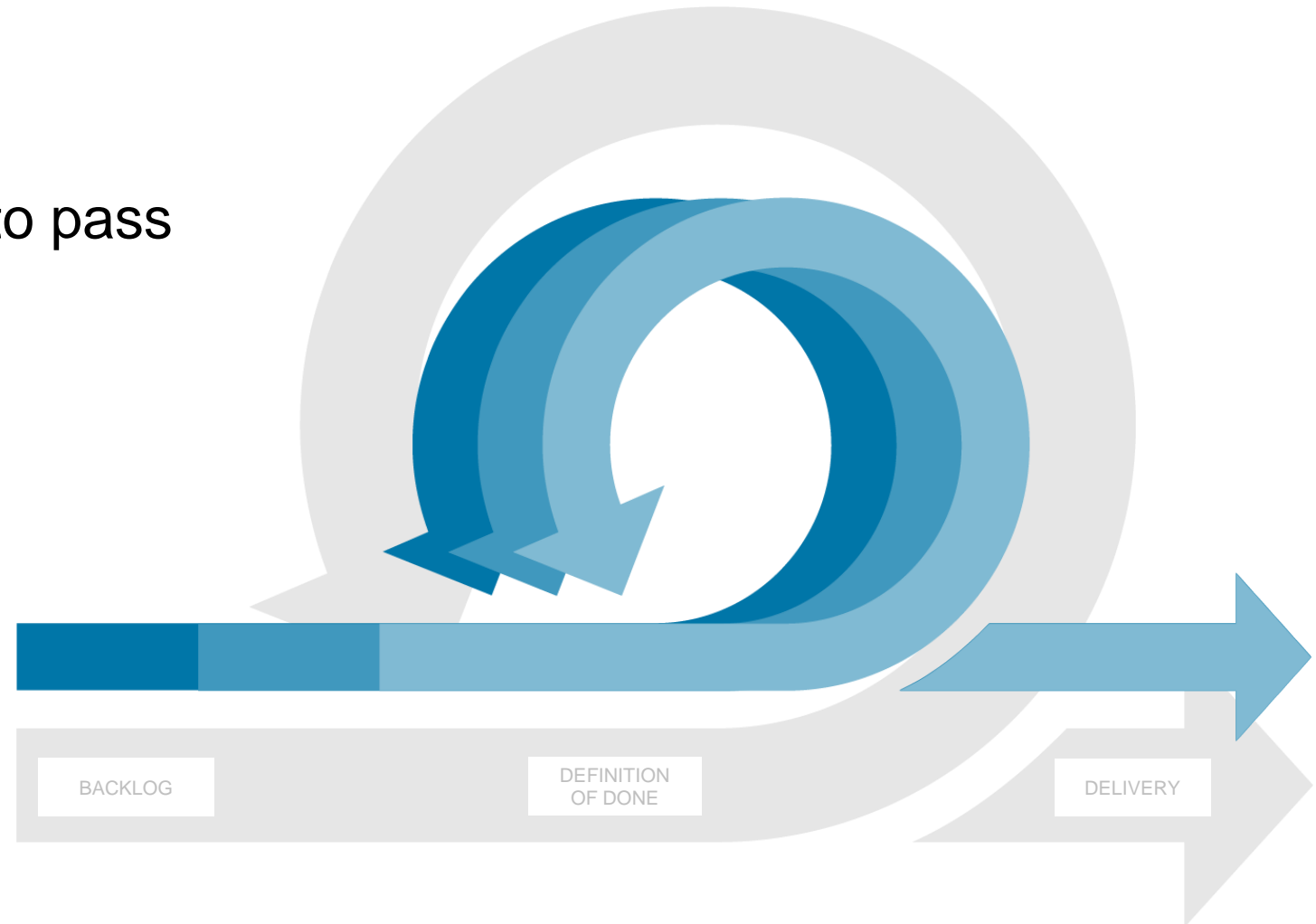


...but how do you deliver faster, meet changing customer requirements, and ensure quality?



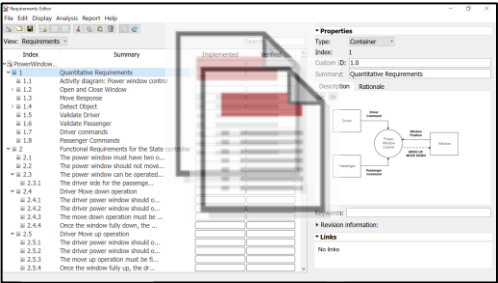
Test Driven Development Cycle

1. Create a test
2. Implement enough for test to pass
3. Refactor

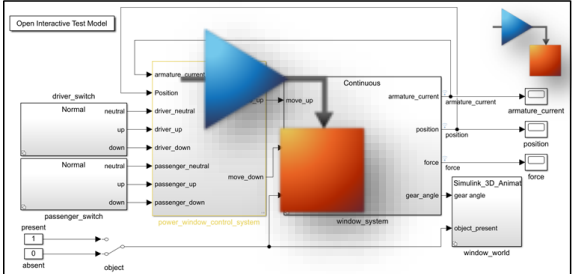


Simulink provides an integrated framework for TDD

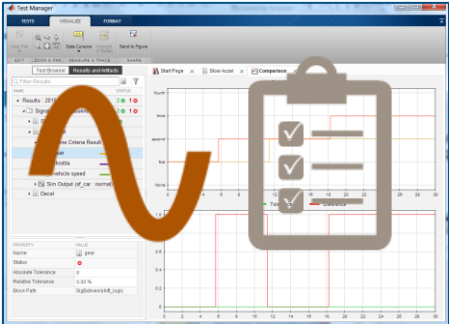
Requirements



Implementation



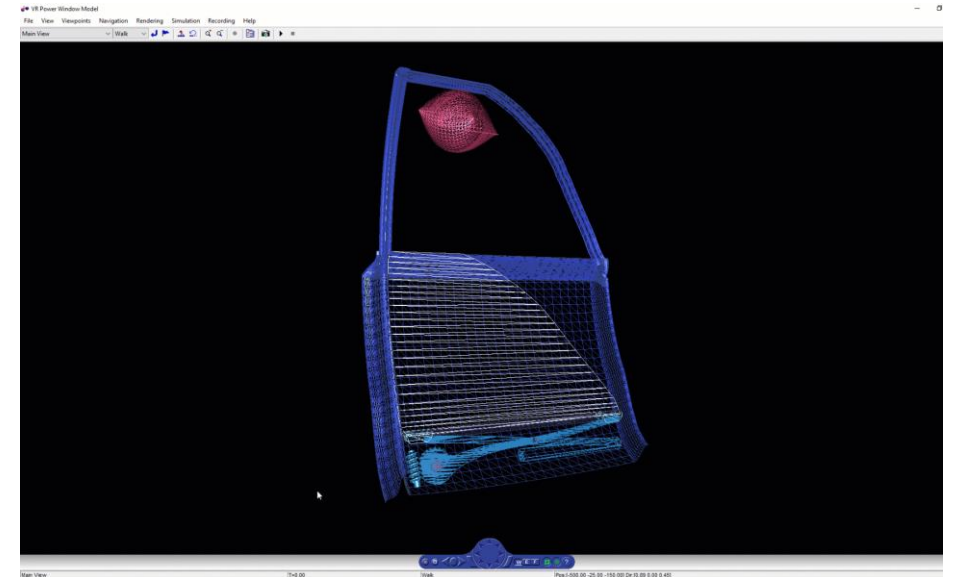
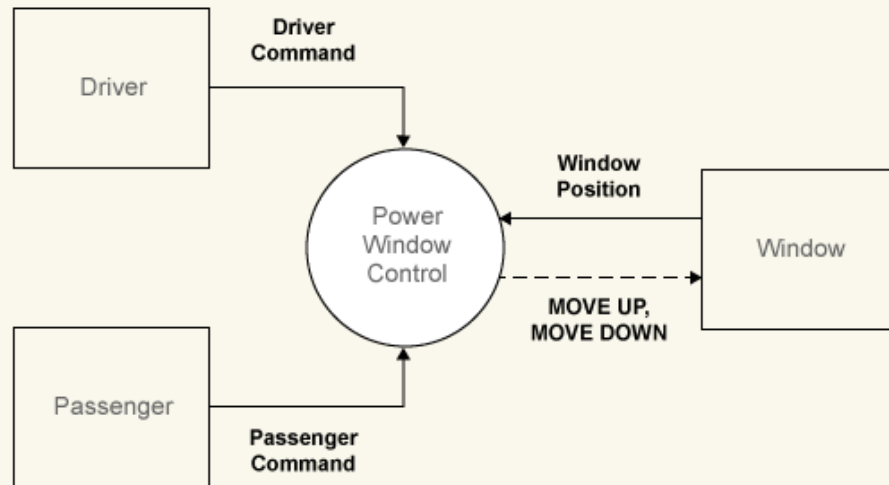
Test



Starting with high level customer requirements

User Requirements:

- Both driver and passenger can control the window
- Window stops closing if an object is detected
- Window should have option to fully open and close



Capturing requirements

The screenshot displays the Requirements Editor application. The main window is divided into a tree view on the left and a detailed view on the right. The tree view shows a hierarchy of requirements under 'PowerWindo...'. Requirement 2, 'Controller Functional Requirements', is selected. The detailed view on the right shows the properties of this requirement, including its type, index, custom ID, and summary. It also contains a description, a rationale, and a diagram of a power window mechanism. The diagram shows a window frame with a red obstacle at the top, and labels for 'top', 'obstacle position', and 'bottom'. Below the diagram, there are fields for keywords, revision information, and links. A link is provided for 'power_window_control_system'.

Index	Summary	Implemented	Verified
1	Quantitative Requirements	[Progress Bar]	[Progress Bar]
1.1	Activity diagram: Power window control	[Progress Bar]	[Progress Bar]
1.2	Open and Close Window	[Progress Bar]	[Progress Bar]
1.2.1	Fully Open	[Progress Bar]	[Progress Bar]
1.2.2	Fully Close	[Progress Bar]	[Progress Bar]
1.3	Move Response	[Progress Bar]	[Progress Bar]
1.4	Detect Object	[Progress Bar]	[Progress Bar]
1.5	Validate Driver	[Progress Bar]	[Progress Bar]
1.6	Validate Passenger	[Progress Bar]	[Progress Bar]
1.7	Driver commands	[Progress Bar]	[Progress Bar]
1.8	Passenger Commands	[Progress Bar]	[Progress Bar]
2	Controller Functional Requirements	[Progress Bar]	[Progress Bar]
2.1	The power window must have two o...	[Progress Bar]	[Progress Bar]
2.2	The power window should not move...	[Progress Bar]	[Progress Bar]
2.3	The power window can be operated...	[Progress Bar]	[Progress Bar]
2.3.1	The driver side for the passenge...	[Progress Bar]	[Progress Bar]
2.4	Driver Move down operation	[Progress Bar]	[Progress Bar]
2.4.1	Driver down button press	[Progress Bar]	[Progress Bar]
2.4.2	Move down to end stop	[Progress Bar]	[Progress Bar]
2.4.3	Move down automatically performance	[Progress Bar]	[Progress Bar]
2.4.4	Enter neutral when fully down	[Progress Bar]	[Progress Bar]
2.5	Driver Move up operation	[Progress Bar]	[Progress Bar]
2.5.1	The driver power window should o...	[Progress Bar]	[Progress Bar]
2.5.2	The driver power window should o...	[Progress Bar]	[Progress Bar]
2.5.3	The move up operation must be fi...	[Progress Bar]	[Progress Bar]
2.5.4	Once the window fully up, the dr...	[Progress Bar]	[Progress Bar]
2.6	Passenger Move down operation	[Progress Bar]	[Progress Bar]
2.6.1	The Passenger power window shoul...	[Progress Bar]	[Progress Bar]
2.6.2	The Passenger power window shoul...	[Progress Bar]	[Progress Bar]
2.6.3	The move down operation must be ...	[Progress Bar]	[Progress Bar]
2.6.4	Once the window fully down, the ...	[Progress Bar]	[Progress Bar]

Properties
Type: Container
Index: 2
Custom ID: REQ 2
Summary: Controller Functional Requirements

Description
Both the driver and passenger can send commands to the window to move it up and down. The controller infers the correct command to send to the window actuator (e.g., the driver command has priority over the passenger command). In addition, diagram monitors the state of the window system to establish when the window is fully opened and closed and to detect if there is an object between the window and frame.

Diagram Labels:
- top
- obstacle position
- bottom

Keywords:
Revision information:
Links
Implemented by: [power_window_control_system](#)

Viewing details

The screenshot shows the Requirements Editor window with a table of requirements. The table has four columns: Index, Summary, Implemented, and Verified. The 'Implemented' column shows blue progress bars, and the 'Verified' column shows green progress bars. A blue arrow points from the text 'Requirement Details' to the 'Index' column of the table.

Index	Summary	Implemented	Verified
PowerWindo...		[Progress Bar]	[Progress Bar]
1	Quantitative Requirements	[Progress Bar]	[Progress Bar]
1.1	Activity diagram: Power window control	[Progress Bar]	[Progress Bar]
1.2	Open and Close Window	[Progress Bar]	[Progress Bar]
1.2.1	Fully Open	[Progress Bar]	[Progress Bar]
1.2.2	Fully Close	[Progress Bar]	[Progress Bar]
1.3	Move Response	[Progress Bar]	[Progress Bar]
1.4	Detect Object	[Progress Bar]	[Progress Bar]
1.5	Validate Driver	[Progress Bar]	[Progress Bar]
1.6	Validate Passenger	[Progress Bar]	[Progress Bar]
1.7	Driver commands	[Progress Bar]	[Progress Bar]
1.8	Passenger Commands	[Progress Bar]	[Progress Bar]
2	Controller Functional Requirements	[Progress Bar]	[Progress Bar]
2.1	The power window must have two o...	[Progress Bar]	[Progress Bar]
2.2	The power window should not move...	[Progress Bar]	[Progress Bar]
2.3	The power window can be operated...	[Progress Bar]	[Progress Bar]
2.3.1	The driver side for the passenge...	[Progress Bar]	[Progress Bar]
2.4	Driver Move down operation	[Progress Bar]	[Progress Bar]
2.4.1	Driver down button press	[Progress Bar]	[Progress Bar]
2.4.2	Move down to end stop	[Progress Bar]	[Progress Bar]
2.4.3	Move down automatically performance	[Progress Bar]	[Progress Bar]
2.4.4	Enter neutral when fully down	[Progress Bar]	[Progress Bar]
2.5	Driver Move up operation	[Progress Bar]	[Progress Bar]
2.5.1	The driver power window should o...	[Progress Bar]	[Progress Bar]
2.5.2	The driver power window should o...	[Progress Bar]	[Progress Bar]
2.5.3	The move up operation must be fi...	[Progress Bar]	[Progress Bar]
2.5.4	Once the window fully up, the dr...	[Progress Bar]	[Progress Bar]
2.6	Passenger Move down operation	[Progress Bar]	[Progress Bar]
2.6.1	The Passenger power window shoul...	[Progress Bar]	[Progress Bar]
2.6.2	The Passenger power window shoul...	[Progress Bar]	[Progress Bar]
2.6.3	The move down operation must be ...	[Progress Bar]	[Progress Bar]
2.6.4	Once the window fully down, the ...	[Progress Bar]	[Progress Bar]

Requirement
Details



Organizing and creating requirement hierarchies

The screenshot shows the Requirements Editor interface. On the left, a tree view displays a hierarchy of requirements under 'PowerWindo...'. The tree is expanded to show sub-requirements like '1.1 Quantitative Requirements', '1.2 Open and Close Window', and '2 Controller Functional Requirements'. On the right, a table provides a summary for each requirement, including columns for 'Implemented' and 'Verified' status, each with a progress bar. Requirement 2 is highlighted in blue.

Index	Summary	Implemented	Verified
PowerWindo...		[Progress Bar]	[Progress Bar]
1	Quantitative Requirements	[Progress Bar]	[Progress Bar]
1.1	Activity diagram: Power window control	[Progress Bar]	[Progress Bar]
1.2	Open and Close Window	[Progress Bar]	[Progress Bar]
1.2.1	Fully Open	[Progress Bar]	[Progress Bar]
1.2.2	Fully Close	[Progress Bar]	[Progress Bar]
1.3	Move Response	[Progress Bar]	[Progress Bar]
1.4	Detect Object	[Progress Bar]	[Progress Bar]
1.5	Validate Driver	[Progress Bar]	[Progress Bar]
1.6	Validate Passenger	[Progress Bar]	[Progress Bar]
1.7	Driver commands	[Progress Bar]	[Progress Bar]
1.8	Passenger Commands	[Progress Bar]	[Progress Bar]
2	Controller Functional Requirements	[Progress Bar]	[Progress Bar]
2.1	The power window must have two o...	[Progress Bar]	[Progress Bar]
2.2	The power window should not move...	[Progress Bar]	[Progress Bar]
2.3	The power window can be operated...	[Progress Bar]	[Progress Bar]
2.3.1	The driver side for the passenge...	[Progress Bar]	[Progress Bar]
2.4	Driver Move down operation	[Progress Bar]	[Progress Bar]
2.4.1	Driver down button press	[Progress Bar]	[Progress Bar]
2.4.2	Move down to end stop	[Progress Bar]	[Progress Bar]
2.4.3	Move down automatically performance	[Progress Bar]	[Progress Bar]
2.4.4	Enter neutral when fully down	[Progress Bar]	[Progress Bar]
2.5	Driver Move up operation	[Progress Bar]	[Progress Bar]
2.5.1	The driver power window should o...	[Progress Bar]	[Progress Bar]
2.5.2	The driver power window should o...	[Progress Bar]	[Progress Bar]
2.5.3	The move up operation must be fi...	[Progress Bar]	[Progress Bar]
2.5.4	Once the window fully up, the dr...	[Progress Bar]	[Progress Bar]
2.6	Passenger Move down operation	[Progress Bar]	[Progress Bar]
2.6.1	The Passenger power window shoul...	[Progress Bar]	[Progress Bar]
2.6.2	The Passenger power window shoul...	[Progress Bar]	[Progress Bar]
2.6.3	The move down operation must be ...	[Progress Bar]	[Progress Bar]
2.6.4	Once the window fully down, the ...	[Progress Bar]	[Progress Bar]

Requirement Hierarchies



Specifying details

The screenshot shows a 'Properties' window for a requirement. It includes fields for Type (Container), Index (2), Custom ID (REQ 2), and Summary (Controller Functional Requirements). There are tabs for Description and Rationale. The Description tab is active, showing a text description and a diagram of a window with labels for top, obstacle position, and bottom. Below the description are fields for Keywords, Revision information, and a Links section with an 'Implemented by' field containing a link to 'power_window_control_system'.

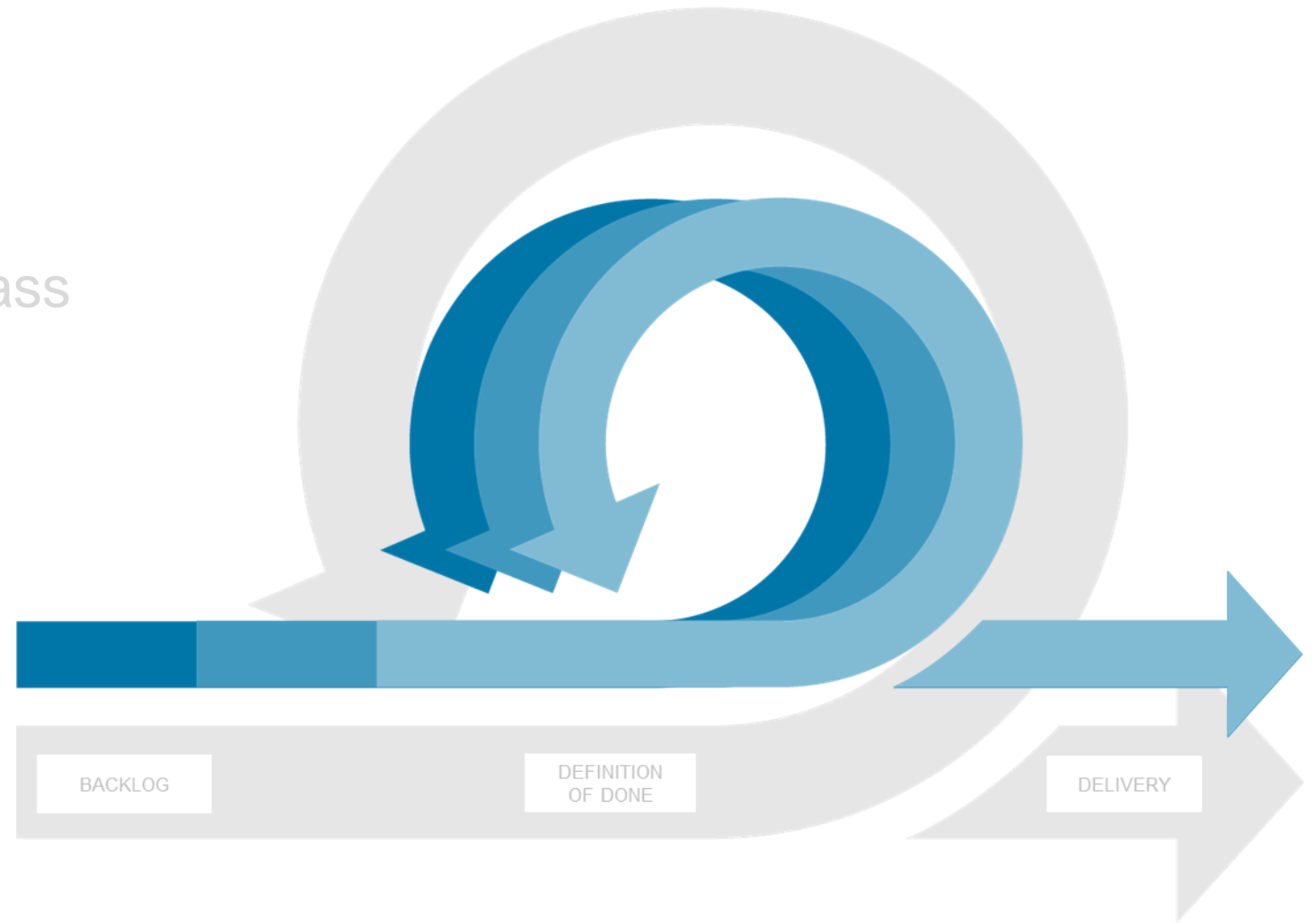
Description / Rationale Fields

Links Pane

1. Create a test

2. Implement enough for test to pass

3. Refactor



Develop, manage, and execute simulation-based tests

Simulink Test

Test Manager

- Author, manage, organize tests

The Test Manager interface includes a Test Browser on the left, a central Test Results window with a graph showing tolerance and difference over time, and a Reports window at the bottom. The report is titled "Report Generated by Test Manager" and includes fields for Title, Author, Date, and Test Environment.

Test Harnesses

- Isolate Component Under Test

The Main Model shows a vehicle simulation with components like Engine, Transmission, and Vehicle. A red dashed box highlights the "shift_logic" component. The Test Harness window shows the "shift_logic" component isolated, with inputs for speed, throttle, and gear, and an output for gear.

Test Authoring

- Specify test inputs, expected outputs, and tolerances

The Test Sequence Editor shows a table of test steps and transitions. The Signal Editor shows a scenario with a signal input. The Temporal Assessments window shows a timing diagram with a trigger and response signal.

Step	Transition	Next Step
init_step input speed = ramp (t); throttle = ramp (t);	1. after (2. sec)	step_2
step_2 speed = 2' ramp (t); throttle = 2' ramp (t);	1. gear == 3	step_3
step_3 peak_speed = speed; peak_throttle = throttle;		

Creating a Test Harness to isolate Component Under Test

Copyright 2013-2016 The MathWorks, Inc.

Specify properties of the Test Harness

Create Test Harness

Specify the properties of the test harness. The component under test is the system for which the harness is being created. After creation, use the block badge to find and open harnesses.

Component under Test: [slexPowerWindowExample/power_window_control_system](#)

Basic Properties | Advanced Properties | Description

Name:

Harnesses saved internally. [More information](#)

Sources and Sinks

→ →

Create scalar inputs

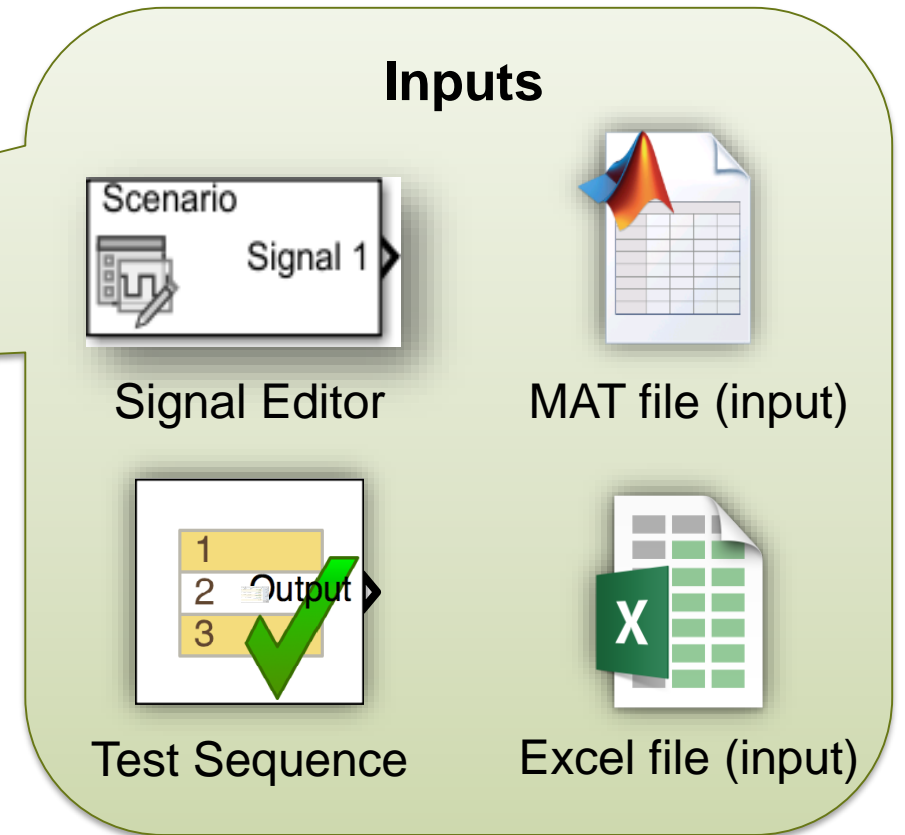
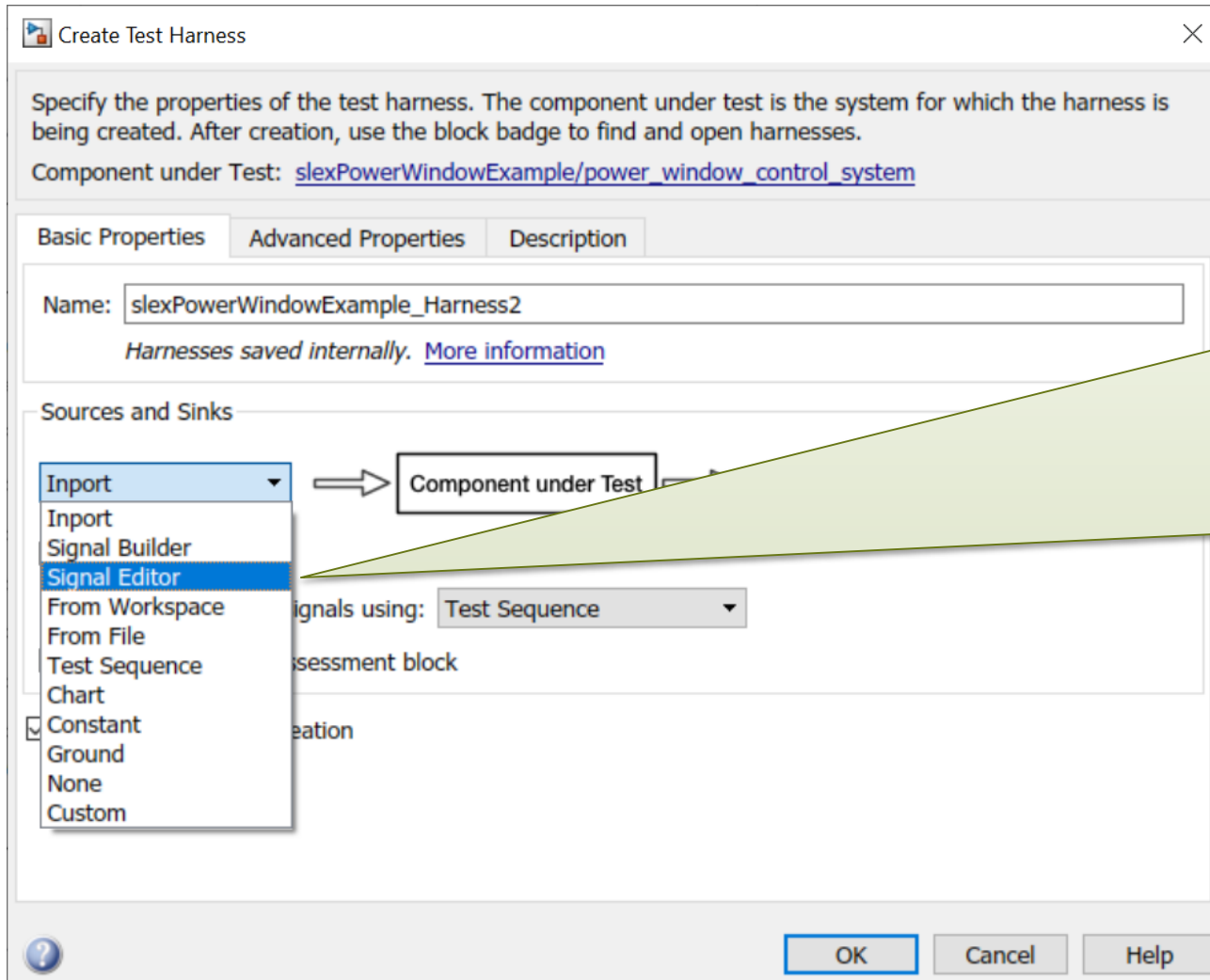
Generate function-call signals using:

Add separate Test Assessment block

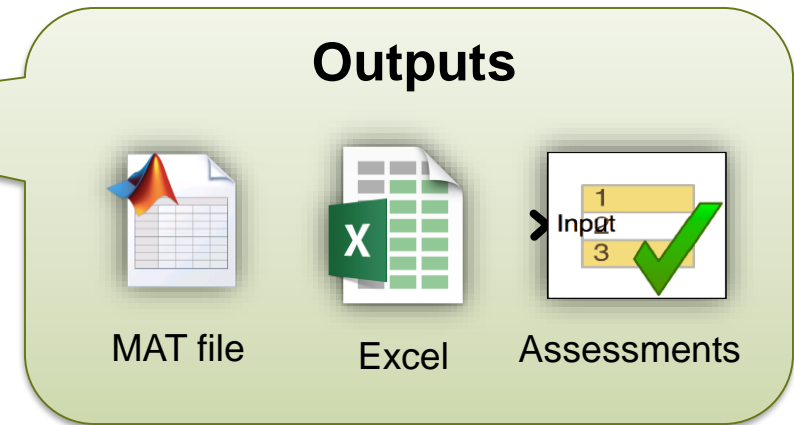
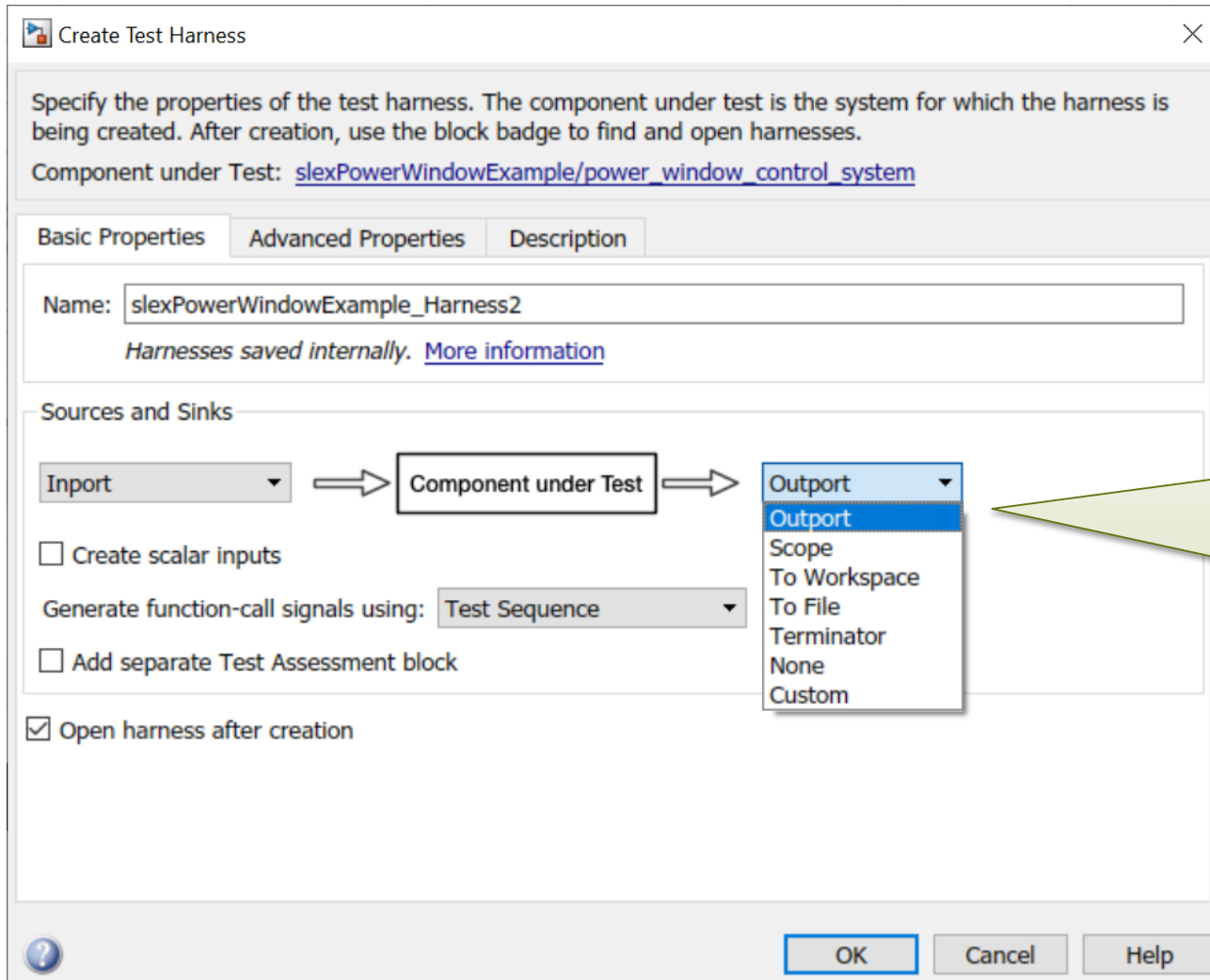
Open harness after creation

? OK Cancel Help

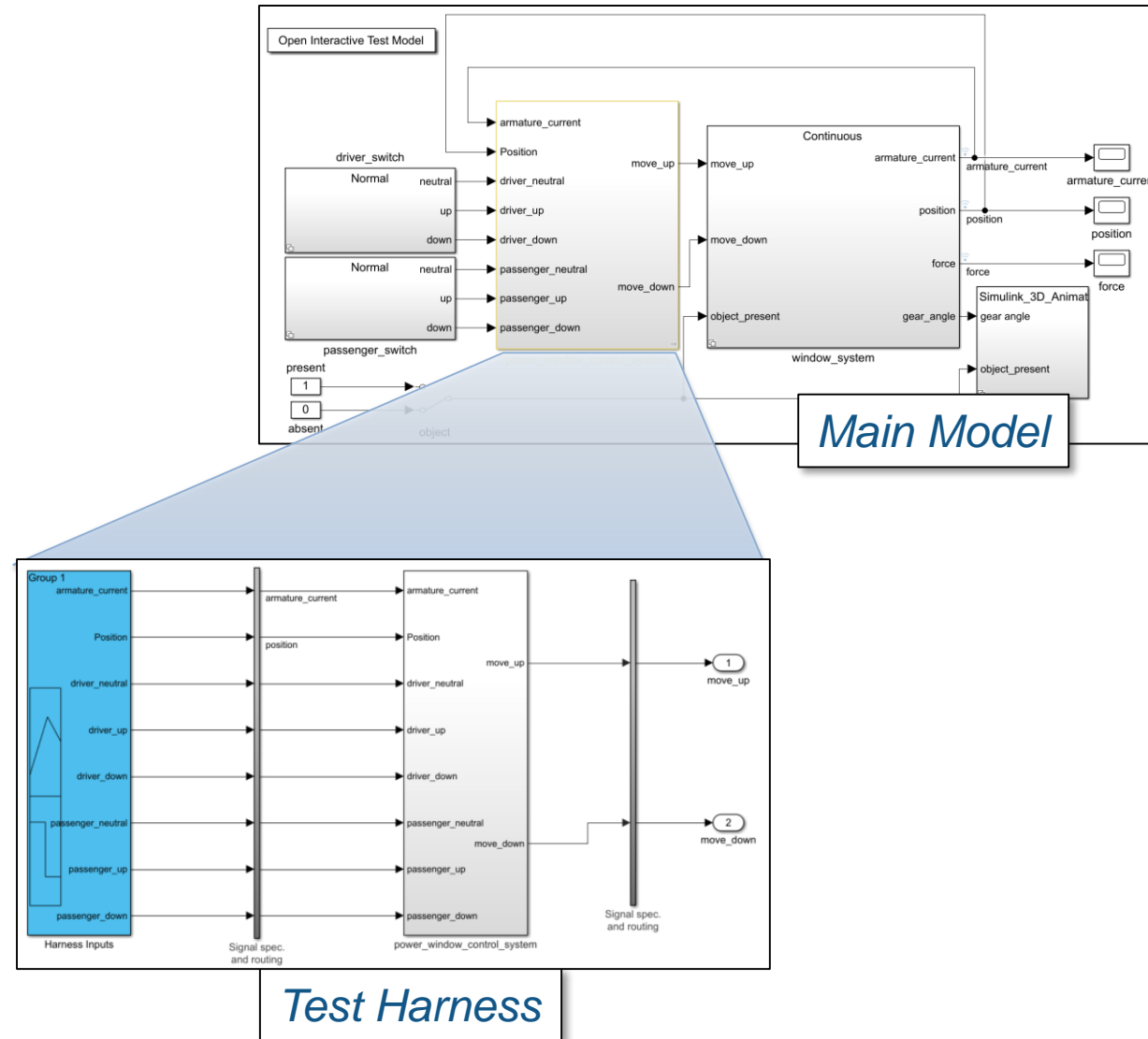
Specify inputs



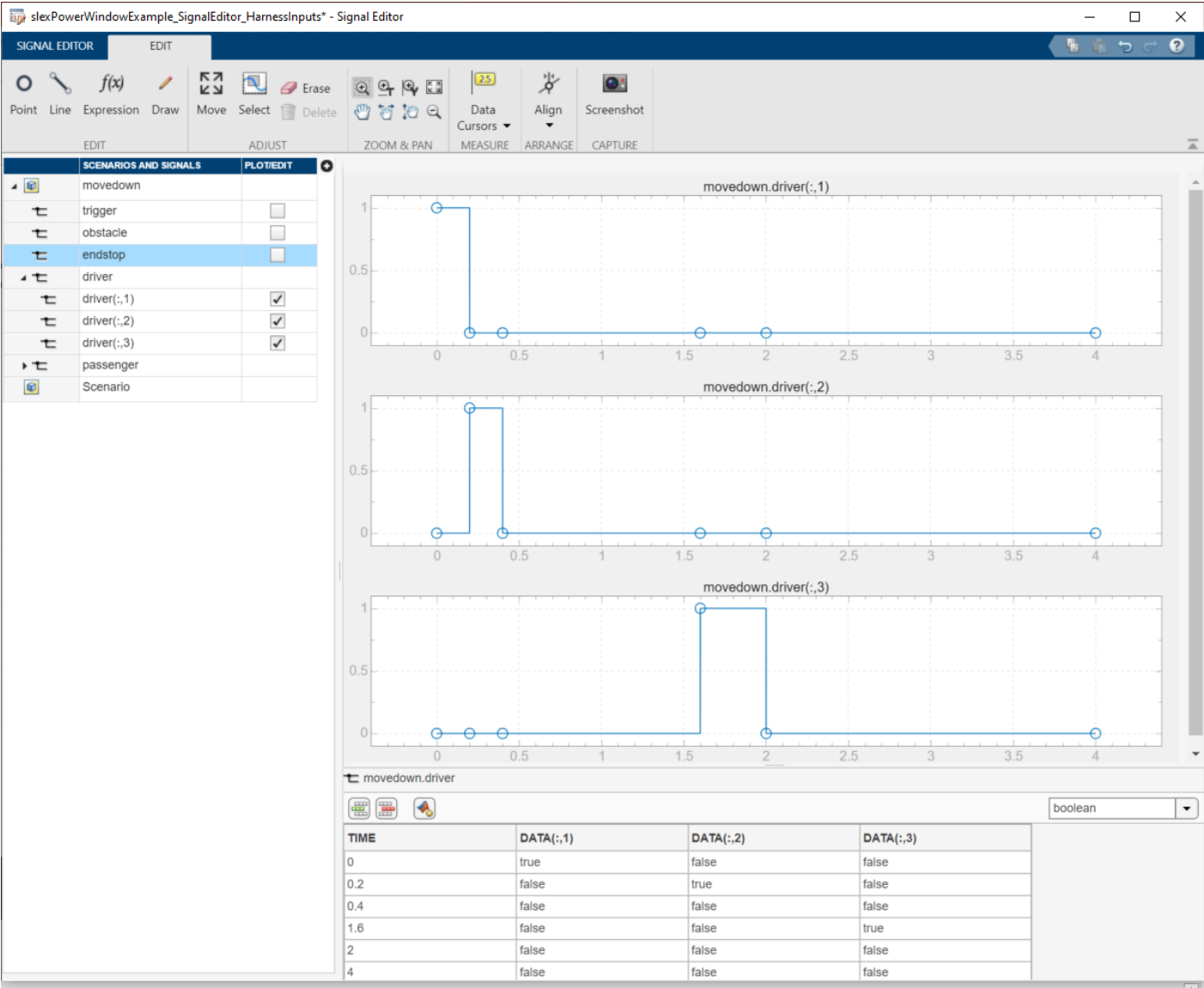
Specify outputs



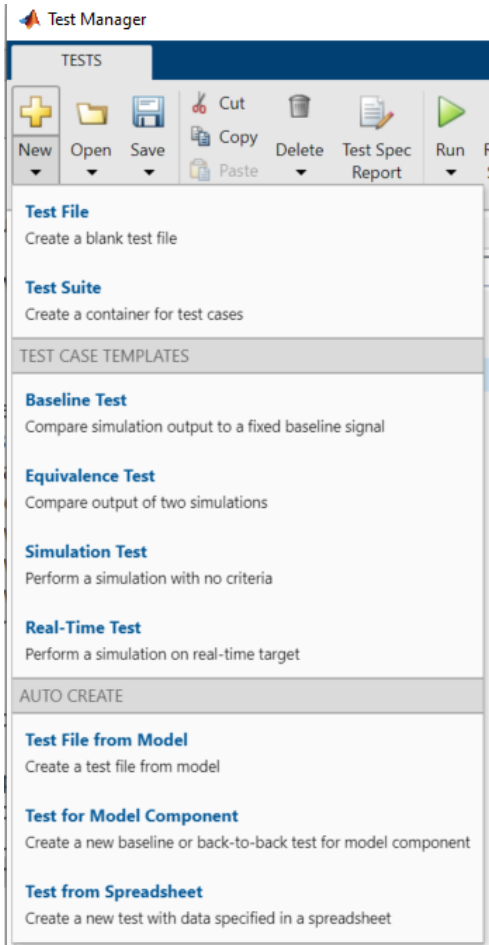
Created Test Harness to isolate Component Under Test



Authoring tests using Signal Editor

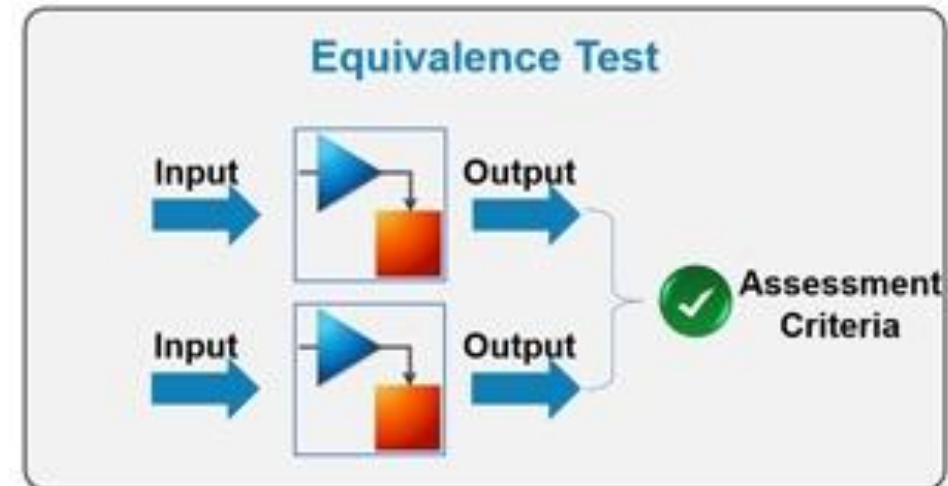
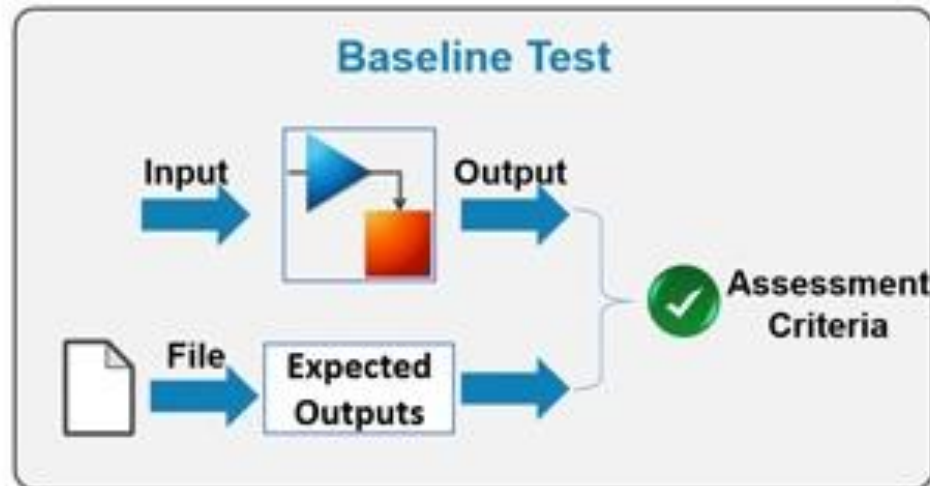


Use templates and wizards to automate test case creation



Use templates and wizards to automate test case creation

Test Case Templates



Create Simulation Test and link to requirement

The screenshot shows the MATLAB Test Manager interface. The main window displays the configuration for a simulation test named 'DriverMovedown'. The interface includes a toolbar with various actions like 'New', 'Open', 'Save', 'Run', and 'Report'. The left pane shows a tree view of test suites, with 'DriverMovedown' selected. The right pane shows the test configuration details, including a list of requirements and system under test settings.

DriverMovedown Enabled

Main_test_SignalBuilder » New_Test_Suite_1 » DriverMovedown

Simulation Test

Select releases for simulation:

Create Test Case from External File

DESCRIPTION

REQUIREMENTS*

[Driver Move down operation: Driver Move down operation \(PowerWindowFunctionalRequirements...](#)

+ Add - Delete

SYSTEM UNDER TEST*

Model:

TEST HARNESS*

Harness:

SIMULATION SETTINGS OVERRIDES

PARAMETER OVERRIDES ?

INPUTS* ?

SIMULATION OUTPUTS ?

CONFIGURATION SETTINGS OVERRIDES ?

ITERATIONS ?

LOGICAL AND TEMPORAL ASSESSMENTS* ?

CUSTOM CRITERIA ?

COVERAGE SETTINGS ?

PROPERTY	VALUE
Name	DriverMovedown
Type	Simulation Test
Model	slexPowerWindow...
Harness Name	slexPowerWindow...
Simulation Mode	[Model Settings]
Location	C:\Demos\TDDPow...
Enabled	<input checked="" type="checkbox"/>
Hierarchy	Main_test_SignalB...
Tags	Type comma or space

Link to requirements

Specify model to test

Test fails due to compilation error

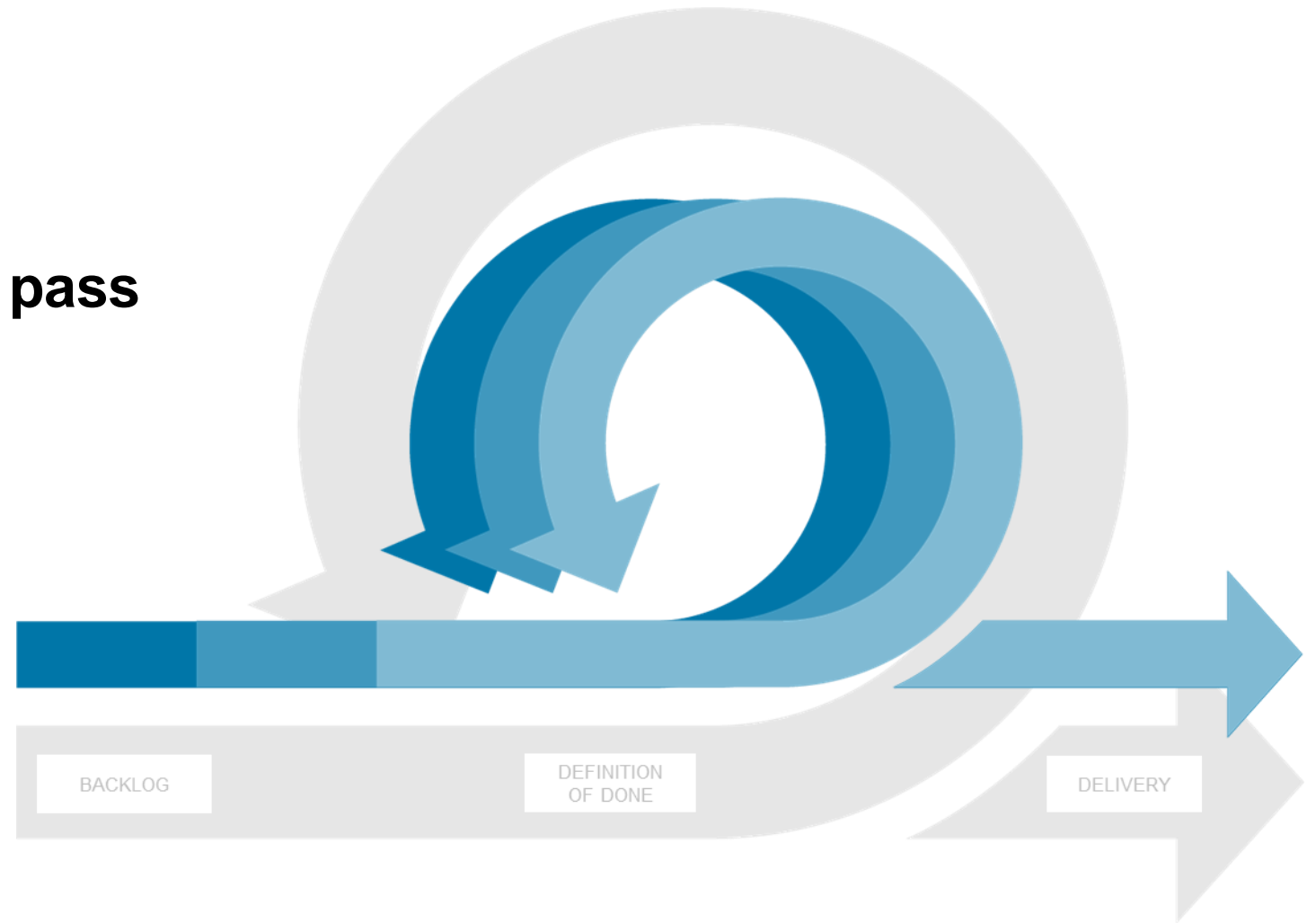
The screenshot shows the MATLAB Test Manager interface. The top menu bar includes options like New, Open, Save, Cut, Copy, Paste, Delete, Test Spec Report, Run, Run with Stepper, Stop, Parallel, Report, Visualize, Highlight in Model, Import, Export, Preferences, and Help. The main area is divided into a Test Browser on the left and a Results and Artifacts pane on the right. The Test Browser shows a tree view of test results for 'Results: 2020-Mar-22 21:24:44', with 'DriverMoveDown' selected. The Results and Artifacts pane shows the details for the 'DriverMoveDown' test case, including a summary table, test requirements, errors, and a description.

PROPERTY	VALUE
Name	DriverMoveDown
Status	1 ✖
Start Time	03/22/2020 21:24:48
End Time	03/22/2020 21:24:48
Type	Simulation Test
Test File Location	C:\Demos\TDDPowerWind...
Test Case Definition	

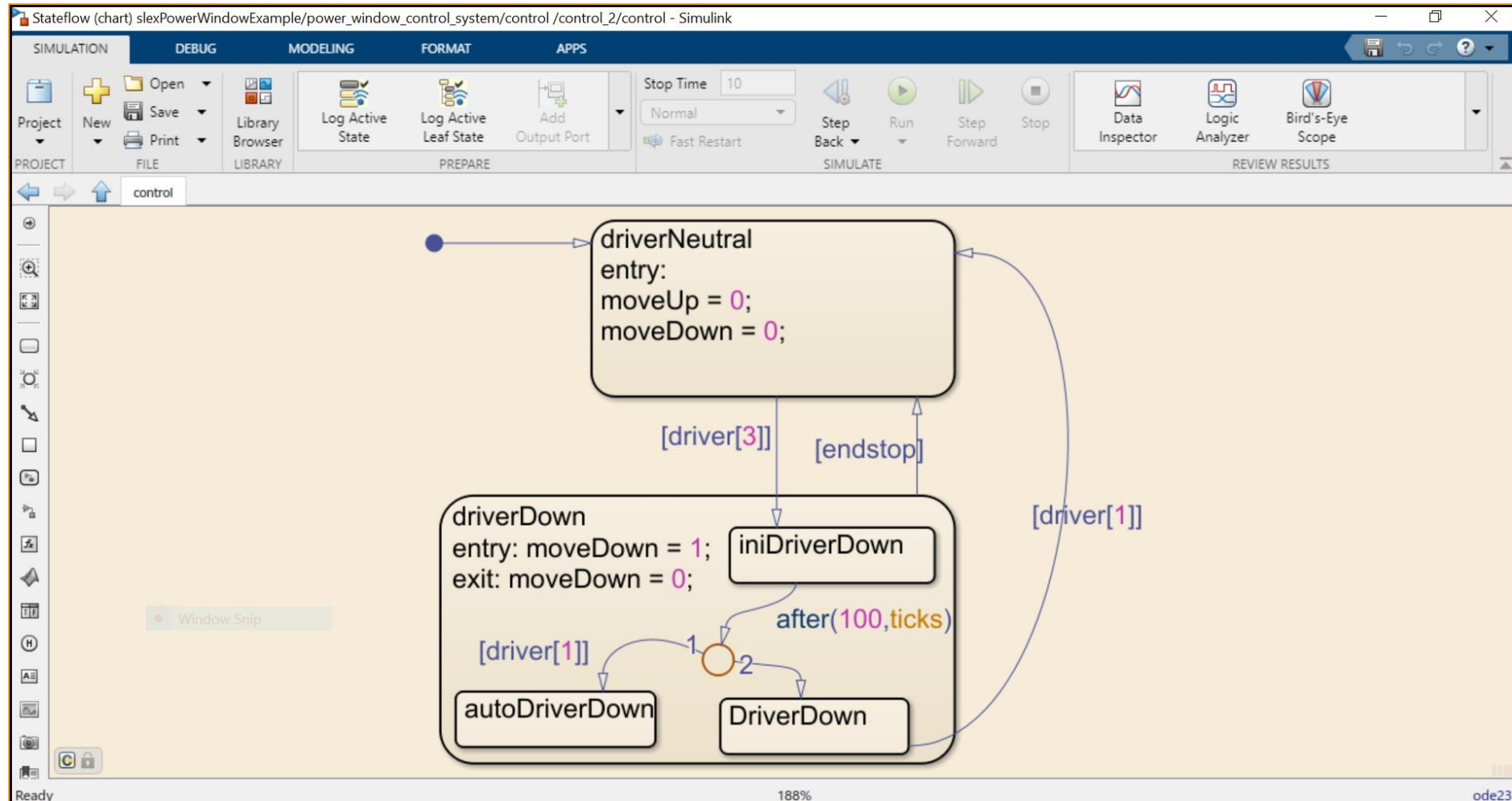
ERRORS

- Signal Editor scenario 'movedown' not found in model slxPowerWindowExample_SignalEditor.

1. Create a test
2. **Implement enough for test to pass**
3. Refactor



Implement enough to get test to pass



Linking implementation to requirements

The screenshot displays the Simulink Requirements Editor interface. The main workspace shows a stateflow chart for a control system. A callout box labeled "1.1.1: Fully Open" with the text "If the up command is issued for between 200 ms and 1 s, the window must fully open." is linked via an "IMPLEMENTS" arrow to the "autoDriverUp" state in the chart. The Property Inspector on the right shows the details for requirement 1.1.1, including its index (1.2.1), custom ID (1.1.1), and summary (Fully Open). The Requirements table at the bottom lists the requirements and their implementation status.

Index	Summary	Implemented	Verified
1.2	Open and Close Window	Yes	
1.2.1	Fully Open	Yes	
1.2.2	Fully Close	Yes	

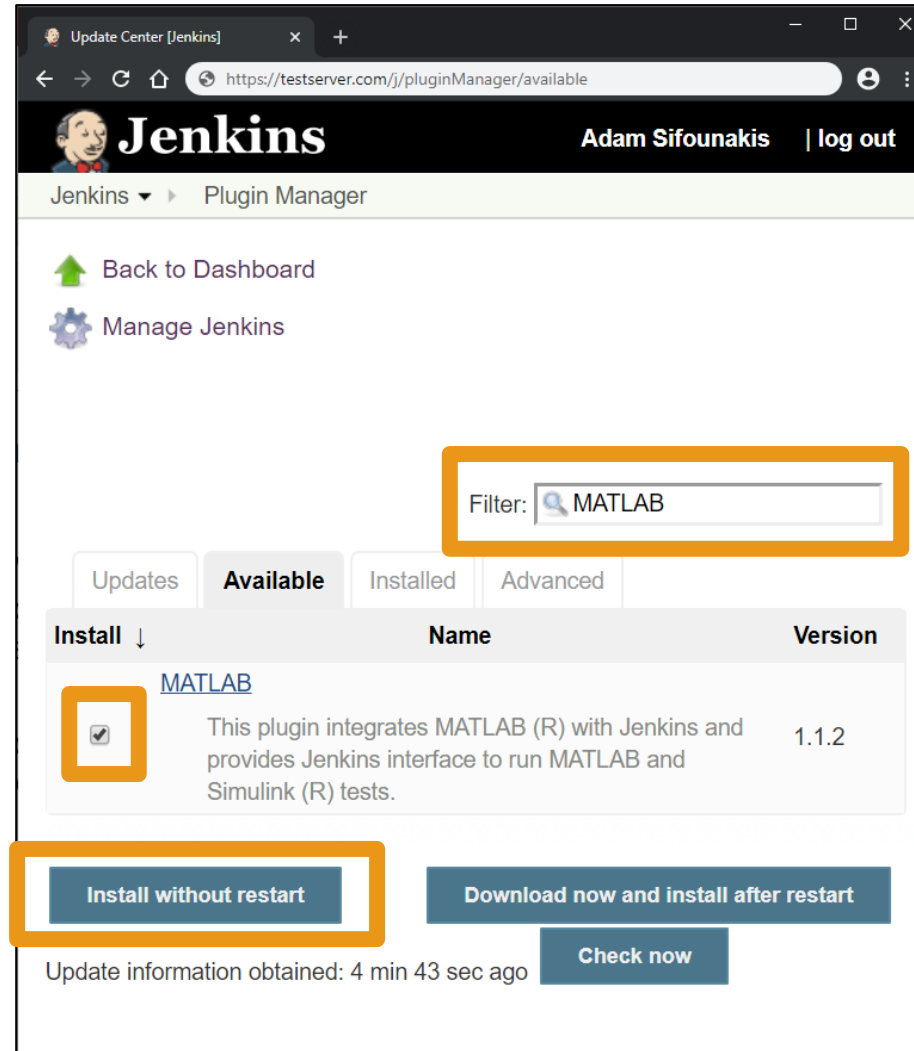
Managing artifacts with source control directly from Projects

The screenshot displays the MATLAB R2020a interface with the 'PROJECT SHORTCUTS' tab selected. A callout box highlights the 'SOURCE CONTROL' section, which includes icons for Git Details, Refresh, Commit, Fetch, Push, Pull, Remote, Branches, Submodules, and Stashes. Below this, a detailed view of the 'Project - Power Window Controller Project' is shown, listing files and folders with their status, date modified, classification, and Git status.

Name	Status	Date Modified	Classification	Git
configureModel	✓	3/31/2020 8:08 AM	Configuration	·
data	✓	3/31/2020 8:08 AM	DesignSupport	·
hmi	✓	3/31/2020 8:08 AM	Visualization	·
images	✓	3/31/2020 8:08 AM	·	·
model	✓	4/1/2020 3:46 PM	Design	·
slexPowerWindowCntlCoverage.slx	✓	3/31/2020 8:08 AM	Test	·
slexPowerWindowCntlCoverageIncrease...	✓	3/31/2020 8:08 AM	Test	·
slexPowerWindowCntlInteract.slx	✓	3/31/2020 8:08 AM	Test	·
slexPowerWindowControl.slx	✓	3/31/2020 8:08 AM	Design	·
slexPowerWindowDSPLib.slx	✓	3/31/2020 8:08 AM	DesignSupport	·
slexPowerWindowExample.slmx	✓	3/31/2020 8:08 AM	·	·
slexPowerWindowExample.slx	✓	3/31/2020 8:08 AM	Design	·

Scale and automate testing with Continuous Integration

- Schedule automatic code and model testing
- Access MATLAB Plugin for Jenkins



The screenshot shows the Jenkins Update Center interface. The browser address bar indicates the URL is `https://testserver.com/j/pluginManager/available`. The Jenkins logo and user name "Adam Sifounakis" are visible at the top. The page title is "Jenkins > Plugin Manager". There are navigation links for "Back to Dashboard" and "Manage Jenkins". A search filter is set to "MATLAB". Below the filter, there are tabs for "Updates", "Available", "Installed", and "Advanced". The "Available" tab is selected, showing a table with the following content:

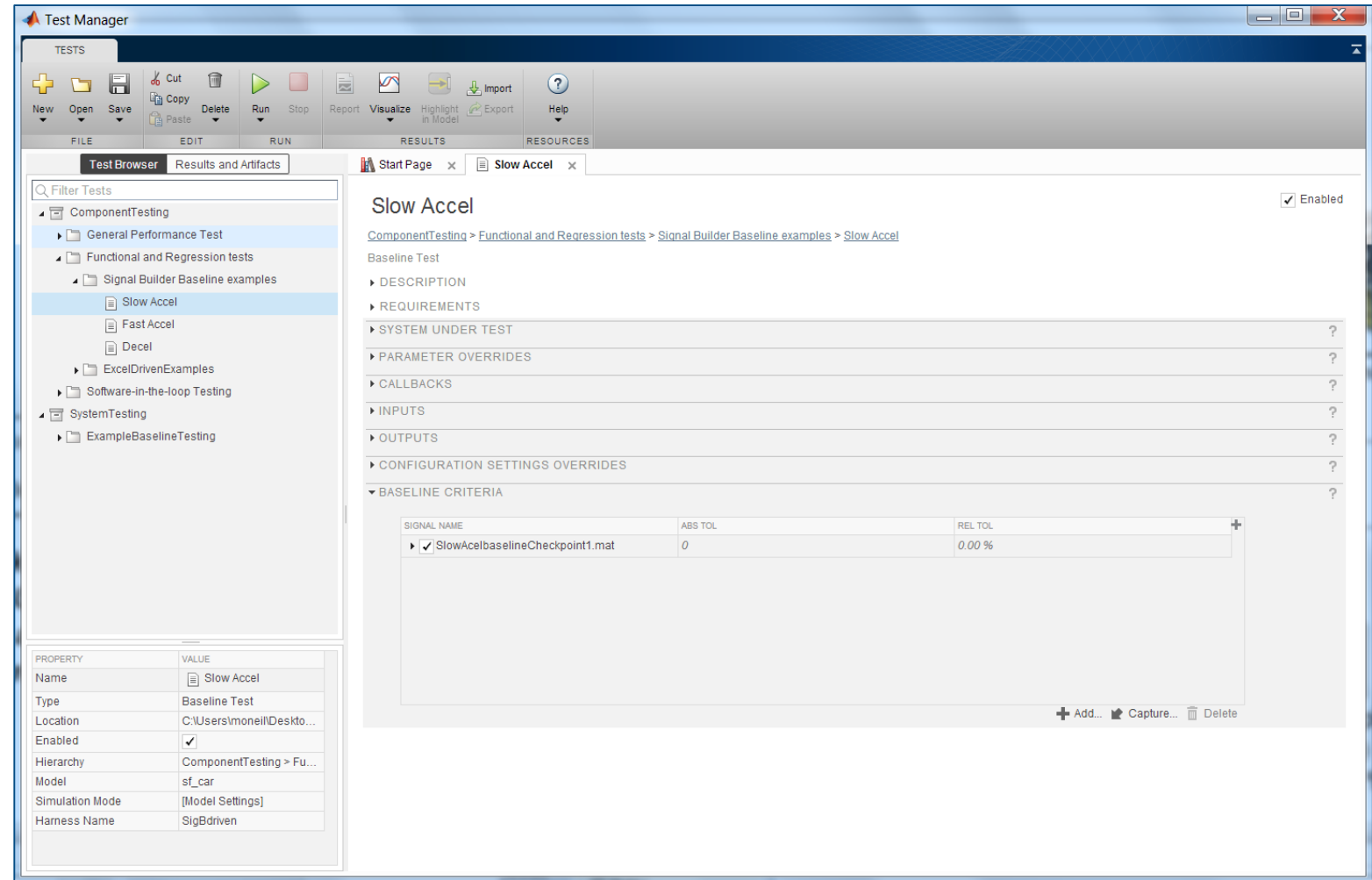
Install ↓	Name	Version
<input checked="" type="checkbox"/>	MATLAB This plugin integrates MATLAB (R) with Jenkins and provides Jenkins interface to run MATLAB and Simulink (R) tests.	1.1.2

Below the table, there are two buttons: "Install without restart" (highlighted with an orange box) and "Download now and install after restart". A "Check now" button is also present. At the bottom, it says "Update information obtained: 4 min 43 sec ago".



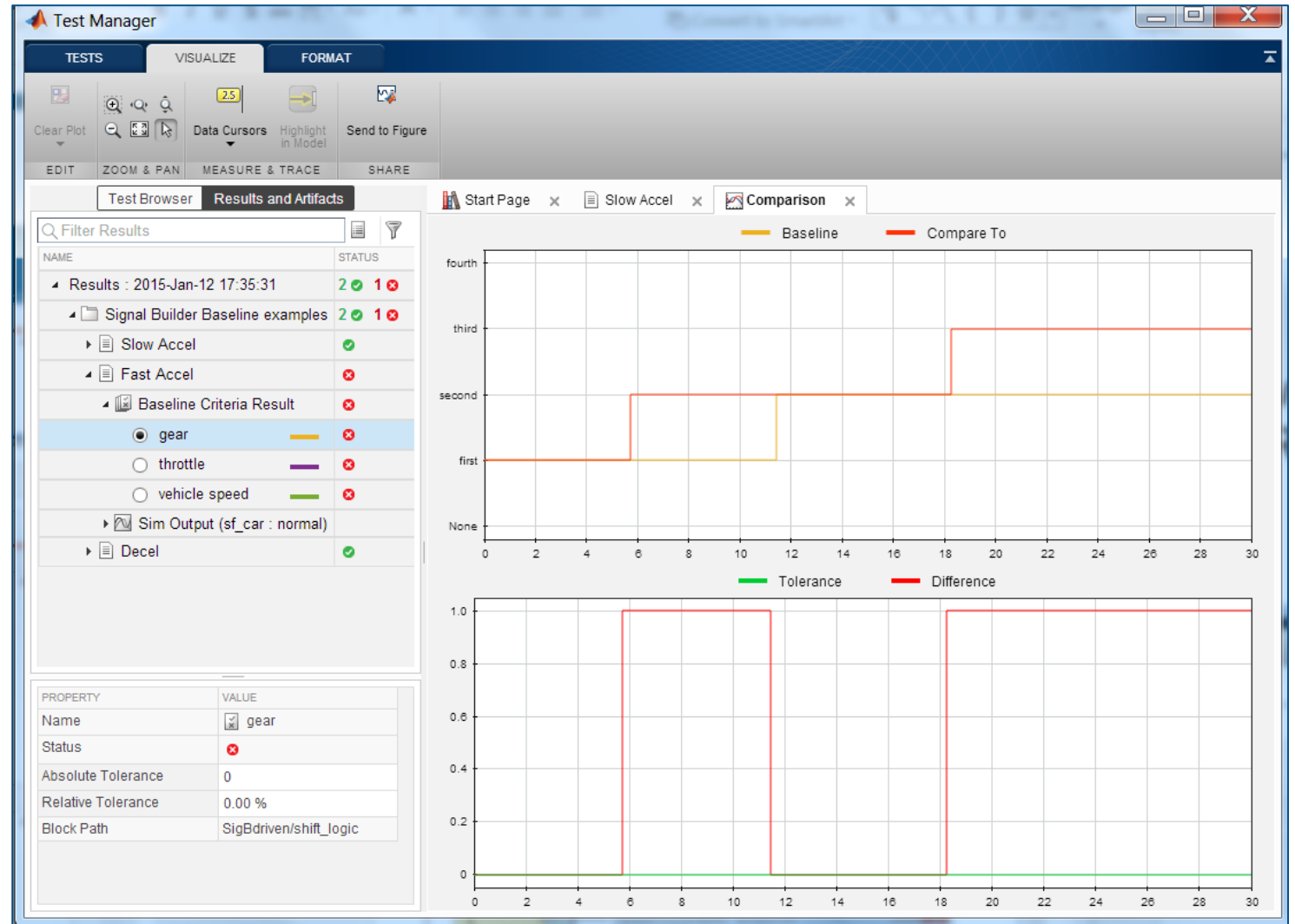
Executing test with Test Manager

- Group into suites and test files
- Execute individual or batch

















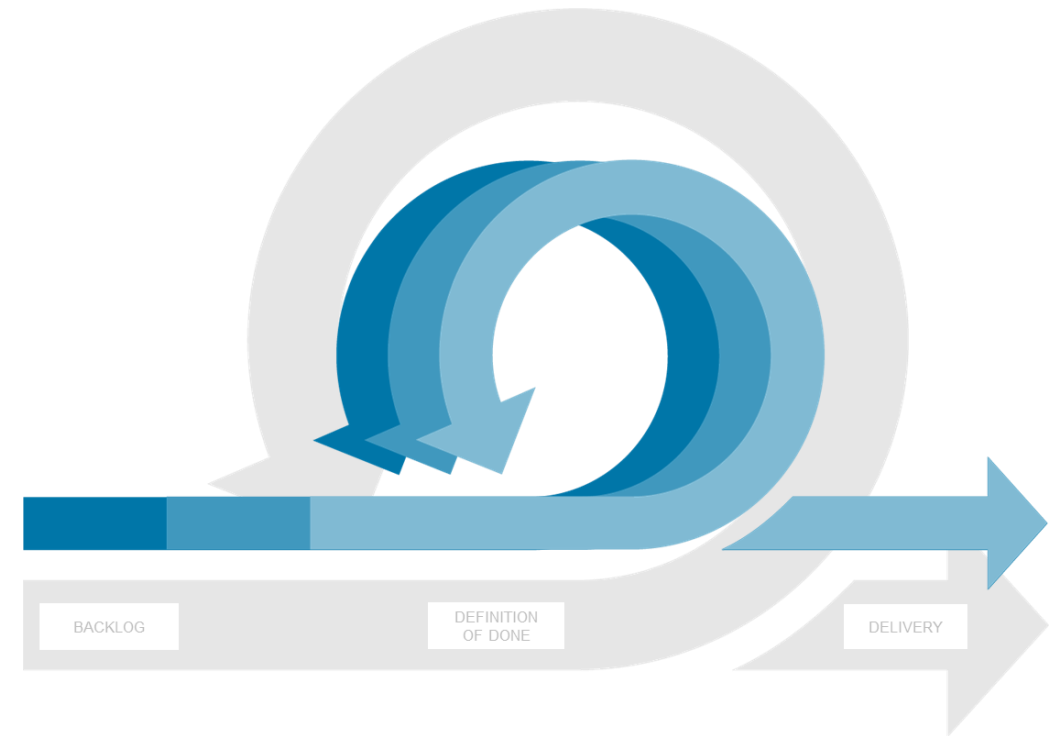
Analyzing and debugging results with Test Manager

- View result summary
- Debug using Simulation Data Inspector
- Archive, export, and report results



Executing all tests until they pass

▼  PowerWindowControlUnitTest_sig	6 
▶  DriverMoveDown	
▶  DriverMoveUp	
▶  EmergencyObstacle	
▶  Detect Object	
▶  Fully Close Window	
▶  PassengerMovedown	



Measuring testing completeness with coverage

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

The image illustrates three methods for measuring testing completeness with coverage:

- Simulink:** A block diagram showing a control loop. It includes a summing junction with a greater-than-or-equal-to (\geq) block, an AND gate, a transfer function block $\frac{1}{z}$, and a switch block with states T and F. Inputs are labeled [upper] and [lower].
- Stateflow:** A state transition diagram with a 'Steady' state. Transitions are labeled with conditions such as `[~Brake && ... Speed<=maxspeed && ... Speed>=mintspeed]` and `[hasChangedTo(AccelResSw,true) ... &&tspeed!=uint8(0)]`.
- Code:** A snippet of C code from `rtwdemo_sil_topmodel.c`. It shows a function `counterTypeB` with an `if (enableB)` block. A red arrow points to the `enableB` condition, labeled 'Links to model element'. A tooltip over the `enableB` block shows 'Decision covered false, but not true', labeled 'Tooltip with code coverage results'. A red arrow points to the `if (enableB)` line, labeled 'Coverage annotation'.

Generating test reports for audits and reviews

Create Test Result Report

Title Page Information

Title:

Author:

Include MATLAB version

Include in Report

Results for:

Test requirements
 MATLAB figures
 Error and log messages
 Simulation metadata
 Coverage results
 Plots of criteria and assessments
 Plots for simulation output and baseline

2 rows x 1 columns of plots per page

Output Options

File Format:

File Name:

Customization

Template File:

Report Class:

C:\Demos\PowerWindow324\newReport.pdf

newReport.pdf 1 / 11

Report Generated by Test Manager

Title: Test
Author:
Date: 25-Mar-2020 15:29:15

Test Environment

Platform: PCWIN64
 MATLAB: (R2020a)

Switch block "[Switch](#)"

[Justify or Exclude](#)

Requirement Testing Details

Implemented Requirements	Verified by Tests	Runs
Cancel Switch Detection	Cancel button	U1.2

Parent: [crs_control](#)

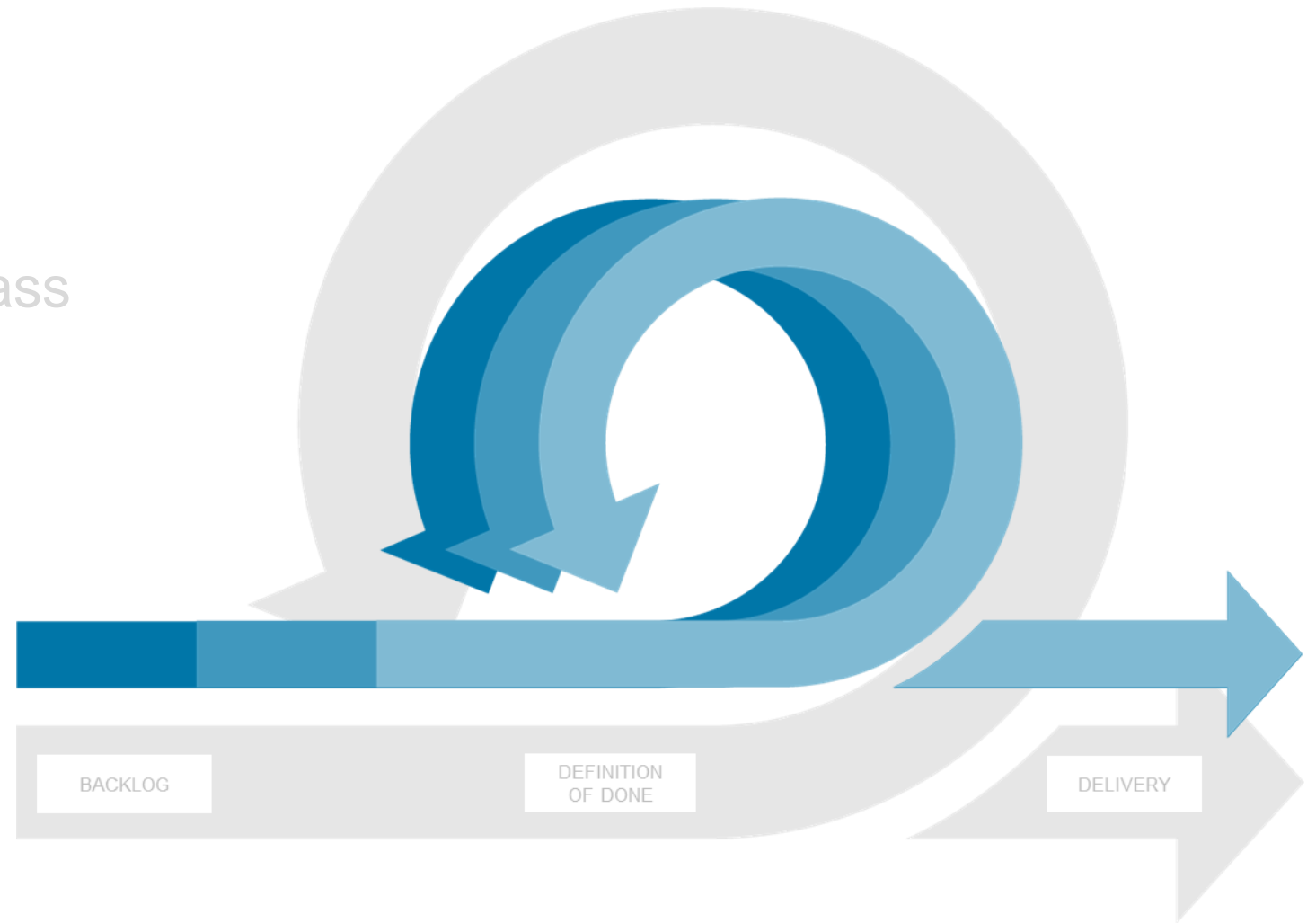
Metric

Cyclomatic Complexity 1
 Decision 100%
 Execution 100%

Summary

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

1. Create a test
2. Implement enough for test to pass
3. **Refactor**

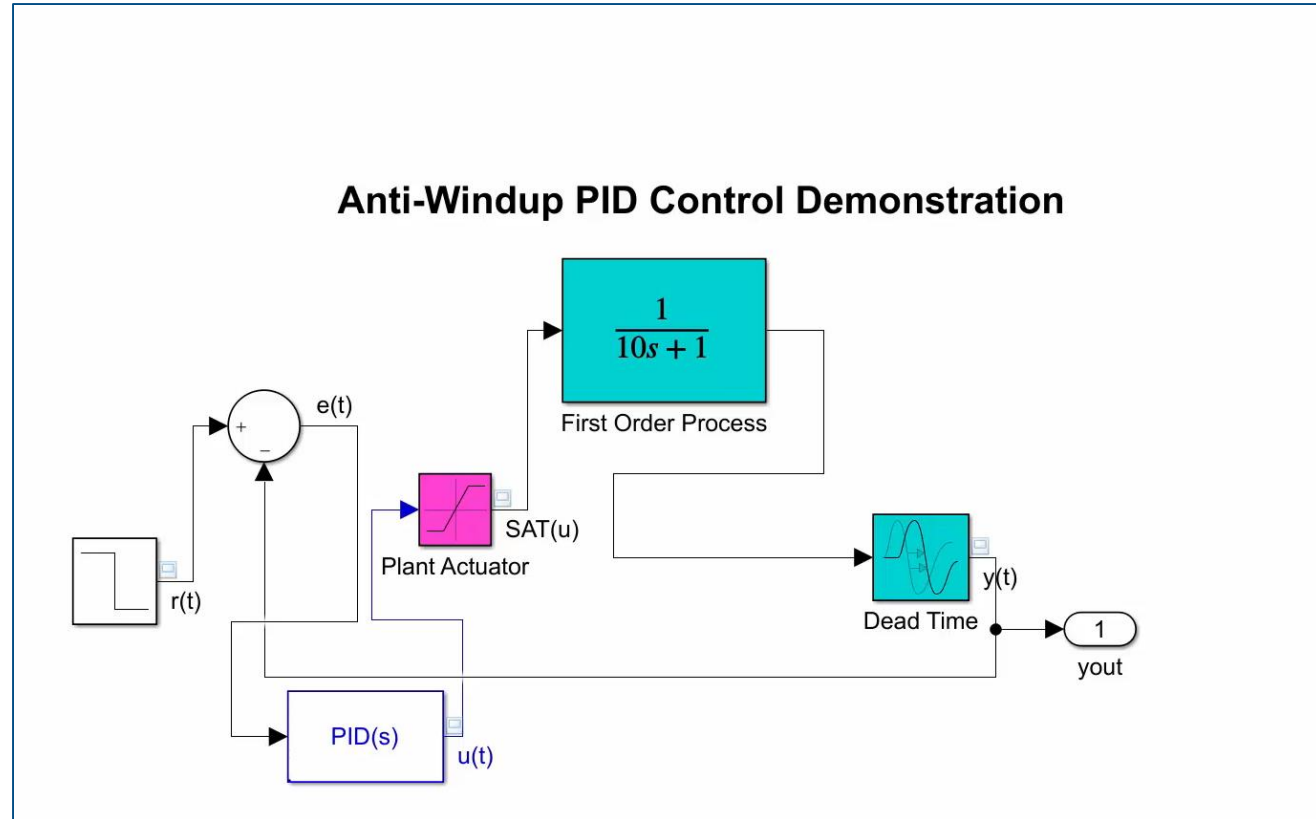


Refactoring

- Refactoring is the process of changing software in such a way that it does not alter the external behavior of the code yet improves its internal structure

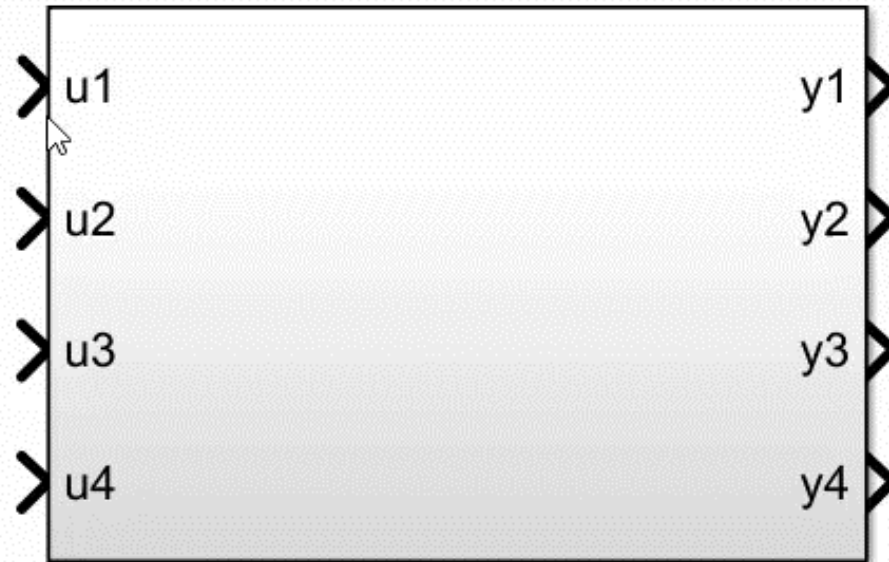
Refactoring takes many shapes and forms

- Rearranging Layout



Refactoring takes many shapes and forms

- Rearranging Layout



Refactoring takes many shapes and forms

- Rearranging Layout
- Restructuring Hierarchy

The screenshot displays the MATLAB R2019b Simulink environment. The main window shows a Simulink model titled 'sldemo_autotrans'. The 'SUBSYSTEM BLOCK' context menu is open, with the 'Referenced Subsystem' option highlighted by a red rectangle. The menu options are:

- Convert
- Is Atomic Subsystem
- Expand
- Content Preview
- Comment Out
- Comment Through
- Uncomment
- Name
- Create Mask
- Add Image
- Mask Parameters
- Lock Under Mask
- Create Subsystem Model Mask

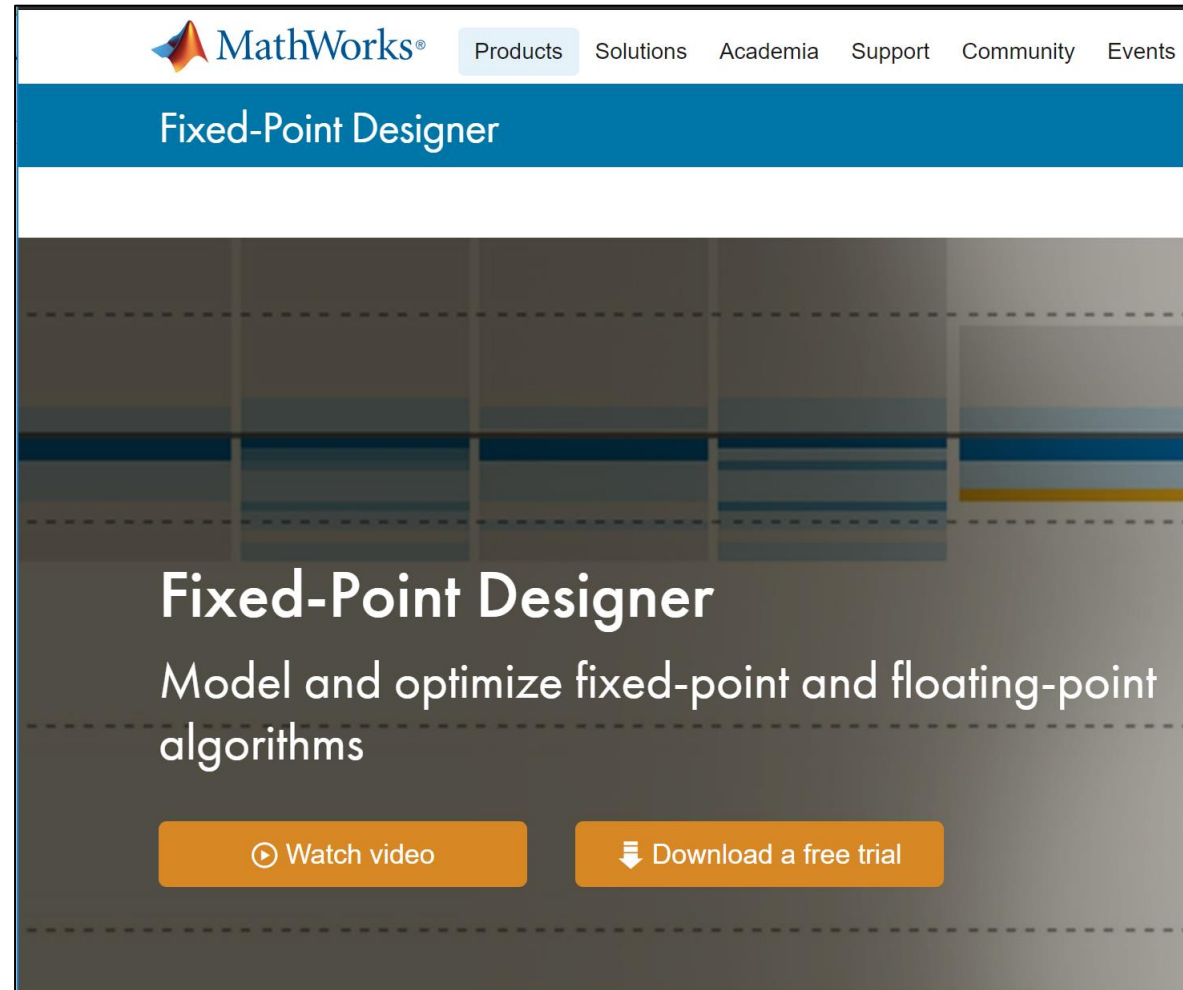
The 'CONVERT TO' section of the menu includes:

- Variant Subsystem: Convert subsystem to variant subsystem
- Referenced Subsystem: Convert subsystem to referenced subsystem** (highlighted)
- Model Block: Convert subsystem to referenced model

The background Simulink diagram shows a 'Passing Maneuver' subsystem (yellow) connected to a 'ShiftLogic' block (orange), which is connected to an 'Engine and Transmission Subsystem' block (blue). The 'Engine and Transmission Subsystem' block is connected to a 'Vehicle' block (car icon) and a 'PlotResults' block (yellow). The 'Vehicle' block outputs 'VehicleSpeed' to the 'PlotResults' block. The 'ShiftLogic' block outputs 'Gear' to the 'Engine and Transmission Subsystem' block. The 'Engine and Transmission Subsystem' block outputs 'Throttle' and 'TransmissionRPM' to the 'Vehicle' block. The 'Vehicle' block outputs 'BrakeTorque' to the 'Engine and Transmission Subsystem' block. The 'Passing Maneuver' subsystem outputs 'Throttle' and 'Brake' to the 'ShiftLogic' block. The 'ShiftLogic' block outputs 'VehicleSpeed' to the 'Engine and Transmission Subsystem' block.

Refactoring takes many shapes and forms

- Rearranging Layout
- Restructuring Hierarchy
- Optimizing Implementation

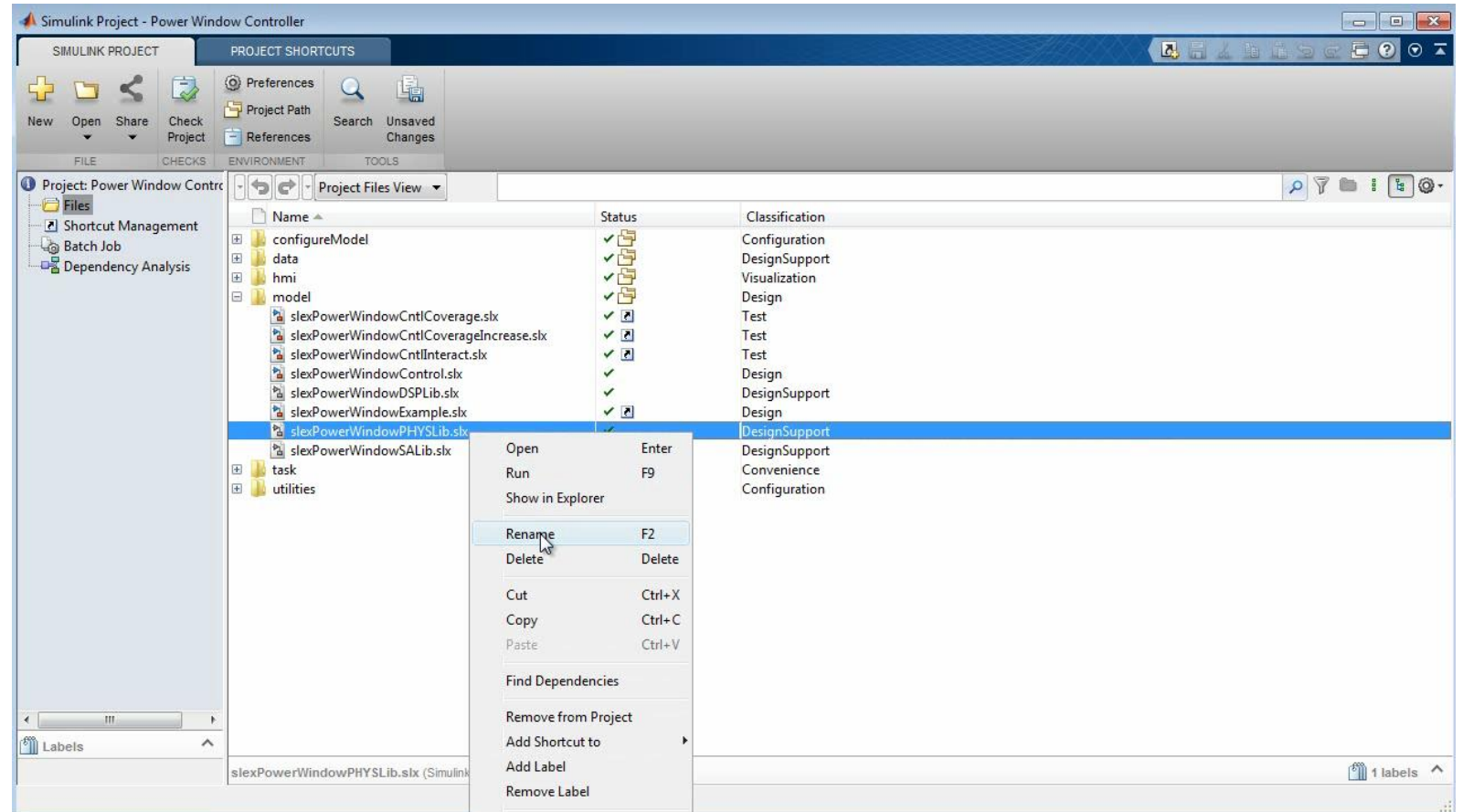


The screenshot shows the MathWorks website for the Fixed-Point Designer product. At the top, the MathWorks logo is on the left, and navigation links for Products, Solutions, Academia, Support, Community, and Events are on the right. Below the navigation is a blue header bar with the text "Fixed-Point Designer". The main content area features a dark background with a grid pattern and a central text block that reads "Fixed-Point Designer" followed by "Model and optimize fixed-point and floating-point algorithms". At the bottom of this section are two orange buttons: "Watch video" and "Download a free trial".

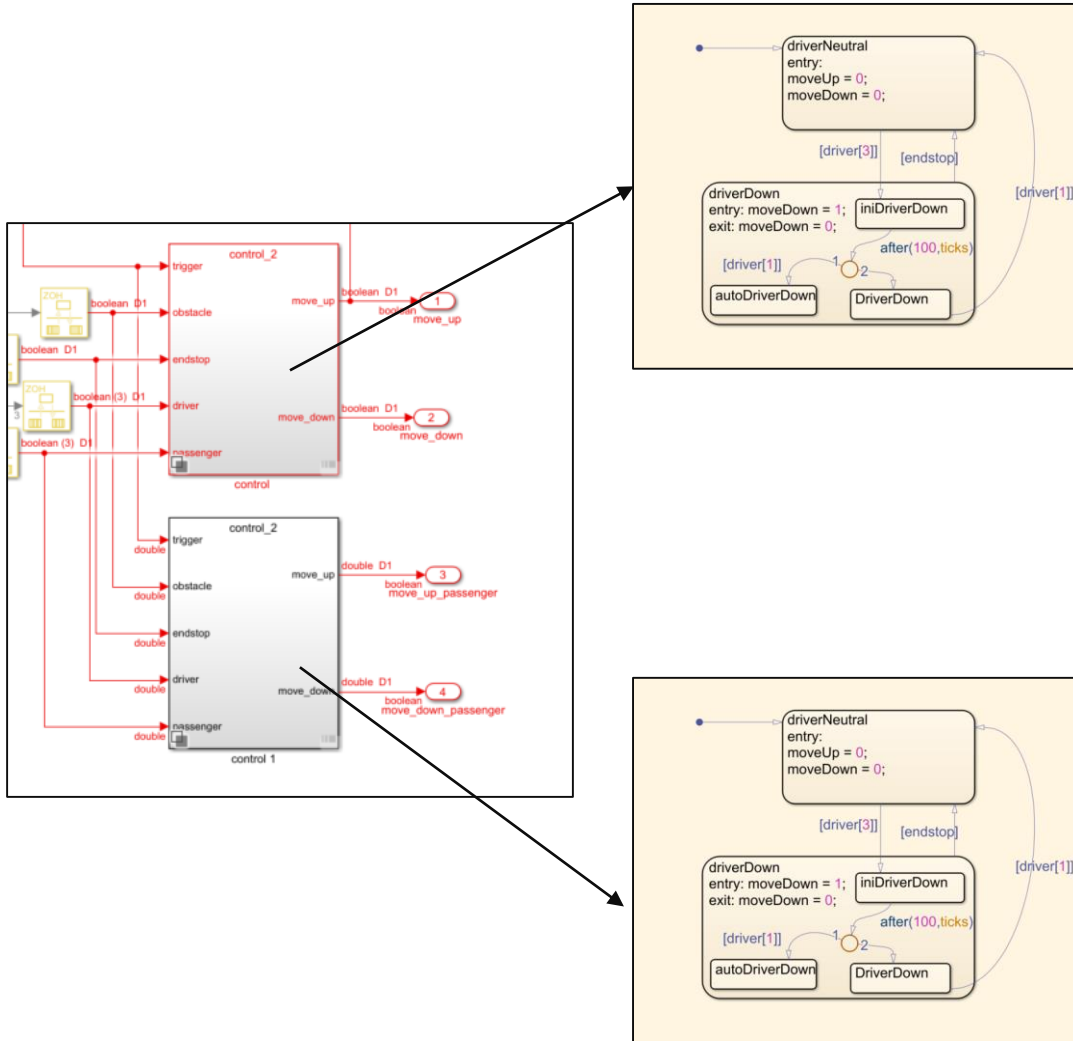
Refactoring takes many shapes and forms

- Rearranging Layout
- Restructuring Hierarchy
- Optimizing Implementation
- Project-wide Renaming

.... and many more!



Refactor by consolidating redundant Stateflow chart



Driver and Passenger Controls are identical

Detecting clones with Clone Detector App

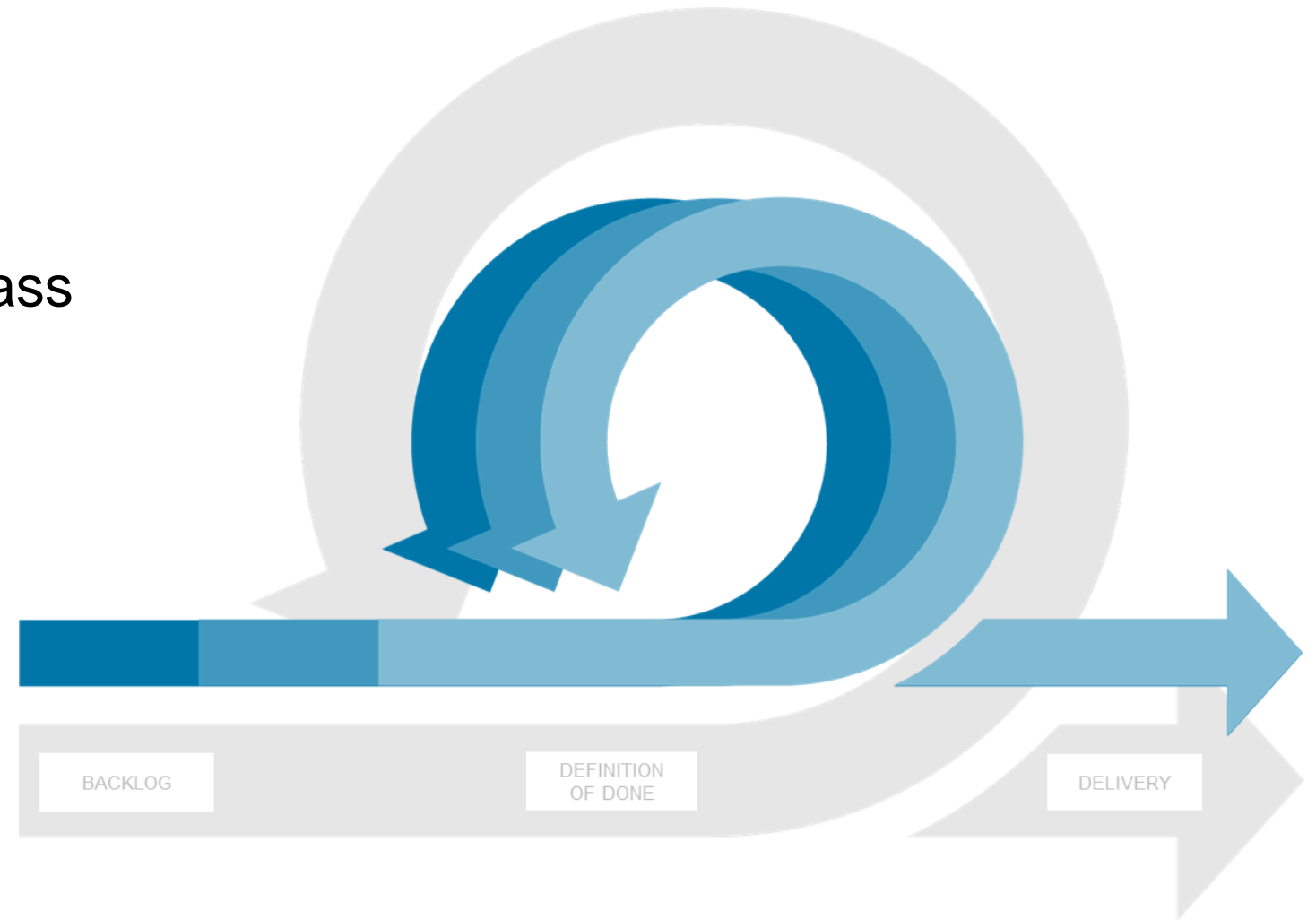
The screenshot displays the Clone Detector App interface within a Simulink environment. The main workspace shows a Simulink model with several blocks highlighted in red, indicating detected clones. A callout box highlights the 'Replace Clones' button, which is used to create library blocks from clones and replace them with library links.

The 'Clone Detection Actions and Results' window is open, showing the following table of results:

Edit	Clones	Number of clones	Blocks Per Clone	Block Difference	Parameter Difference	Library Path to Replace Clones	Similarity Plot
>	Exact Clone Group 1	2	24	0	0	newLibraryFile	
>	Exact Clone Group 2	2	24	0	0	newLibraryFile	
>	Exact Clone Group 3	2	1	0	0	newLibraryFile	
>	Exact Clone Group 4	2	1	0	0	newLibraryFile	
>	Similar Clone Group 1	4	17	1	1	newLibraryFile	

Test Driven Development Cycle

1. Create a test
2. Implement enough for test to pass
3. Refactor



Conclusion and key takeaways



Simulink provides an integrated framework for TDD

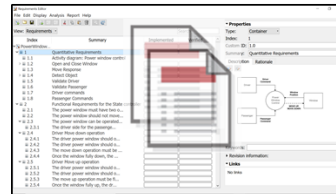


Systematically verify requirements

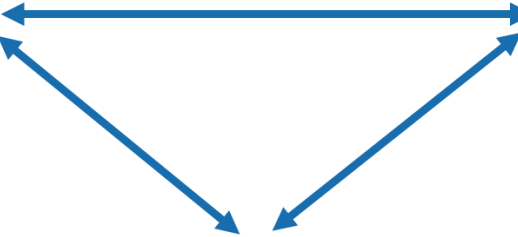
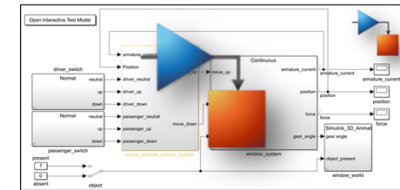


Automate testing to deliver working systems faster

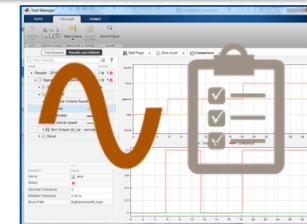
Requirements



Implementation



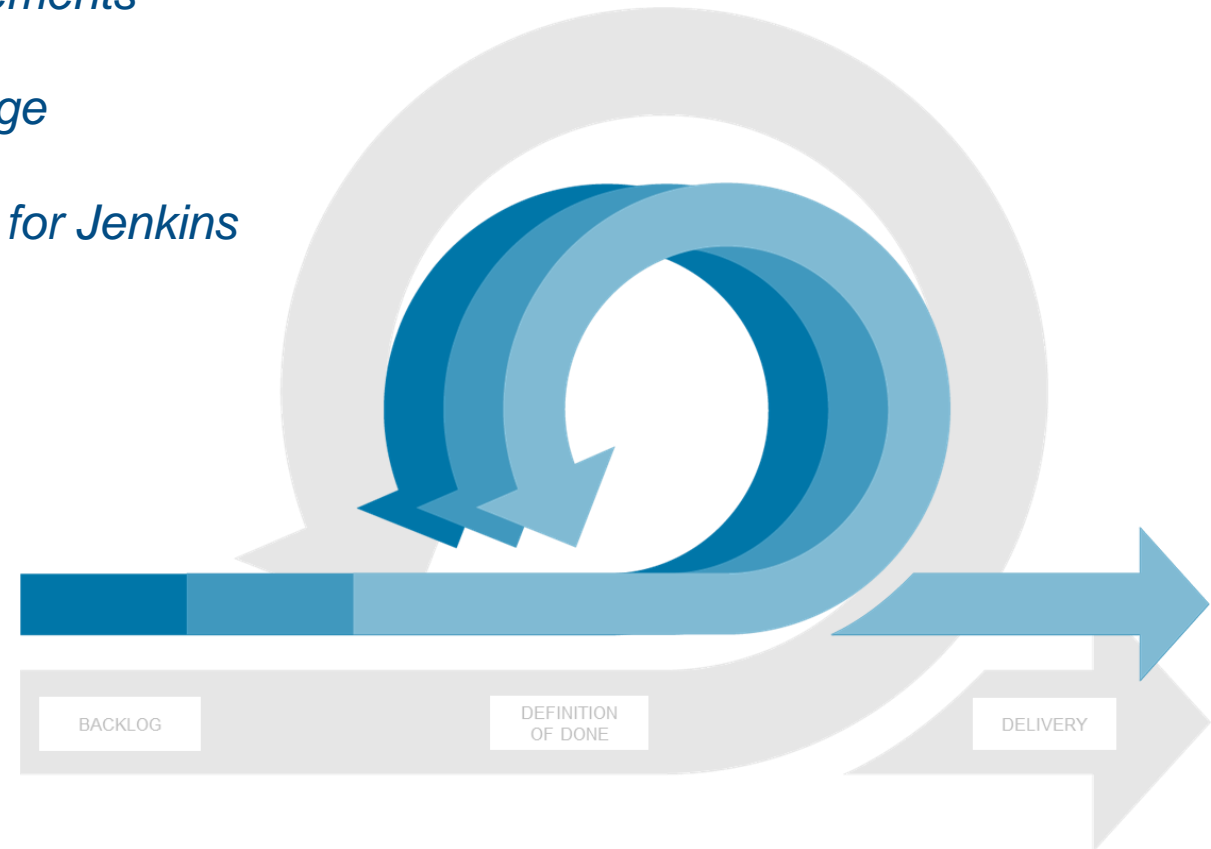
Test



Test Driven Development

powered by MATLAB and Simulink

- **Model-Based Design**
 - *Simulink and Stateflow*
 - *Simulink Requirements*
 - *Simulink Test*
 - *Simulink Coverage*
 - *Simulink Check*
 - *MATLAB Plug in for Jenkins*
 - *Projects*
- **Manage Requirements**
- **Author and Execute Tests**
- **Measure Test Completeness**
- **Refactor and Verify Compliance**
- **Continuous Integration**
- **Organize, Manage and Share**



Learn more

- [Agile System Development with Model-Based Design](#)
- [Agile Model-Based Design: Accelerating Simulink Simulations in Continuous Integration Workflows](#)
- [Verification, Validation, and Test Solution Page](#)
- [Continuous Integration Solution Page](#)