



Sviluppo di un sistema di sospensioni semiattive mediante Model-Based Design con architettura AUTOSAR e conforme allo standard A-SPICE

Presented by:

Andrea Palazzetti

Milano
25/06/2019

Roma
26/06/2019



Ride Dynamics

MATLAB EXPO 2019

Marelli - Ride Dynamics

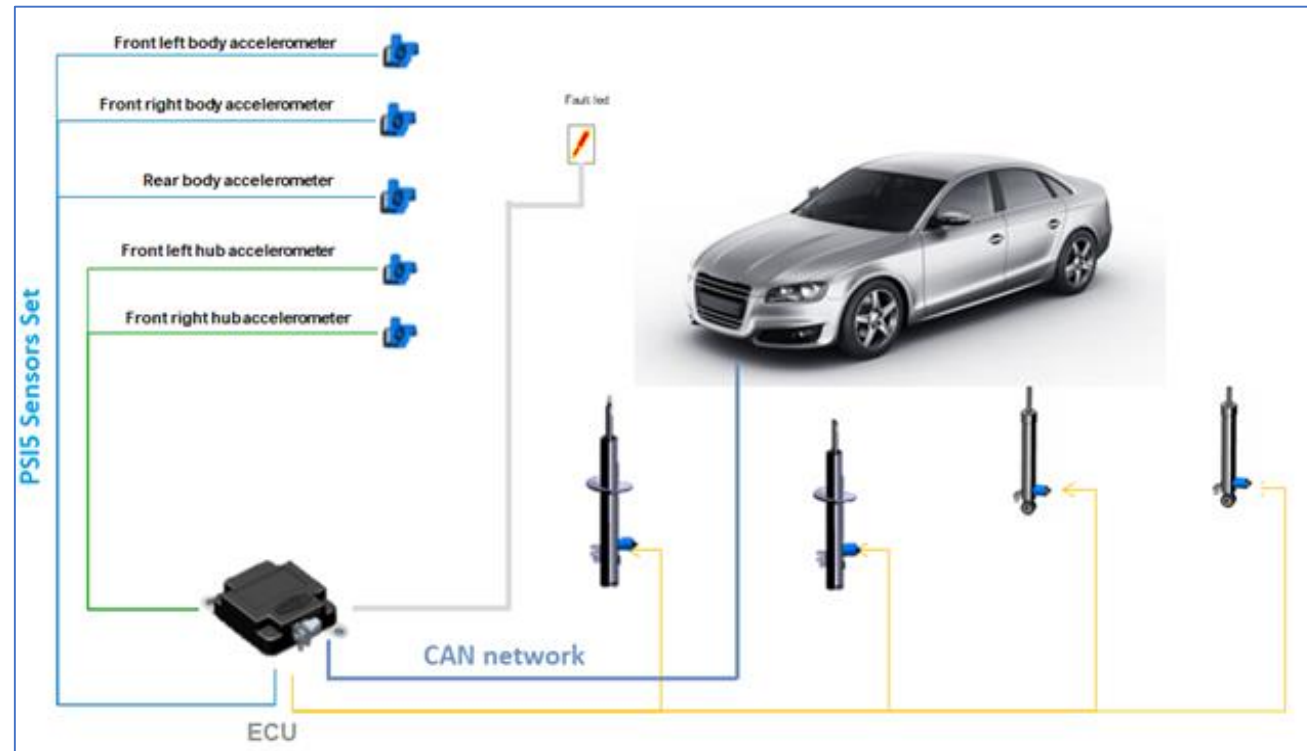


- Marelli – Ride Dynamics – Mechatronic team
 - Design and development of semi-active suspensions system
 - Responsible for the whole system
- Mechatronic's team is based in Turin
- ECU Application Software development
 - Shock Absorber damping force control strategies and diagnosis

Smart Damping Control System



- SDC system consists of
 - 4 shock absorbers with one proportional EV each
 - 5 accelerometers
 - ECU for closed loop damping control



Key Takeaways



- “State of the art”: AUTOSAR and A-SPICE development process
- Short time to market
- Focus on bidirectional traceability
- One single development environment for all SW related processes

Software development: goals and challenges



- State of-the-art for embedded automotive application software
 - Model-Based Design and automatic code generation
 - AUTOSAR Software architecture
 - Development process compliant to A-SPICE reference model
- Such a development process and SW architecture are required by main OEMs
- Constraint: Short time to market

AUTOSAR



What is AUTOSAR?



AUTOSAR – AUTomotive Open Systems ARchitecture

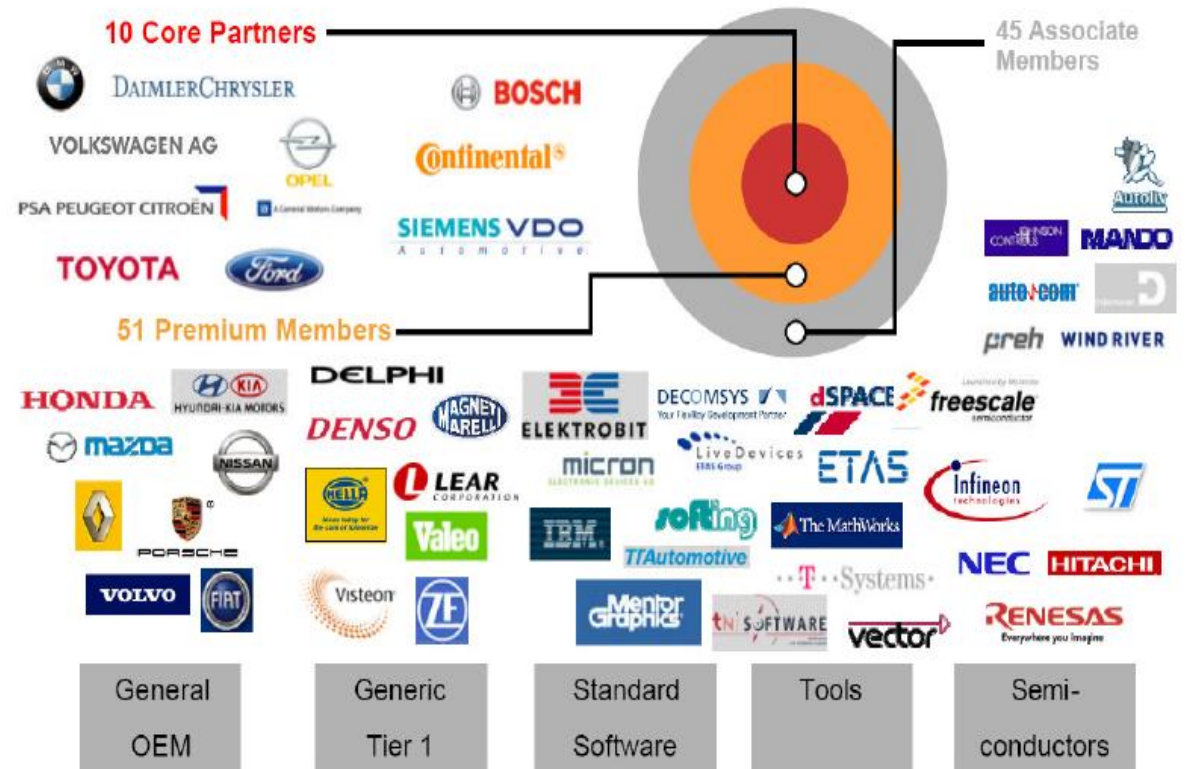
Middleware and system-level standard, jointly developed by automobile manufacturers, electronics and software suppliers and tool vendors.

More than 100 members

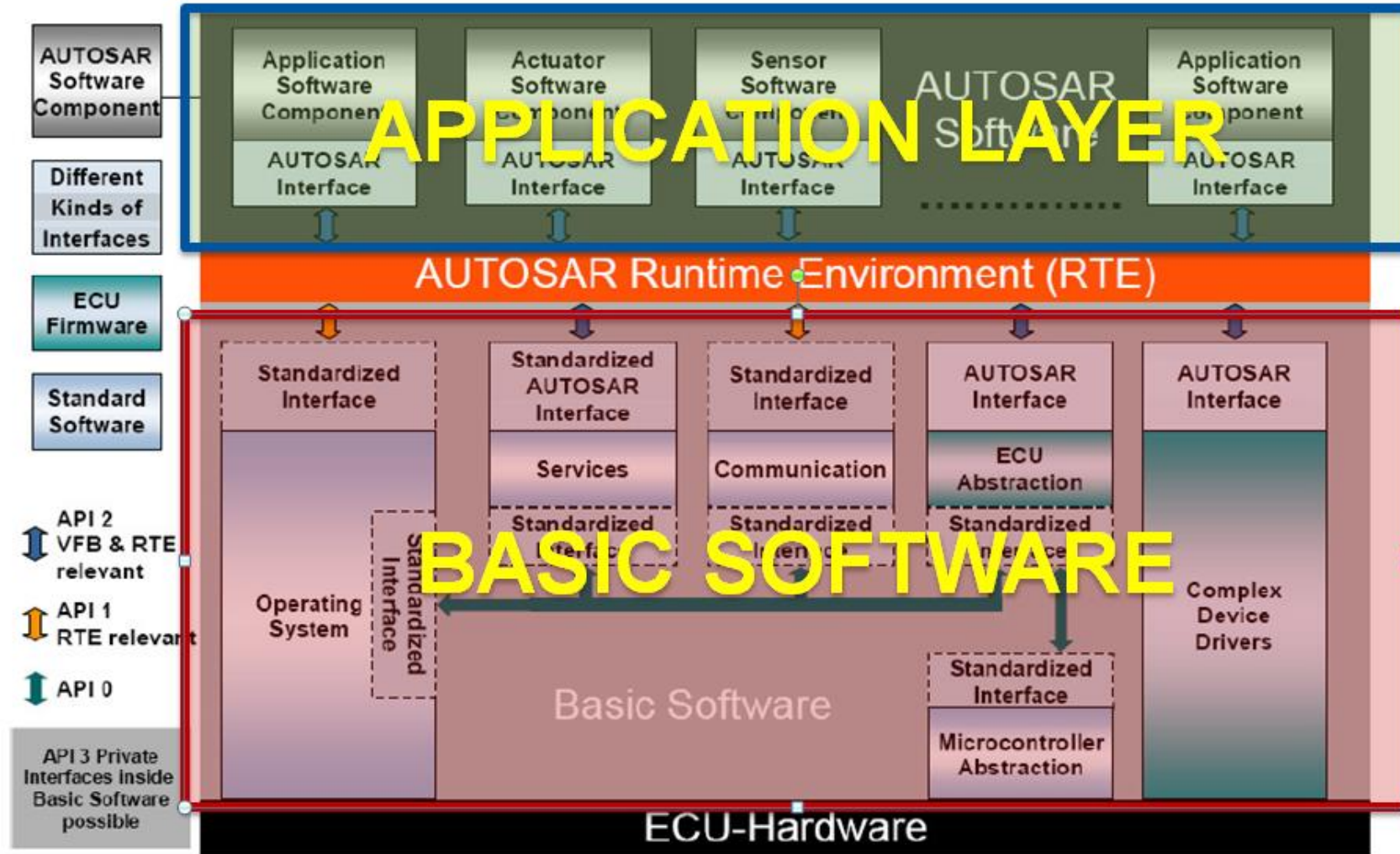
Motto: *“cooperate on standards, compete on implementations”*

Reality: current struggle between OEM and Tier1 suppliers

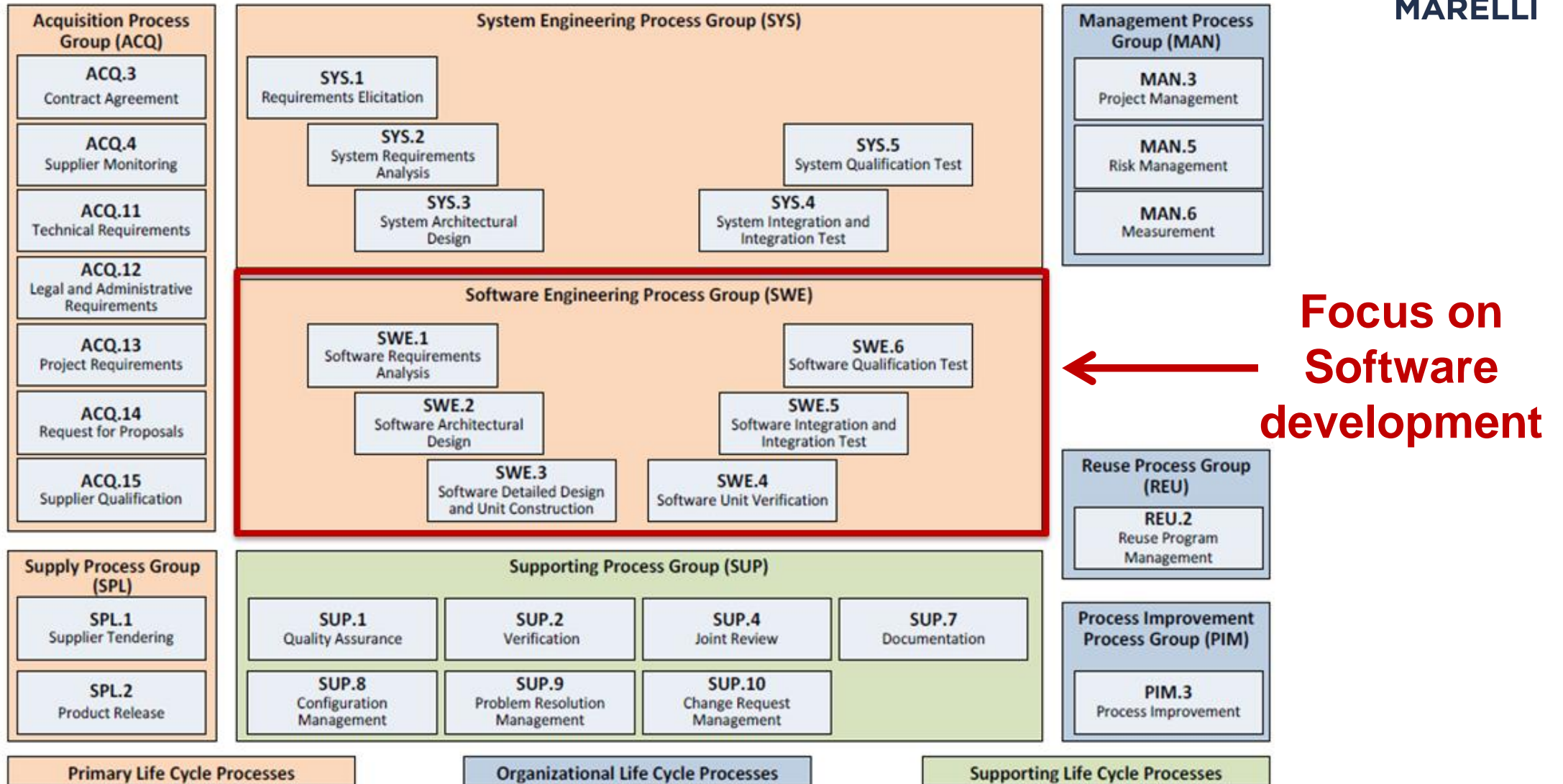
Target: facilitate portability, composability, integration of SW components over the lifetime of the vehicle



AUTOSAR ECU SW architecture



Automotive SPICE process reference model




Subset of recommended A-SPICE base practices

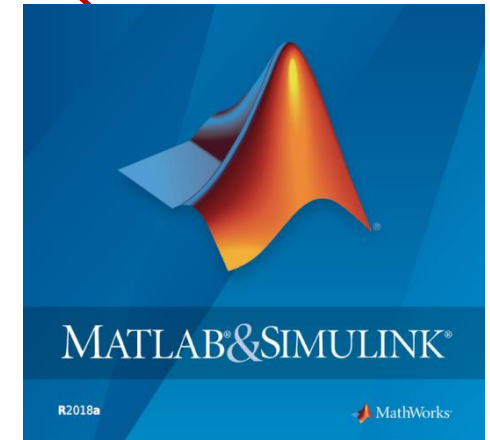
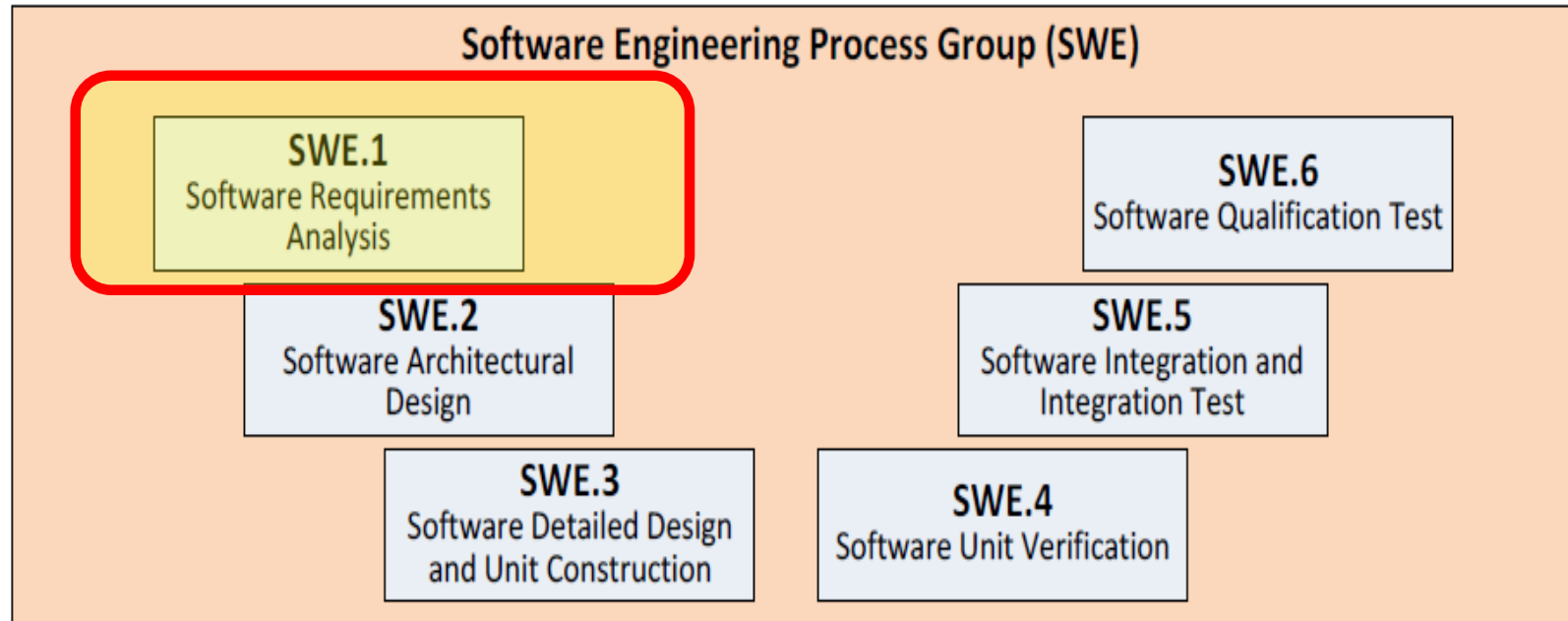


- Specify software requirements
- Structure software requirements
- Establish **bidirectional traceability** between
 - software and system requirements
 - software requirements and software architectural element
 - software requirements and software units
 - software detailed design and the unit test specification
 - elements of the software architectural design and test cases
 - software qualification test specification and software qualification test results
- Develop a detailed design for each software component
- Define interfaces of software elements
- Define interfaces of software units.

**Focus on
traceability**

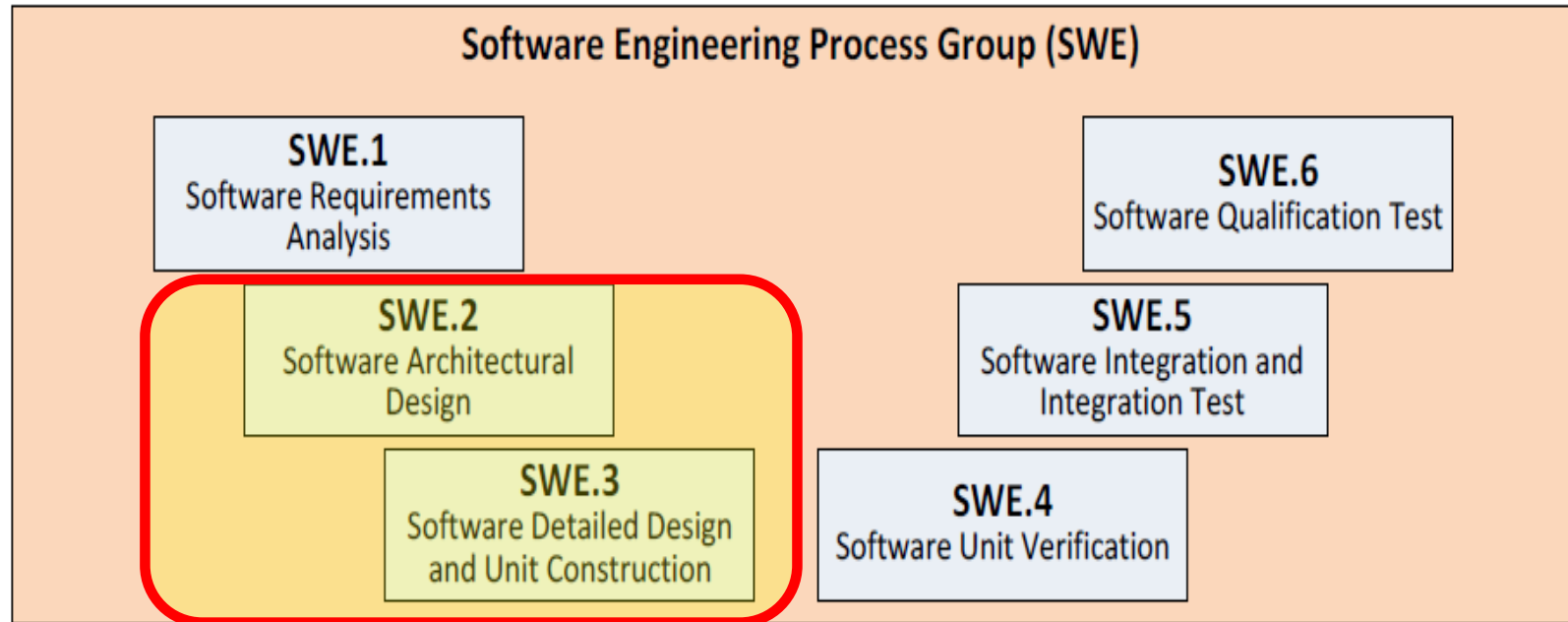
A red arrow originates from the text 'Focus on traceability' and points towards the underlined text 'bidirectional traceability' in the list item above.

Whole SW development tool set based on MATLAB & Simulink R2018a



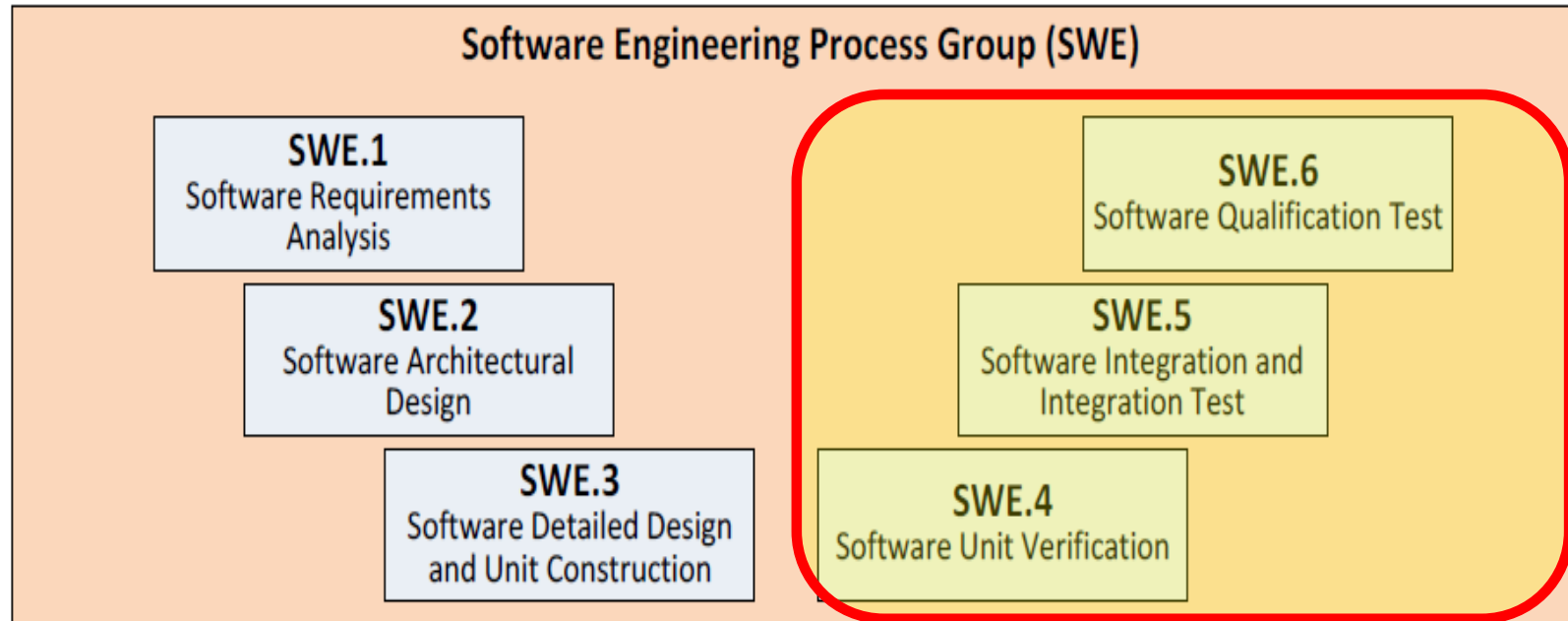
Simulink Requirements: requirements specification

Whole SW development tool set based on MATLAB & Simulink R2018a



- Simulink – Stateflow: SW units design and simulation
- Simulink Check and Design Verifier : coding guidelines check- Simulink model analysis
- Embedded Coder - Support package for Autosar: SW Components' AUTOSAR interfaces design and C-code autogeneration

Whole SW development tool set based on MATLAB & Simulink R2018a



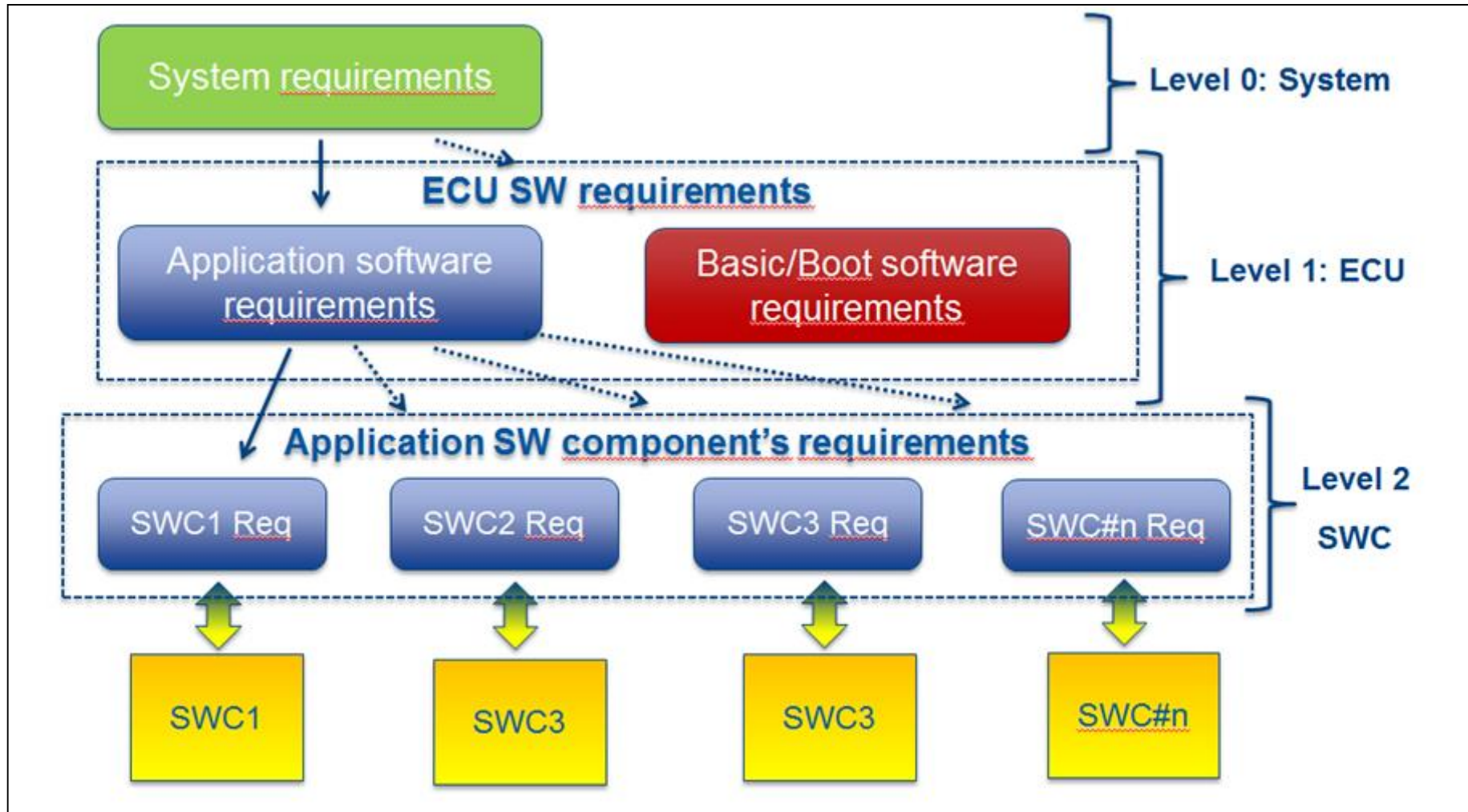
- Simulink Test: Unit testing – MIL testing
- Simulink Coverage: for testing coverage metrics

Subset of recommended A-SPICE base practices



- Specify software requirements
- Structure software requirements
- Establish bidirectional traceability between
 - software and system requirements
 - software requirements and software architectural element
 - software requirements and software units
 - software detailed design and the unit test specification
 - elements of the software architectural design and test cases
 - software qualification test specification and software qualification test results
- Develop a detailed design for each software component
- Define interfaces of software elements
- Define interfaces of software units.

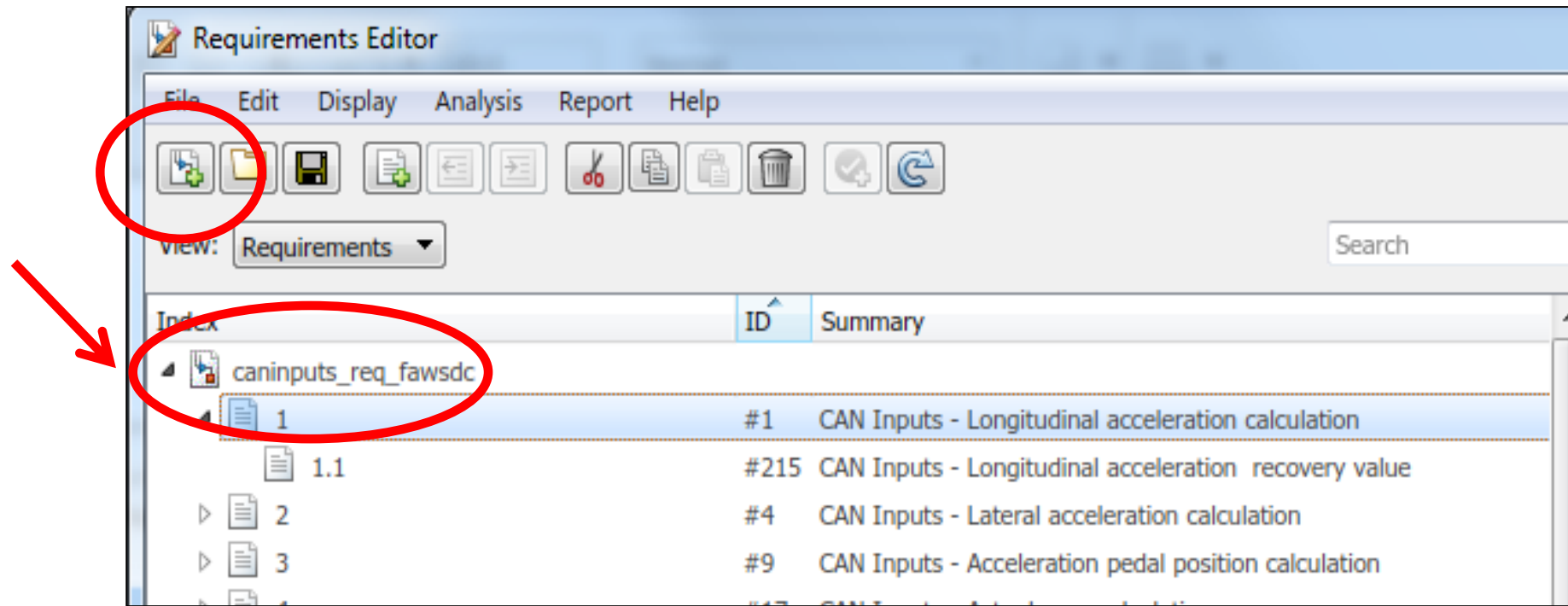
Requirements' structure: three levels



SW requirements specification



- Simulink Requirements is used for requirements specification and linking
- Several “Requirement sets” used for grouping requirements
- One requirement set for every SW Component



Requirements set: example



Item ID	Description
#1	ECU SW - CAN Message: Prohibited frame identifier
#2	ECU SW - CAN Message: Remote frame
#3	ECU SW - CAN Message: Reserved fields filling
#4	ECU SW - CAN Message: message parameter T
#5	ECU SW - CAN Message: event message parameter N
#8	ECU SW - CAN Message: livecounter computation
#9	ECU SW - CAN Message: livecounter monitoring condition
#10	ECU SW - CAN Message: livecounter management
#11	ECU SW - CAN Message: checksum computation
#12	ECU SW - CAN Message: checksum location
#13	ECU SW - CAN Message: checksum management
#14	ECU SW - CAN Stop Condition
#88	ECU SW - CAN CDC Reception
#13.1	ECU SW - CAN signal conversion

Item ID	Description
#1	CAN Inputs - Longitudinal acceleration calculation
#4	CAN Inputs - Lateral acceleration calculation
#9	CAN Inputs - Acceleration pedal position calculation
#17	CAN Inputs - Actual gear calculation
#41	CAN Inputs - Brake pedal status calculation
#49	CAN Inputs - Combustion torque calculation
#54	CAN Inputs - Drive style status calculation
#62	CAN Inputs - Engine speed calculation
#8.1	CAN Inputs - Engine speed punctual recovery
#66	CAN Inputs - Gradient acceleration pedal calculation
#70	CAN Inputs - ABS intervention calculation

level 1: ECU SW

level 2: sw component

Bidirectional traceability: Simulink Requirements view



The screenshot displays the Simulink Requirements view for a requirement. The 'Properties' section includes 'Index: 1', 'Custom ID: #1', and 'Summary: CAN Inputs - Longitudinal acceleration calculation'. The 'Description' tab is active, showing the text: 'Longitudinal acceleration shall be calculated as: Ax_out= Acceleration_X *0.027 - 21.593 [m/s^2]'. Below this is a 'Keywords' field. The 'Revision information' section is collapsed. The 'Custom Attributes' section is expanded and circled in red, showing fields for 'CR' (checkbox), 'Plan' (FAW1), 'Reference' (CAN matrix), 'Status' (Approved), and 'Verification' (Unit Test). The 'Links' section is also expanded and circled in red, showing 'Implemented by' (Raw2Phy26, Conversion_Recover) and 'Verified by' (ax_testfile, ax_test_suite, ax test case 1). At the bottom, 'Related to' links are shown: #158 ECU SW - CAN ESP 3: Acceleration_X and #185 ECU SW - CAN signal conversion.

← Requirement specification

← Additional information

BIDIRECTIONAL LINKS

➤ link to implementation Simulink model

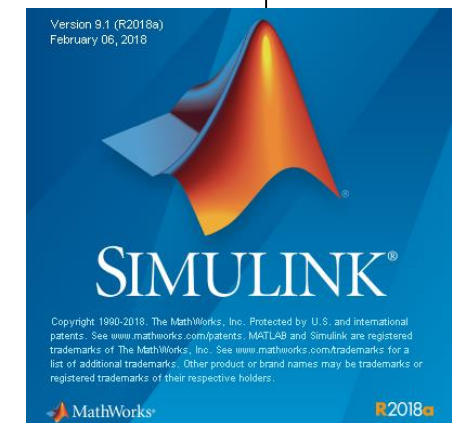
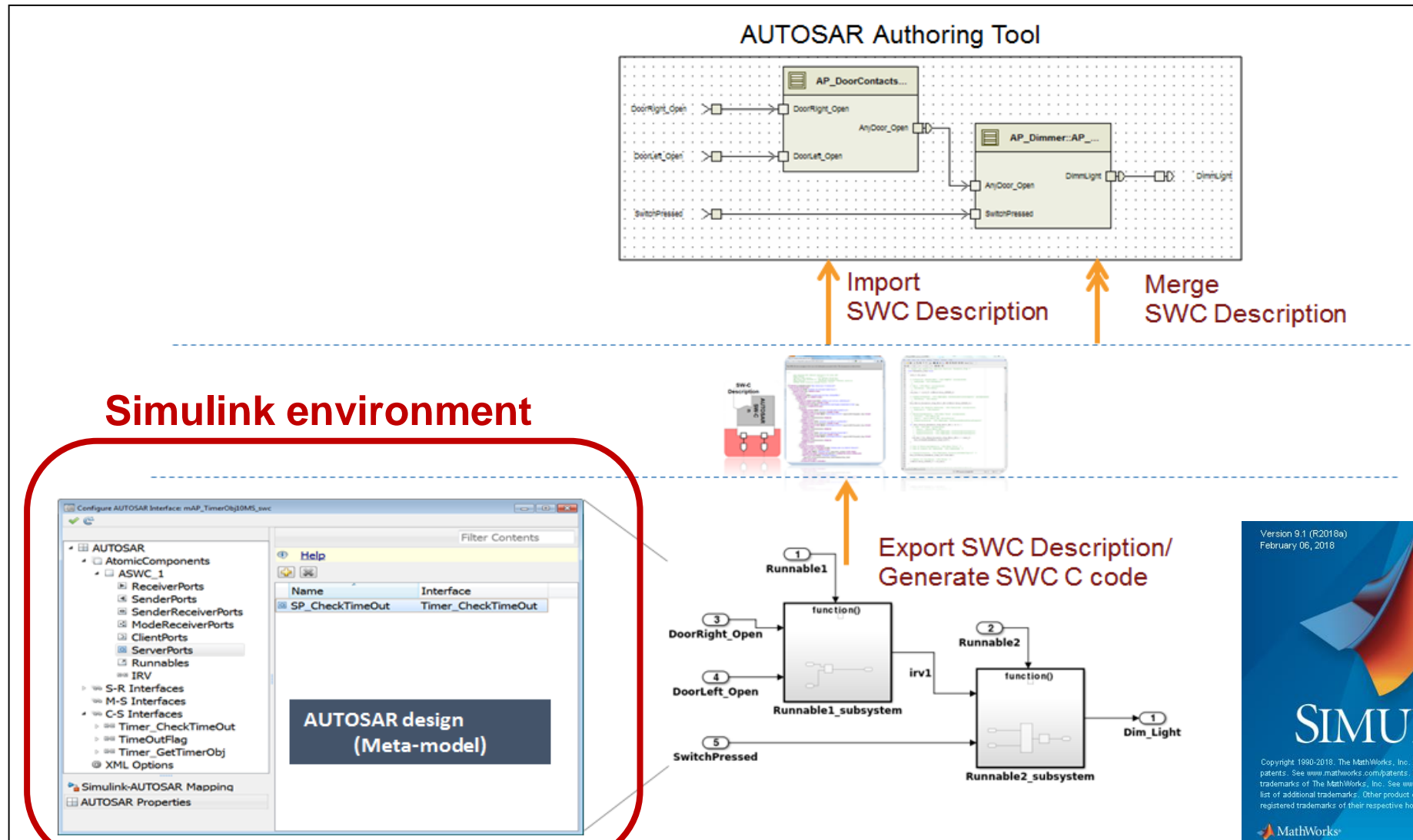
➤ link to verification harness model

Subset of recommended A-SPICE base practices

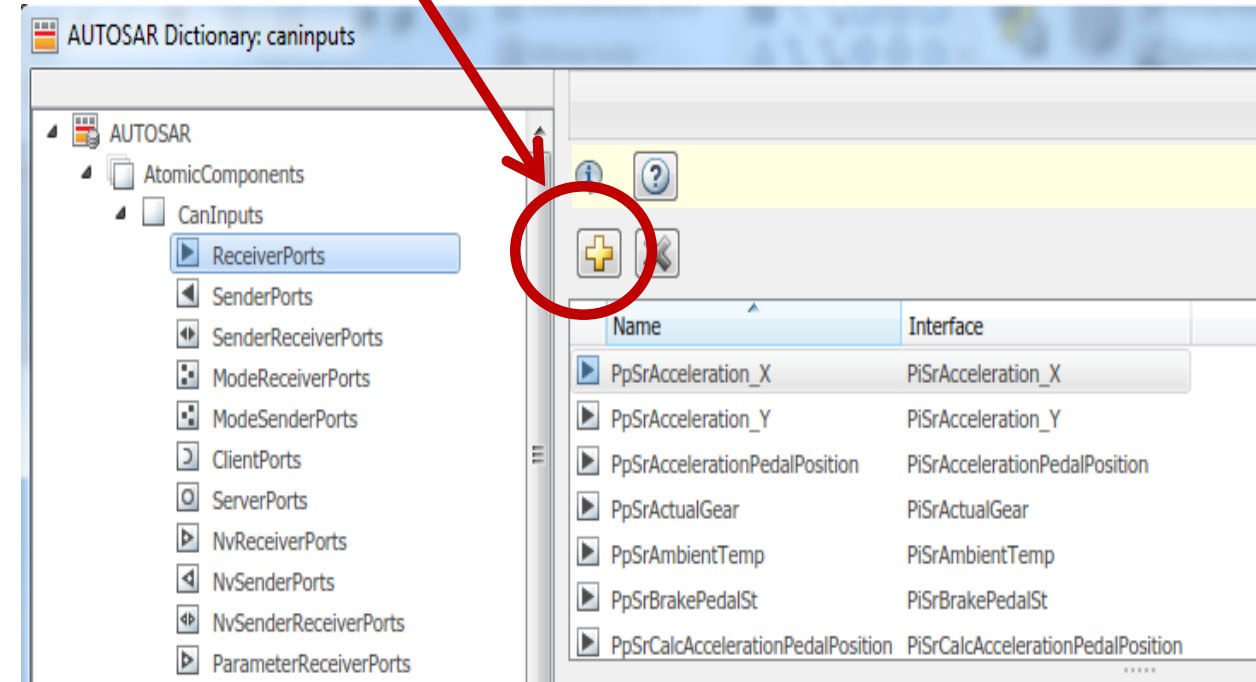
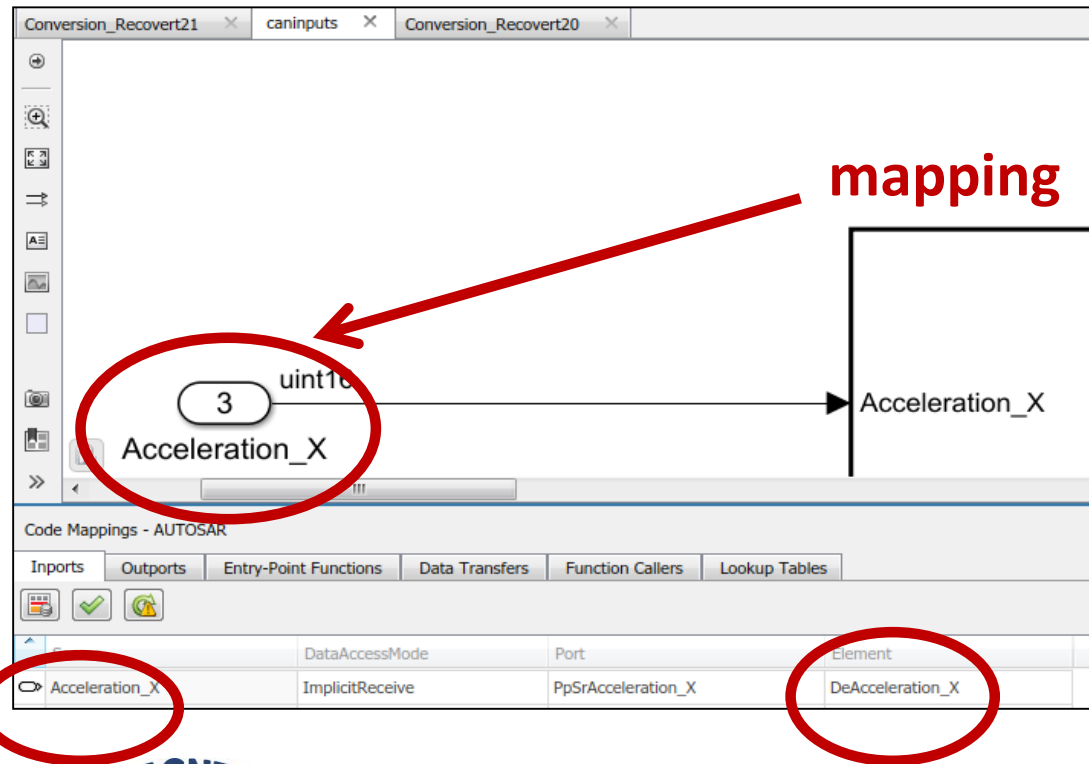
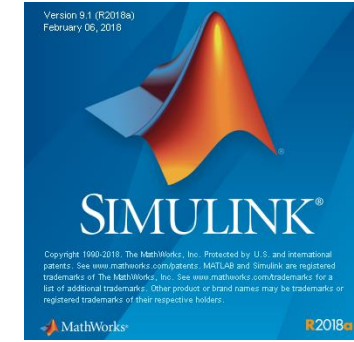
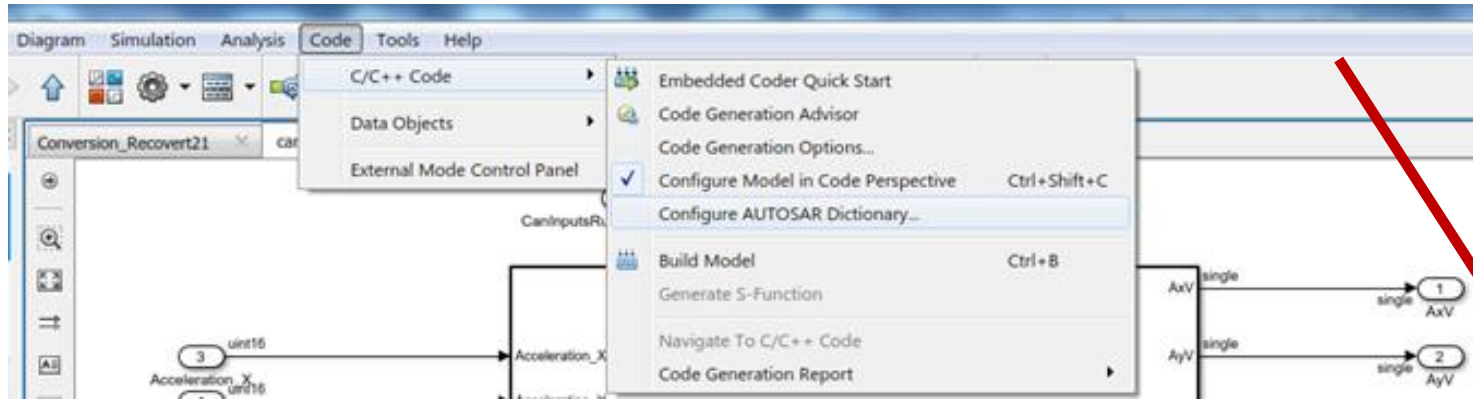


- Specify software requirements
- Structure software requirements
- Establish bidirectional traceability between
 - software and system requirements
 - software requirements and software architectural element
 - software requirements and software units
 - software detailed design and the unit test specification
 - elements of the software architectural design and test cases
 - software qualification test specification and software qualification test results
- Develop a detailed design for each software component
 - Define interfaces of software elements
 - Define interfaces of software units.

AUTOSAR INTERFACES design : BOTTOM-UP APPROACH



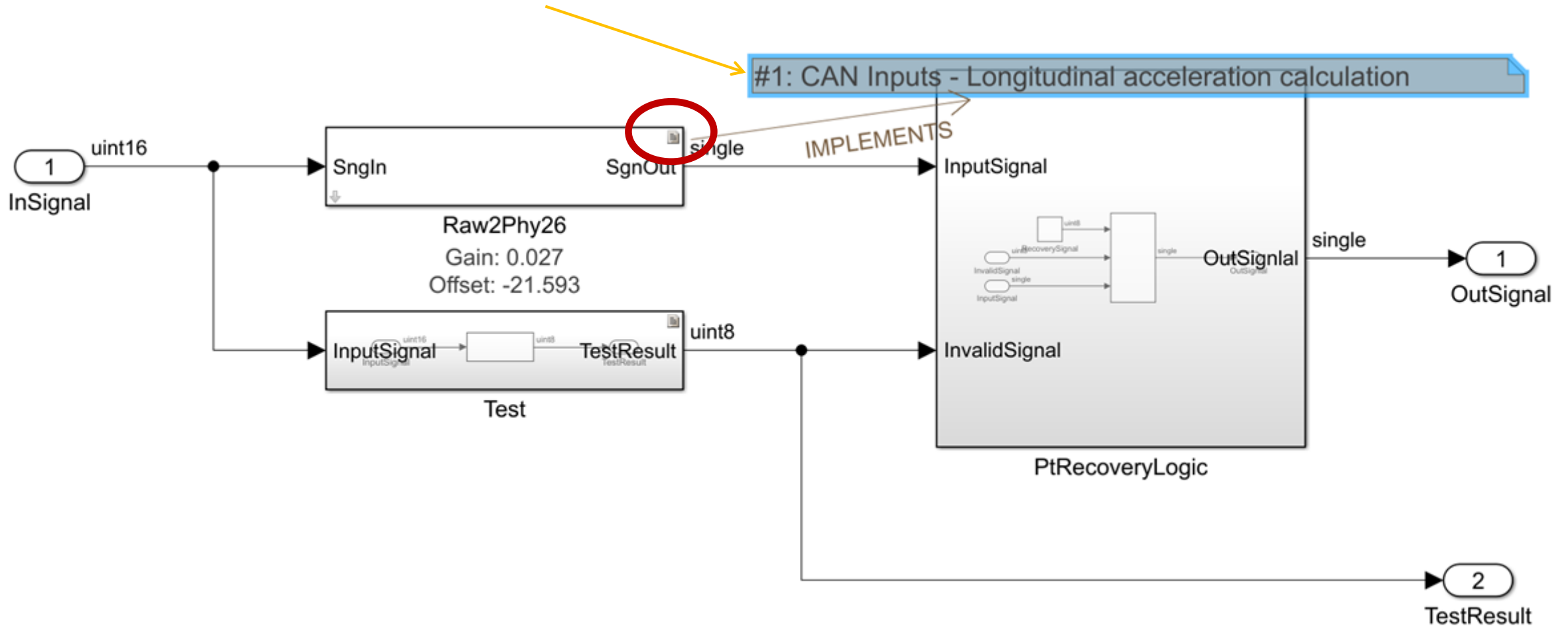
Adding and mapping an AUTOSAR PORT



Detailed design



Bidirectional linking to implemented requirement



Data dictionary



Data Dictionary specifies: tuneable parameters, measurable variables, constants, bus object ...

The screenshot shows the MATLAB Model Explorer interface. On the left, the Model Hierarchy tree is expanded to 'caninputs' > 'Reference (Active)'. The central pane shows a table with columns 'Name' and 'BlockType'. The entry 'InCan_O_AxV' is highlighted with a red oval, and a red arrow points to it from the text 'Example: measurable variable'. The right pane shows the properties for 'Simulink.Signal: InCan_O_AxV'. The 'Unit' property is set to 'm_s2' and circled in red. The 'Description' property is set to 'Vehicle longitudinal acceleration' and also circled in red.

Name	BlockType
InCan_O_AxV	

Simulink.Signal: InCan_O_AxV

Data type: single

Dimensions: -1 Dimensions mode: auto

Initial value: Complexity: auto

Minimum: [] Maximum: []

Unit: m_s2 Sample time: -1

Code generation options

Storage class: ExportedGlobal

Alias:

Alignment: -1

Description: Vehicle longitudinal acceleration

Example: measurable variable

Model Advisor: Model check before code generation



Modeling standards: MAAB

- MAAB/JMAAB Checks
 - ✓ Check for indexing in blocks
 - ✓ Check for prohibited blocks in discrete controllers
 - ✓ Check for prohibited sink blocks
 - ✓ Check positioning and configuration of ports
 - ✓ ⚠ Check for matching port and signal names
 - ✓ Check whether block names appear below blocks
 - ✓ ⚠ Check for mixing basic blocks and subsystems
 - ✓ Check for unconnected ports and signal lines
 - ✓ Check position of Trigger and Enable blocks
 - ✓ Check usage of tunable parameters in blocks
 - ✓ ⚠ Check the display attributes of block names
 - ✓ Check display for port blocks
 - ✓ Check subsystem names
 - ✓ Check port block names
 - ✓ Check character usage in signal labels
 - ✓ Check character usage in block names
 - ✓ Check Trigger and Enable block names
 - ✓ Check orientation of Subsystem blocks
 - ✓ Check usage of Relational Operator blocks
 - ✓ ⚠ Check font formatting
 - ✓ Check transition orientations in flow charts

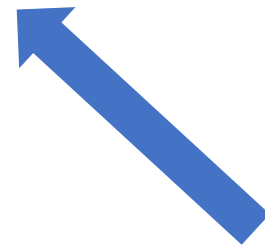
Model Advisor Report - caninputs.slx

Simulink version: 9.1
System: caninputs

Run Summary

Pass	Fail	Warning	Not Run	Total
✓ 55	✗ 0	⚠ 14	📄 0	69

MAAB/JMAAB Checks



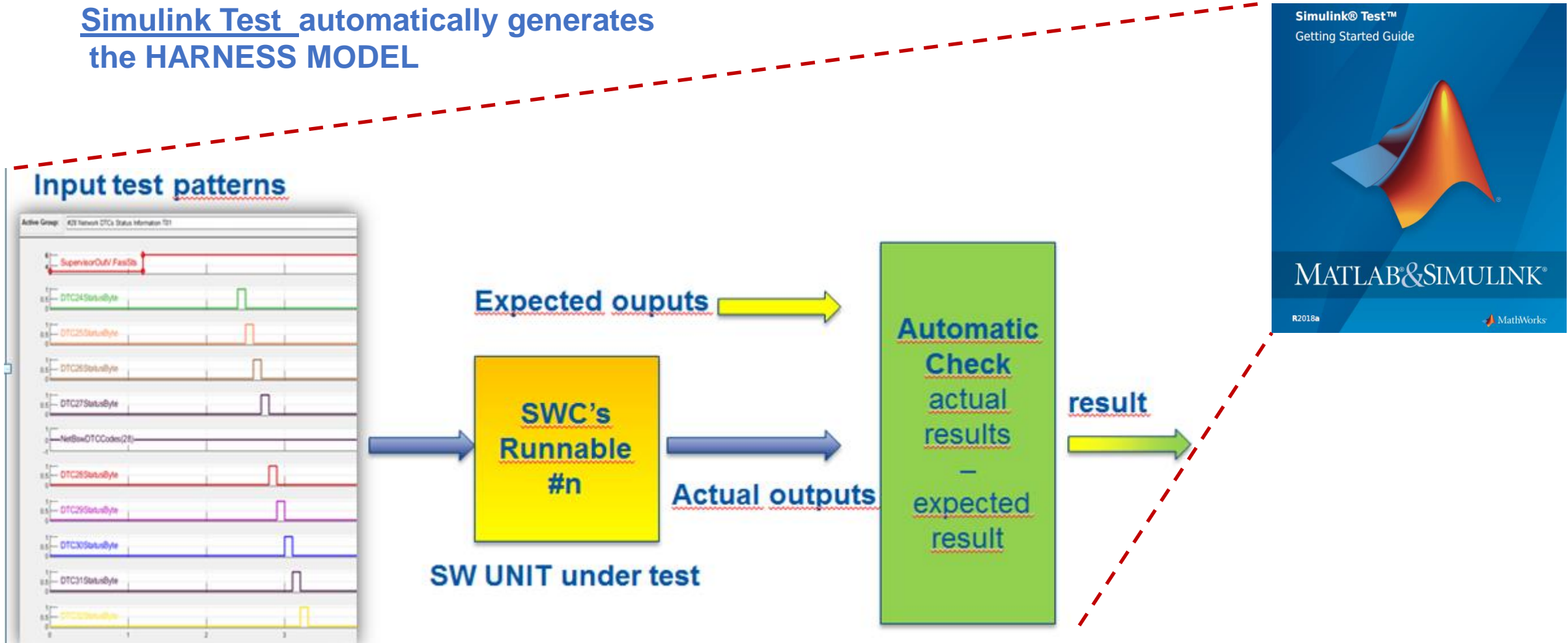
Automatic report

MAAB rules automatically checked

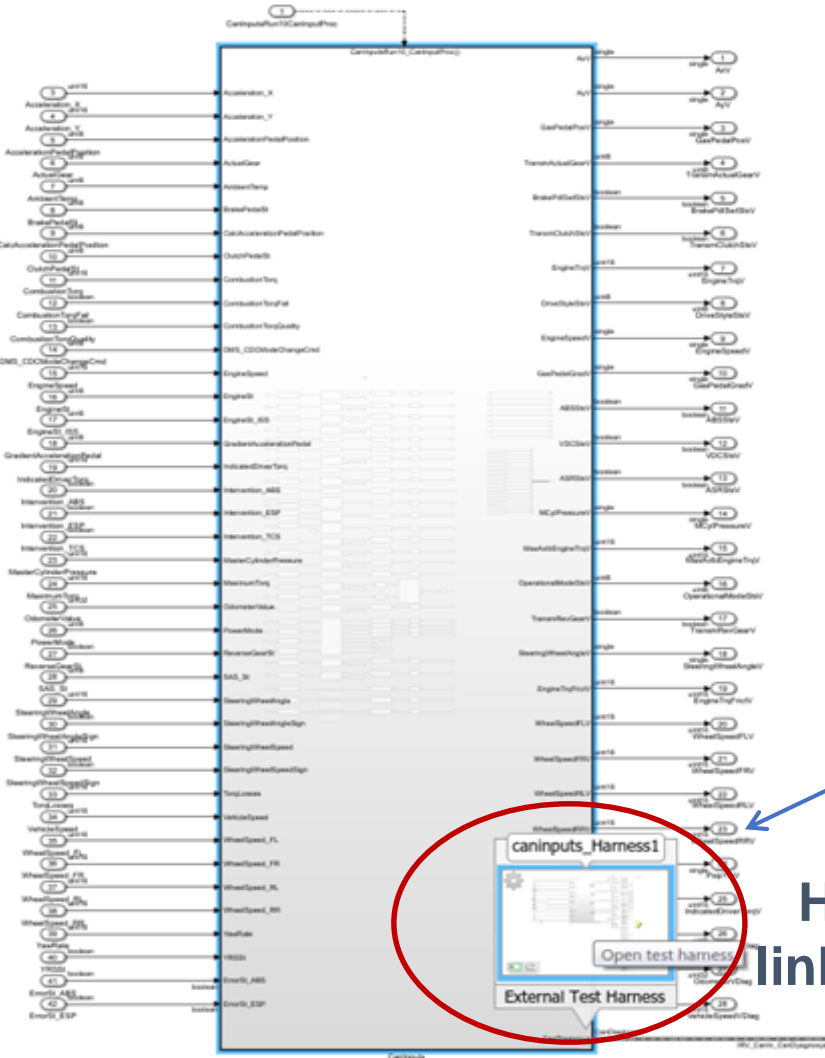
Unit testing: Harness Model



Simulink Test automatically generates the HARNESS MODEL

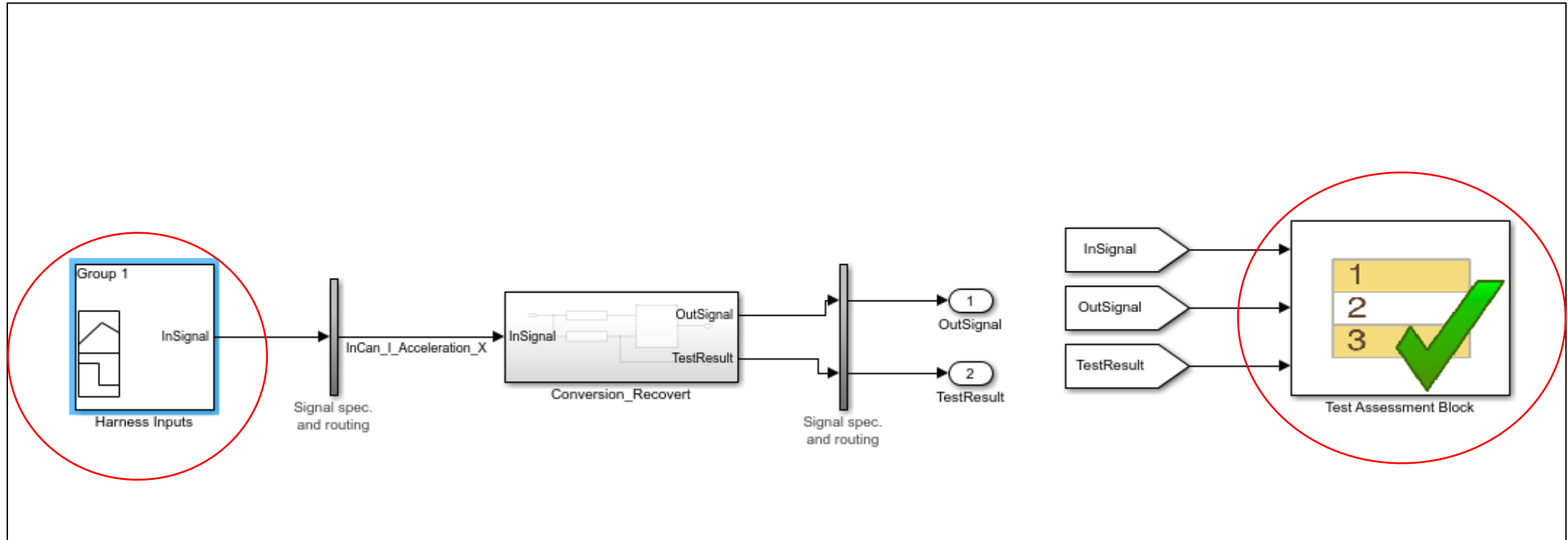


Creation of harness model



Harness model is linked to the SW unit

Harness Model: example



Input test patterns

Assessment block:
expected result evaluation

Simulink Test - Test Manager



Link to requirement under test

The screenshot displays the Simulink Test Manager interface. On the left, the 'Test Browser' pane shows a tree view with 'ax_testfile' expanded to 'ax_test_suite', where 'ax test case 1' is selected. The main area shows the details for 'ax test case 1', which is a 'Baseline Test'. The 'REQUIREMENTS*' section is expanded, and a requirement named 'CAN Inputs - Longitudinal acceleration calculation (caninputs_req_fawsdc#1)' is highlighted in blue. A yellow arrow points from the top of the requirement text down to the highlighted requirement name. Below the requirement list are '+ Add' and 'Delete' buttons. The 'SYSTEM UNDER TEST*' section shows the 'Model' set to 'caninputs'.

Unit Testing status: example



Index	ID	Summary	Implemented	Verified
caninputs_req_fawcdc				
1	#1	CAN Inputs - Longitudinal acceleration calculation	Implemented	Verified
1.1	#215	CAN Inputs - Longitudinal acceleration punctual recovery	Implemented	Verified
2	#4	CAN Inputs - Lateral acceleration calculation	Implemented	Verified
3	#9	CAN Inputs - Acceleration pedal position calculation	Implemented	Verified
4	#17	CAN Inputs - Actual gear calculation	Implemented	Verified
5	#41	CAN Inputs - Brake pedal status calculation	Implemented	Verified
6	#49	CAN Inputs - Combustion torque calculation	Implemented	Verified
7	#54	CAN Inputs - Drive style status calculation	Implemented	Verified
8	#62	CAN Inputs - Engine speed calculation	Implemented	Verified
9	#66	CAN Inputs - Gradient acceleration pedal calculation	Implemented	Verified
10	#70	CAN Inputs - ABS intervention calculation	Implemented	Verified
11	#76	CAN Inputs - ESP intervention calculation	Implemented	Verified
12	#81	CAN Inputs - TCS intervention calculation	Implemented	Verified
13	#86	CAN Inputs - Master cylinder pressure calculation	Implemented	Verified
14	#90	CAN Inputs - Maximum torque calculation	Implemented	Verified
15	#94	CAN Inputs - Operational mode calculation	Implemented	Verified
16	#100	CAN Inputs - Steering wheels angle calculation	Implemented	Verified
17	#104	CAN Inputs - Steering wheel angle sign calculation	Implemented	Verified
18	#109	CAN Inputs - Torque losses calculation	Implemented	Verified
19	#112	CAN Inputs - Front left wheel speed calculation	Implemented	Verified
20	#115	CAN Inputs - Front right wheel speed calculation	Implemented	Verified
21	#118	CAN Inputs - Rear left wheel speed calculation	Implemented	Verified
22	#121	CAN Inputs - Rear right wheel speed calculation	Implemented	Verified
23	#124	CAN Inputs - Yaw rate calculation	Implemented	Verified
24	#173	CAN Inputs - Clutch pedal status calculation	Implemented	Verified
25	#179	CAN Inputs - Driver torque calculation	Implemented	Verified
26	#204	CAN Inputs - WheelSpeed_FL and WheelSpeed_FR invalid	Implemented	Verified
27	#205	CAN Inputs - WheelSpeed_RL and WheelSpeed_RR invalid	Implemented	Verified
28	#206	CAN Inputs - WheelSpeed_FL and WheelSpeed_RR invalid	Implemented	Verified
29	#207	CAN Inputs - WheelSpeed_FL and WheelSpeed_RL invalid	Implemented	Verified
30	#208	CAN Inputs - WheelSpeed_FR and WheelSpeed_RL invalid	Implemented	Verified
31	#209	CAN Inputs - WheelSpeed_FR and WheelSpeed_RR invalid	Implemented	Verified
32	#210	CAN Inputs - WheelSpeed_FR, WheelSpeed_FL and WheelSpeed_RR invalid	Implemented	Verified
33	#211	CAN Inputs - WheelSpeed_FR, WheelSpeed_FL and WheelSpeed_RL invalid	Implemented	Verified
34	#212	CAN Inputs - WheelSpeed_RR, WheelSpeed_RL and WheelSpeed_FL invalid	Implemented	Verified

Simulink Requirements:
overall view

Embedded Coder: code generation



Code Generation Report

Find:

Contents

- [Summary](#)
- [Subsystem Report](#)
- [Code Interface Report](#)
- [Traceability Report](#)
- [Static Code Metrics Report](#)
- [Code Replacements Report](#)

Generated Code

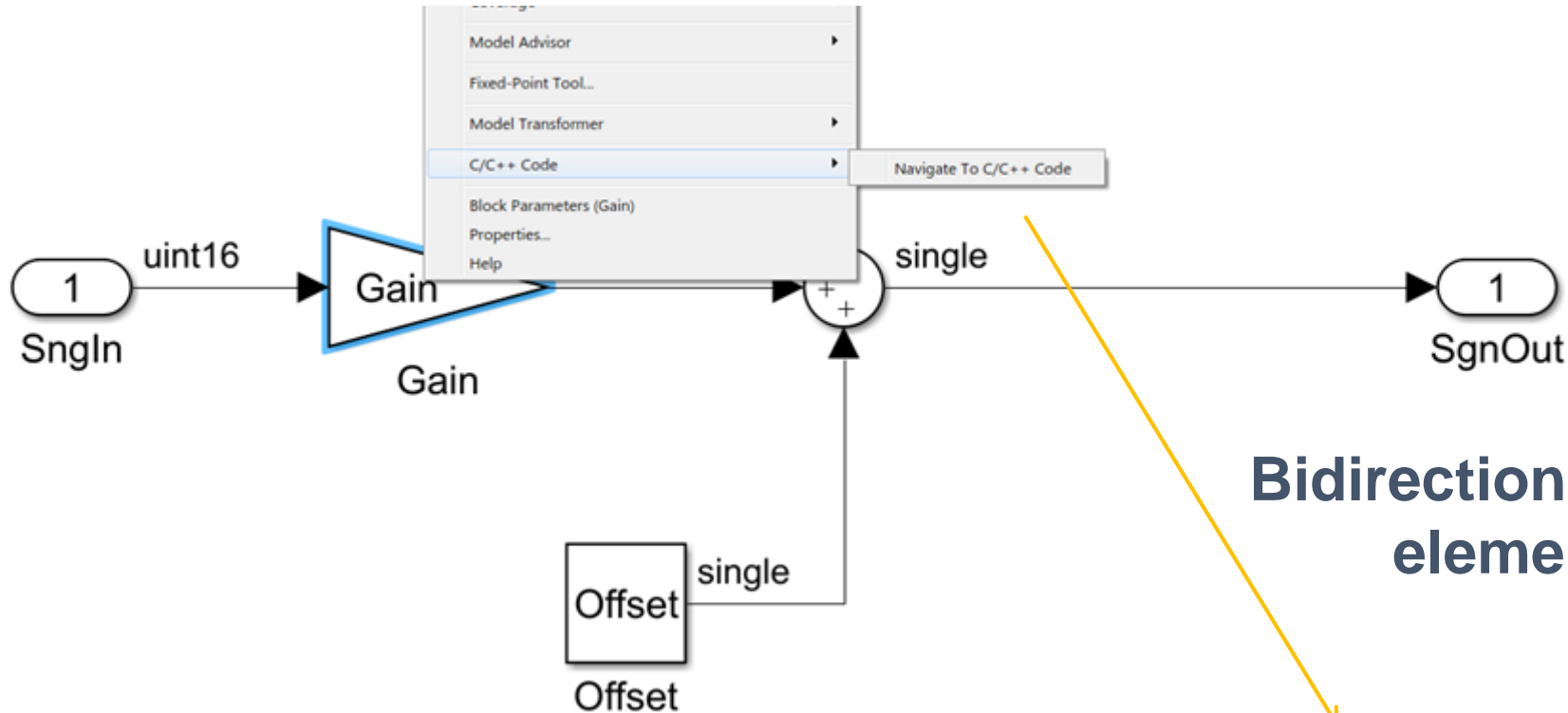
- [-] Model files**
 - [caninputs.c](#)
 - [caninputs.h](#)
 - [caninputs_private.h](#)
 - [caninputs_types.h](#)
- [+] Shared files (1)**
- [-] Interface files**
 - [caninputs.a2i](#)
 - [caninputs.arxml](#)

Function Name	Accumulated Stack Size (bytes)	Self Stack Size (bytes)	Lines of Code	Lines	Complexity
[+] CanInputsRun10CanInputProc	70	70	370	1,009	38
[+] CanInputsRun10CanDiagnosis	14	14	55	145	6
CanInputs_Runnable_Init	0	0	0	4	1

```
/begin MEASUREMENT
/* Name */ InCan_0_AxV
/* Long identifier */ "Vehicle longitudinal acceleration"
/* Data type */ FLOAT32_IEEE
/* Conversion method */ caninputs_CM_single_m_s2
/* Resolution (Not used) */ 0
/* Accuracy (Not used) */ 0
/* Lower Limit */ -3.4E+38
/* Upper Limit */ 3.4E+38
ECU_ADDRESS 0x0000 /* @ECU_Address@InCan_0_AxV@ */
```



Embedded Coder: code generation

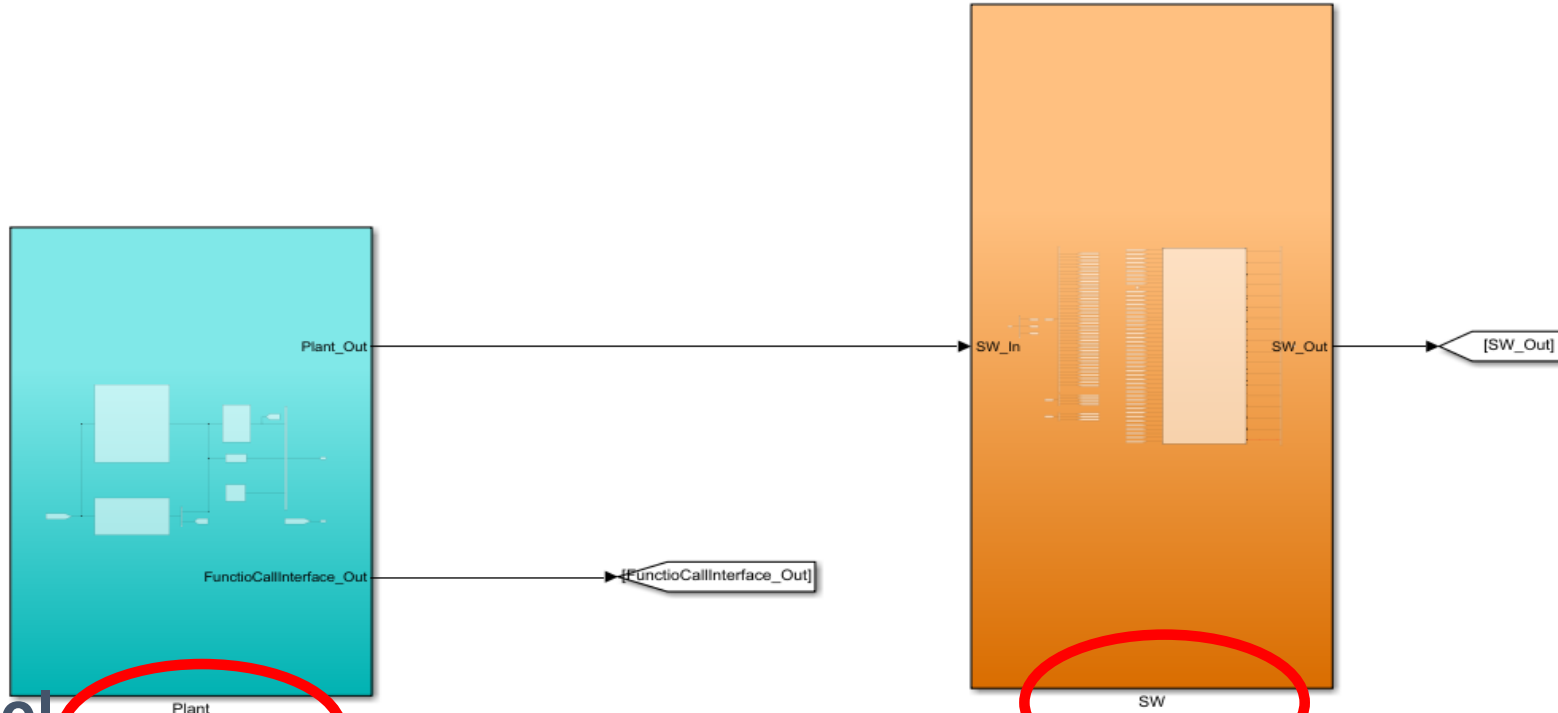


Bidirectional link between SW element and C-code

```
if (Can_rtb_Compare_dw > 0) {  
    InCan_0_AxV = 0.0F;  
} else {  
    InCan_0_AxV = 0.0269999504F * (real32_T)InCan_I_Acceleration_X + -21.593F;  
}
```

MIL testing

Testing of the whole application layer: level 1 requirements



Plant model

SW-Cs composition



MIL testing

- Function callers blocks are used for simulating AUTOSAR S/R and C/S ports
- It is not needed to configure models for MIL
- Same models used for code generation are able to run even in MIL environment

AUTOSAR's interface simulation



MIL testing example: fault injection



Achievements and Outlook



- **ECU SW put in production in April 2019**
- **18 months of development**
- **Technical, organizational and business results.**
 - The standardization of development environment and the “bottom up” approach has increased the cross-competence inside the SW team
 - No need of other tools for AUTOSAR architecture design as regards to application SWCs
 - One single data base for requirements, software models, code and testing results
 - Cutting of time needed for documentation since it is automatically generated

Achievements and Outlook



- Integrated toolchain based on Simulink environment for SW development made traceability easier to achieve
- Use of the tool's standard features only, avoiding customization (scripts) made the toolchain lean and easier to update
- Bottom – up approach made AUTOSAR SW components design quicker

Forward-looking plans



- Use of new and upcoming MathWorks tools as System Composer for
 - System design in accordance with A-SPICE requirements

THANK YOU
FOR YOUR
ATTENTION