

# Mastering Software-Defined Vehicle Complexity: Systems Engineering and Model-Based Design for Distributed Architectures



**KPIT**

## Our presenters today



Tanmay Agrawal

KPIT

Principal Architect  
Powertrain, SDV & Systems Engineering



Domenico Ferrari

MathWorks

Senior Application Engineer

# You ALL have experienced this...

10:03 – “A new customer requirement just materialized!”

10:05 – “It’s too late! Feature freeze is next week”

10:08 – “This is high priority! Let’s do an impact analysis”

10:15 – “...”

16:07 – “We can’t trace the requirement change to the software...”

16:08 – “Let’s create a TASK FORCE ASAP!”

...and with SDV is even more challenging...

10:03 – “A new customer requirement just materialized!”

10:05 – “Ok, we need to update a service interface in the HPC”

10:08 – “Hold on — the change impacts the gateway logic. We need a new signal/service mapping”

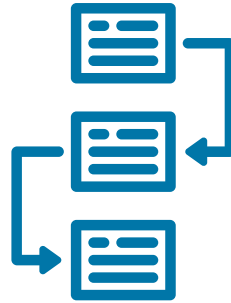
10:15 – “What is the impact on sensor/actuator ECU?”

16:08 – “This is an SDV: we are expected to bring this new feature to market FAST!”

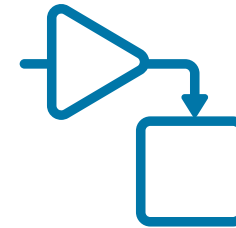
You can master SDV complexity by



Adopting a structured process



Establishing traceability



Using models throughout  
the design cycle

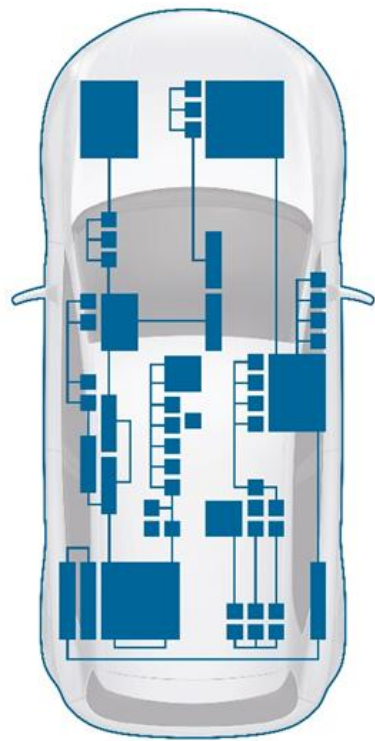
We'll walk you through a concrete example



# Agenda

- SDV Complexity and Key Development Challenges
- Mastering Complexity Through Model-Based System Engineering and Model-Based Design
- Live Demonstration of a Practical Use Case
- Conclusions and Key Take-Aways

# SDV Complexity and Key Development Challenges



Central vehicle  
performance CPUs and GPUs,  
coupled with zone controllers



## Key Challenges

Architectural Shift

Increased Software Size

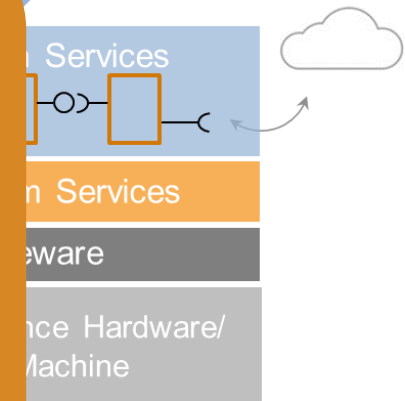
Ensuring Safety Security

Quality



SW updates

- Frequent
- Selective
- Over-the-air



oriented architecture  
Decoupling SW & HW

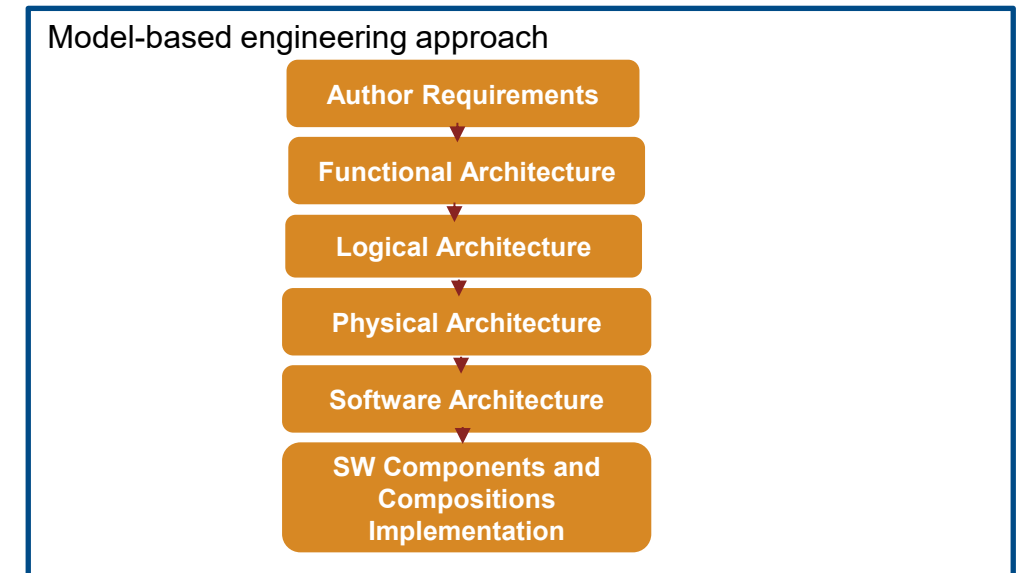
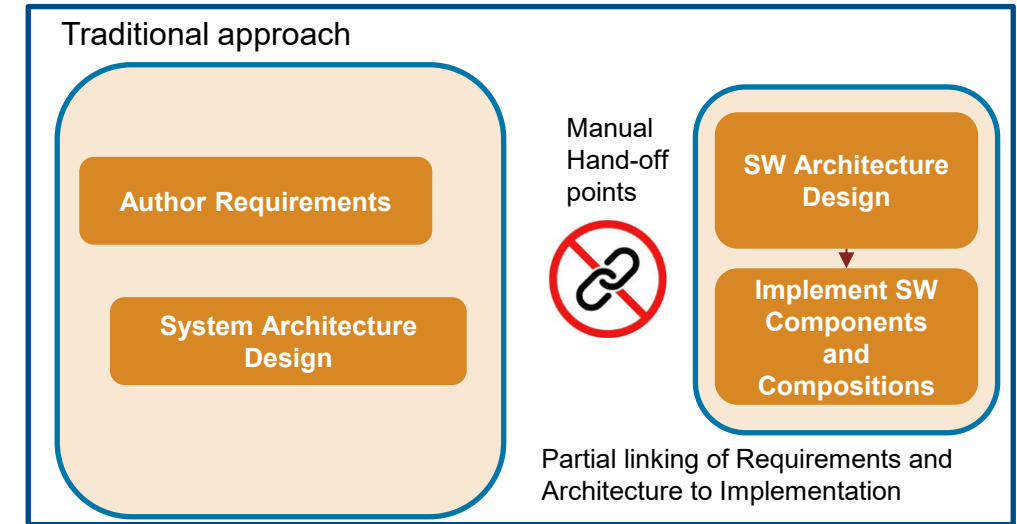
Enable applications to update over the air

# Approaches to handle SDV Complexity and Development Challenges

- Traditional - **disconnected system and software engineering** with limited traceability

**Inadequate for SDV complexity!**

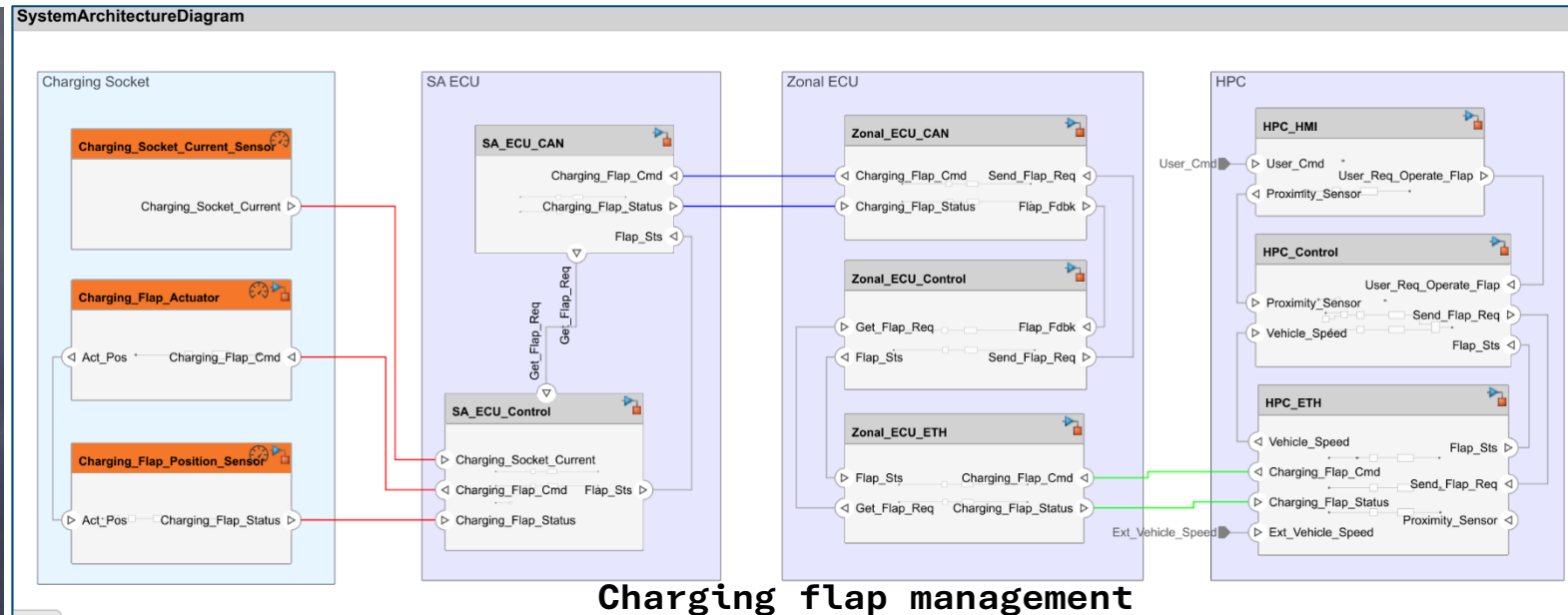
- Model-based engineering approach - unifying system and software development through **executable models, enabling full traceability and early validation**





# Practical Use Case: Flap management of Electrified SDV

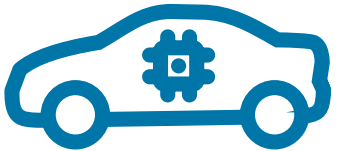
- Flap management
  - it manages the charging port to ensure reliable protection while ensuring secure and automated charging operations
- Flap management feature is distributed across
  - HPC (Central compute)
  - Zonal ECU (Gateway)
  - Nodal ECU (Sensors/Actuators)



## Practical Use Case: Implement a New Requirement in the Flap Management Feature



**New requirement:** “if open, the flap must be closed if the vehicle speed exceeds a given threshold”



**SDV environment:**

- Distributed architecture
- Fast development cycle
- Validation with limited/no hardware availability

## Why This is Hard (without a systematic and consistent approach)



**No single source of truth:** requirements for the feature are **scattered across documents** (often outdated, unclear, or incomplete)



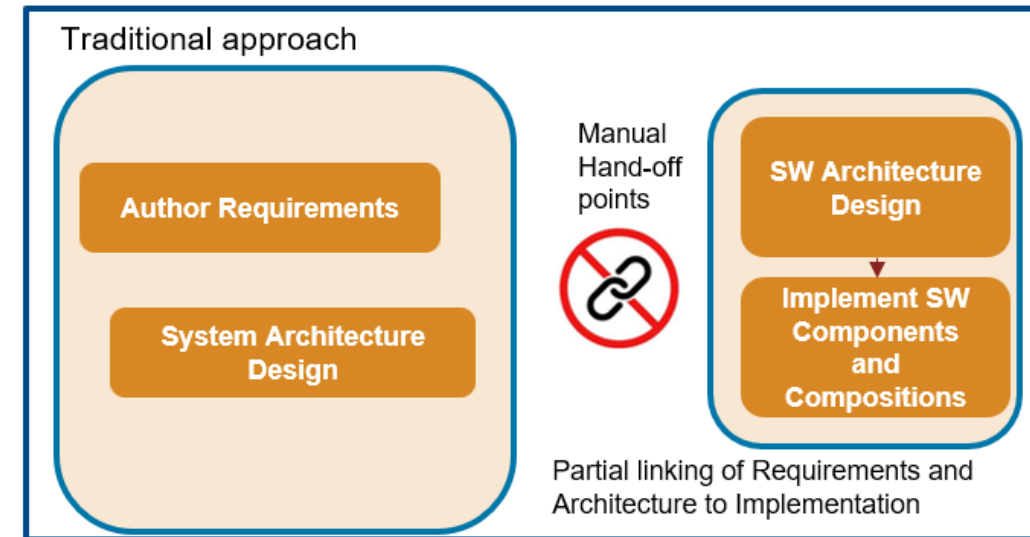
**Feature behavior** is not captured in a formal, model-based form: just text, old diagrams or code



**No end-to-end traceability** from requirements to architecture to implementation, so identifying *where* to insert new logic becomes a manual, error-prone investigation

## Why This is Hard (without a systematic and consistent approach)

- Without a behavioral model, **teams can't predict side effects** on related features like charging, HMI, or supervisory logic
- HPC, zonal, and nodal **software teams work in silos**, causing potential inconsistencies and integration issues



Adding “one small feature” can trigger extensive refactoring, regressions and integration risks

# Solution: System Thinking and Feature-driven Engineering

A Unified Model-Based System Engineering + Model-Based Design Workflow

## Systems Thinking Workflow

**Add/update System requirement** for *"flap auto close beyond certain vehicle speed"*



**Use system models for impact analysis** of change in value stream



**Update Logical & Physical (System) Architecture** based on impact analysis



**Model based System to Software hand off** for consistent change propagation in Software



**Simulate the system behavior** for early validation of the change



**Autogenerate C/C++ code** from the model-based design

## Benefits it brings

Clear, Complete & traceable requirements

Quick & consistent incorporation of change at System & software level

Logical subsystems & Physical ECUs stay in sync with functional change

Software design is fully consistent & traceable with system design

Avoid later integration challenges in SDV by early defect identification

Consistent software code with Software design adhering to standards

Let's see the approach in action on a practical use case



# Impact Analysis of the New Requirement

Identify the system-level requirement that needs to be updated



**New requirement:** “if open, the flap must be closed if the vehicle speed exceeds a given threshold”

The screenshot shows the Requirements Editor interface. The main table lists requirements with columns for Index, ID, Summary, Implemented, and Verified. Requirement 7, 'Execute Charging Flap Closing', is highlighted. The Properties panel on the right shows details for this requirement, including its Type (Functional), Index (7), Custom ID (7), and Summary (Execute Charging Flap Closing). The Description tab is active, showing the text: 'When the user removes the charging gun and the measured proximity resistance is equal to zero, the system shall close the charging flap.'

Index	ID	Summary	Implemented	Verified
▼ SysReq_Update				
▼ Import1				
1	1	Validate Charging Flap Opening		
2	2	Verify Flap Position		
3	3	Execute Charging Flap Opening		
4	4	Notify Closed Flap Status		
5	5	Notify Open Flap Status		
6	6	Notify Charging Socket Error		
7	7	Execute Charging Flap Closing		
> SW_Reqs				

**Properties**

Type: Functional

Index: 7

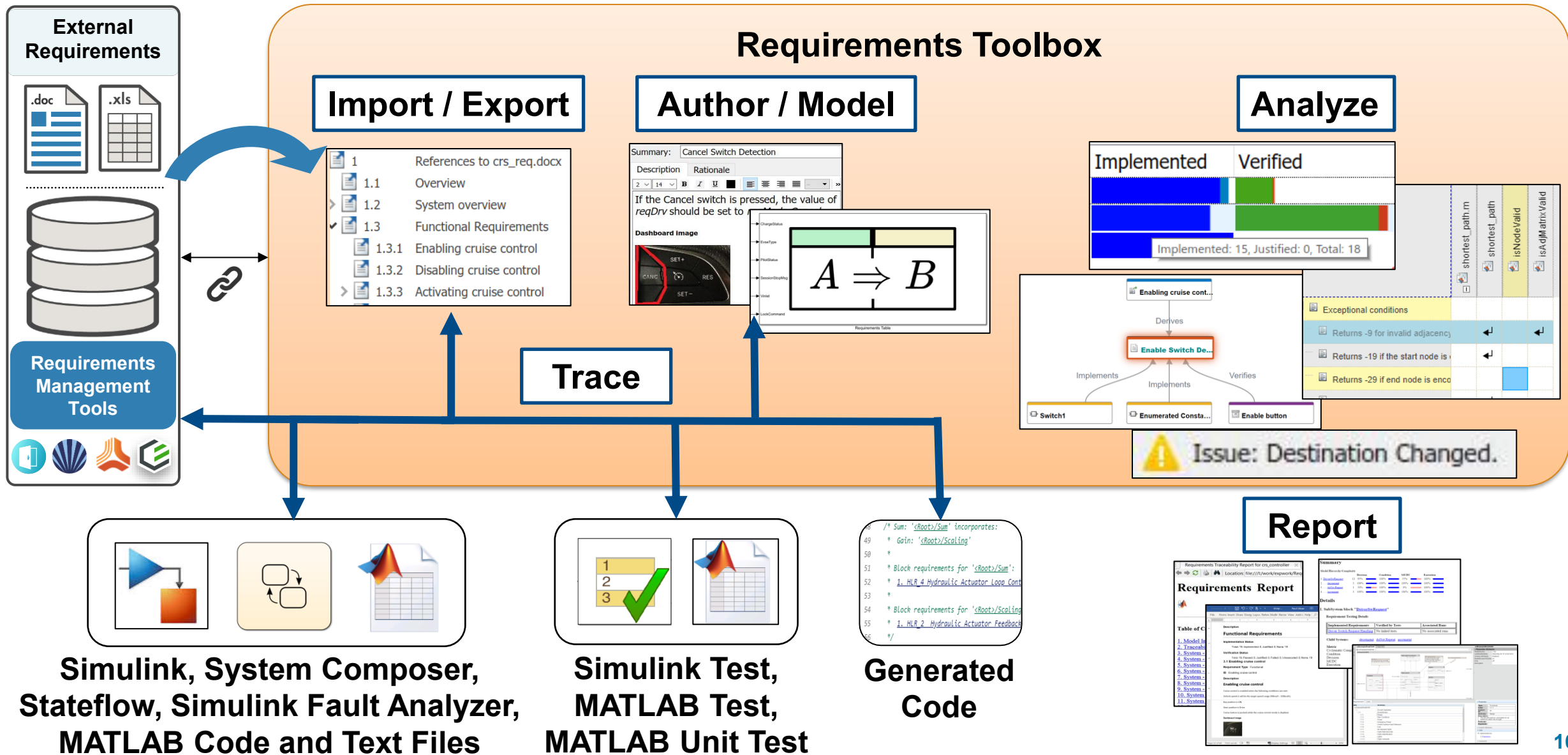
Custom ID: 7

Summary: Execute Charging Flap Closing

Description Rationale

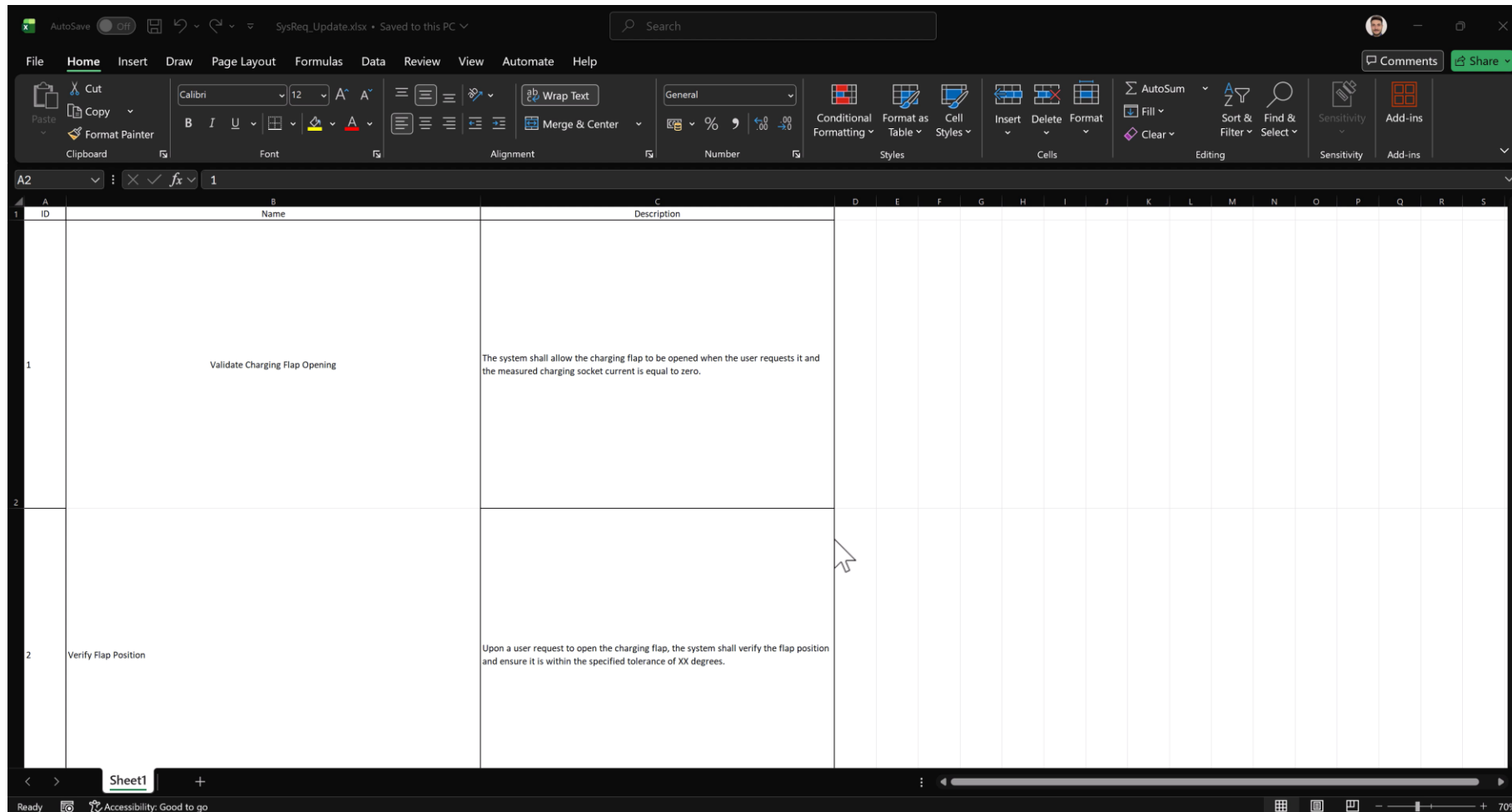
When the user removes the charging gun and the measured proximity resistance is equal to zero, the system shall close the charging flap.

# Full Requirements Traceability with System and Software



# Impact Analysis of the New Requirement

Import the change in Requirements Toolbox to start the impact analysis



The screenshot shows the Microsoft Excel interface with the following data in the 'Requirements' table:

ID	Name	Description
1	Validate Charging Flap Opening	The system shall allow the charging flap to be opened when the user requests it and the measured charging socket current is equal to zero.
2	Verify Flap Position	Upon a user request to open the charging flap, the system shall verify the flap position and ensure it is within the specified tolerance of XX degrees.

# Impact Analysis of the New Requirement

Requirements Editor

REQUIREMENTS

FILE PROFILE REQUIREMENTS LINKS VIEW EDIT ANALYSIS SHARE

New Requirement Set Open Save Import Close Load Profile Editor Add Requirement Add Link Show Requirements Show Links Search Traceability Matrix Traceability Diagram Model Testing Dashboard Export

Index	ID	Summary	Implemented	Verified
▼ SysReq_Update				
▼ Import1				
1	1	Validate Charging Flap Opening		
2	2	Verify Flap Position		
3	3	Execute Charging Flap Opening		
4	4	Notify Closed Flap Status		
5	5	Notify Open Flap Status		
6	6	Notify Charging Socket Error		
7	7	Execute Charging Flap Closing		
> SW_Reqs				

When the user removes the charging gun and the measured proximity resistance is equal to zero or vehicle speed is over a threshold, the system shall close the charging flap.

Keywords:

Revision information:

Show in document Unlock

Links

Derives:

- SASW-12 Flap Close Request
- SASW-14 Flap Close

Implemented by:

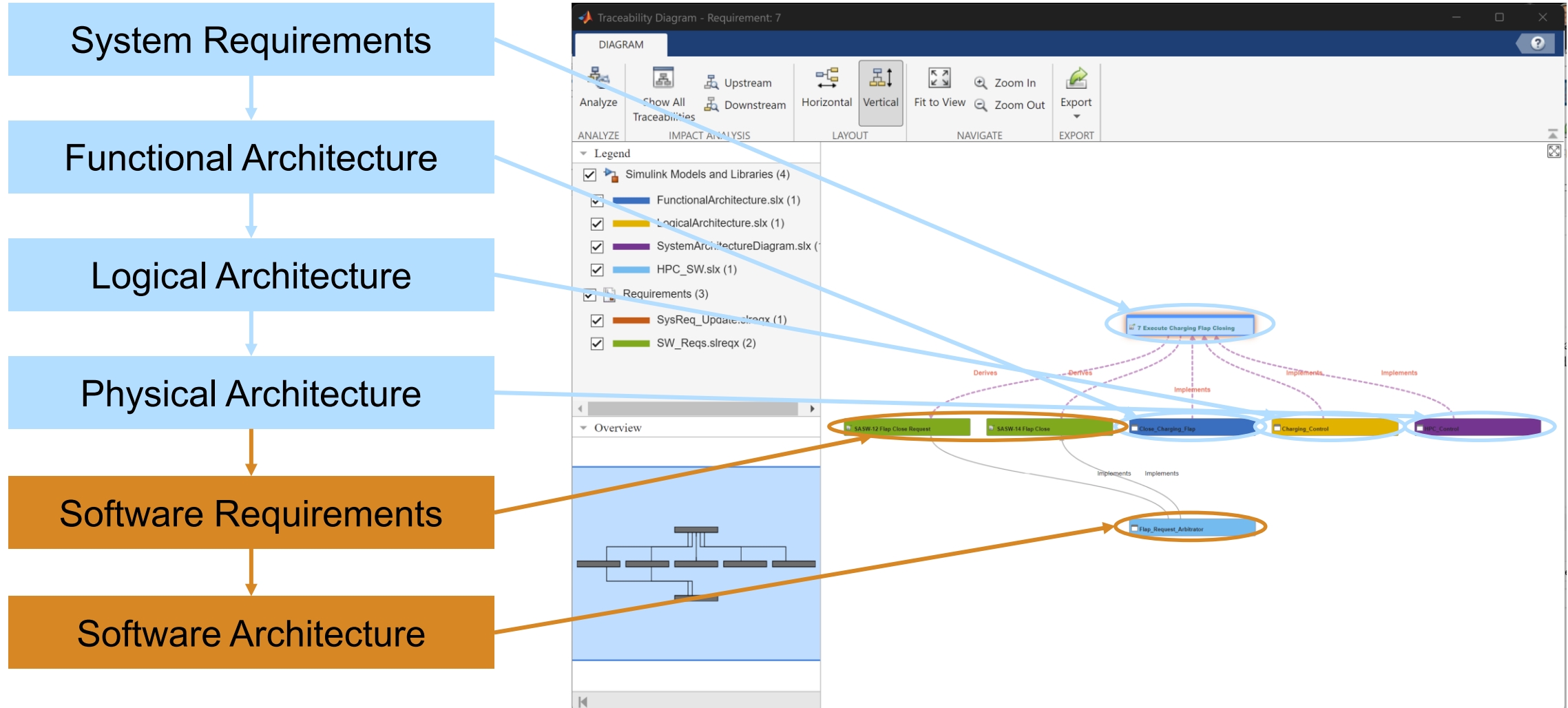
- Close\_Charging\_Flap
- Charging\_Control
- HPC\_Control

Comments

The change of the requirement is propagated on different system and software artifacts that we need to modify.

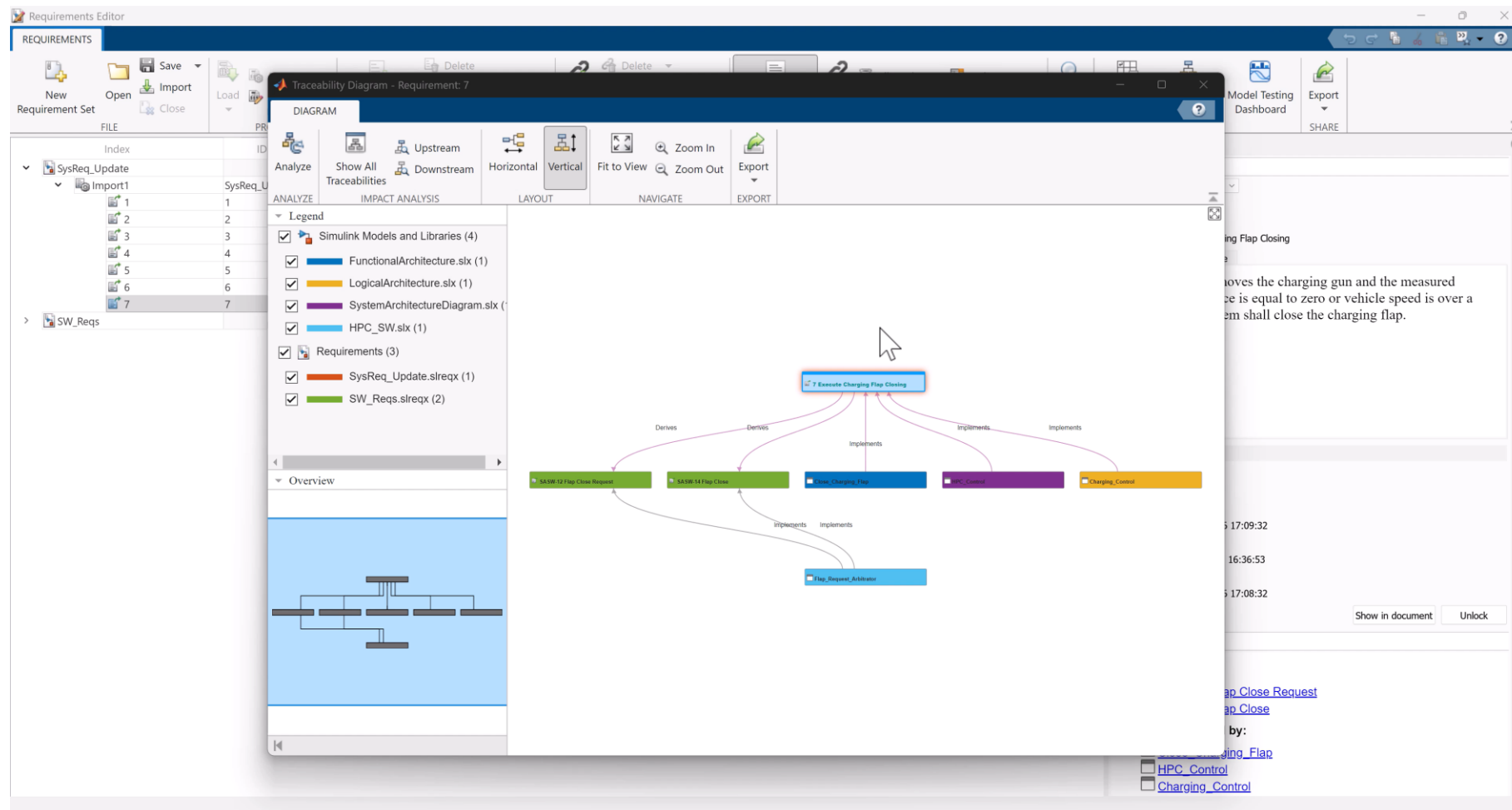
# Impact Analysis of the New Requirement

The Traceability Diagram highlights the impacts at different levels



# Change on Functional Architecture

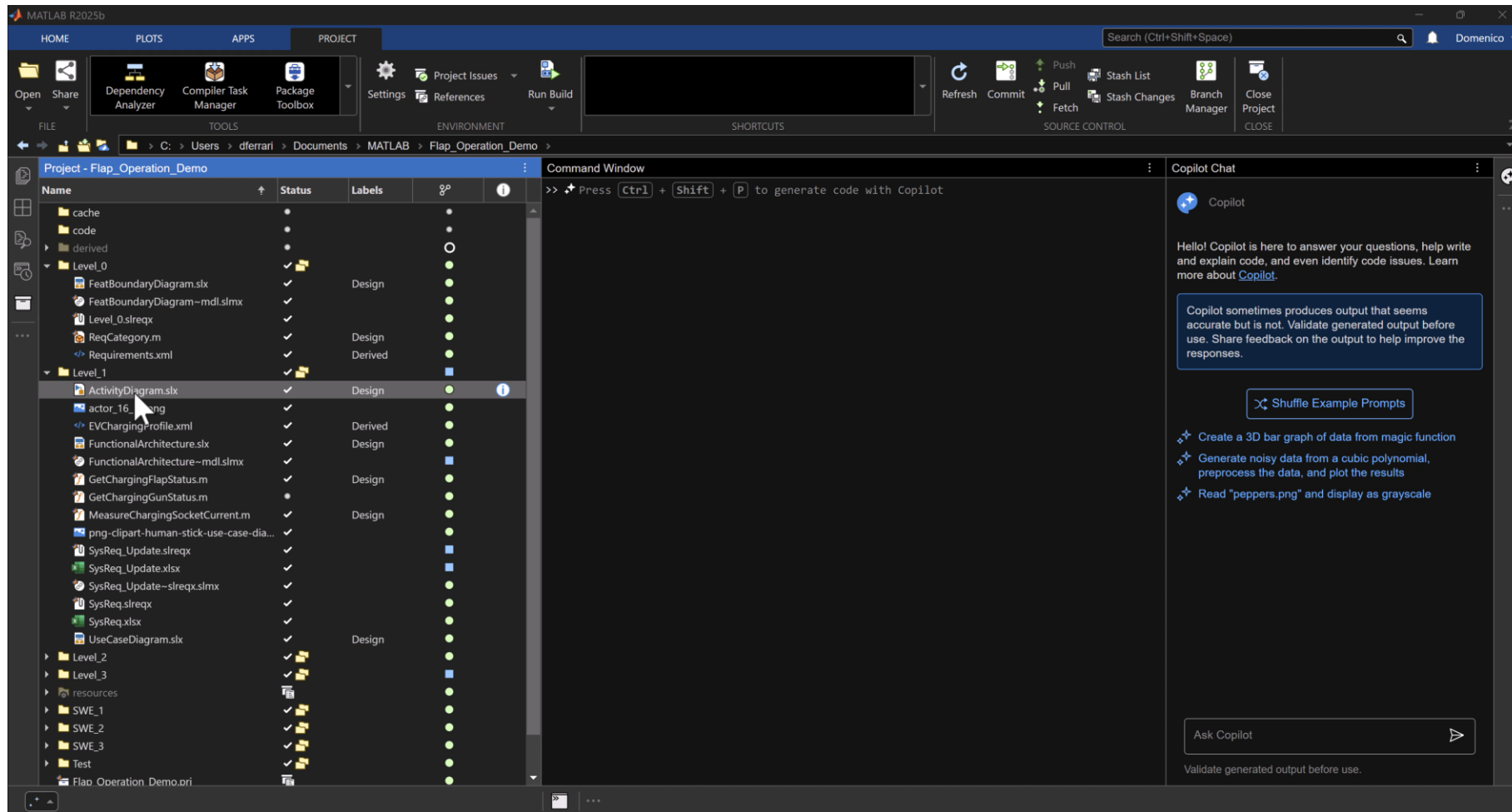
Perform change in the function impacted by the new requirement



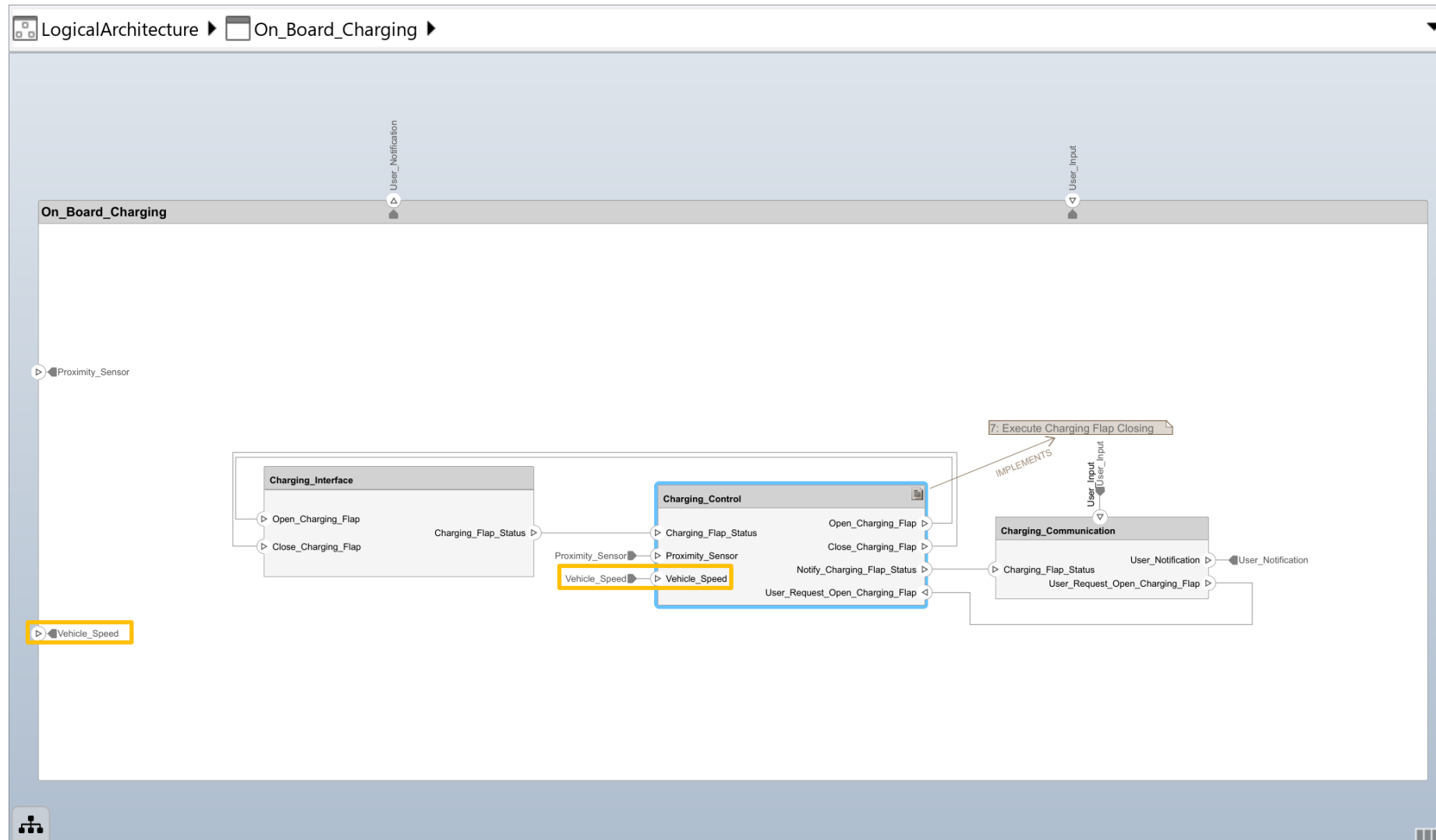


# Change on Functional Architecture

## Modify the Activity Diagram to validate the new function behavior



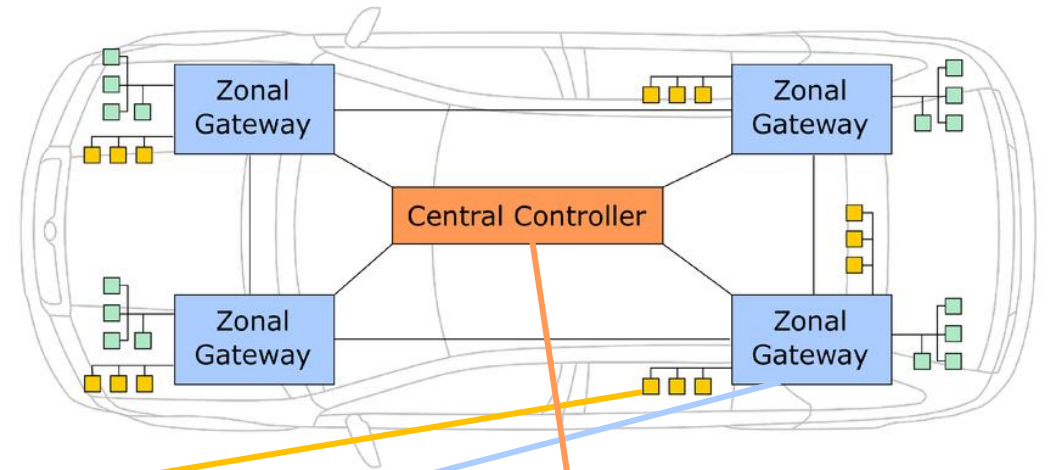
# Change on Logical Architecture



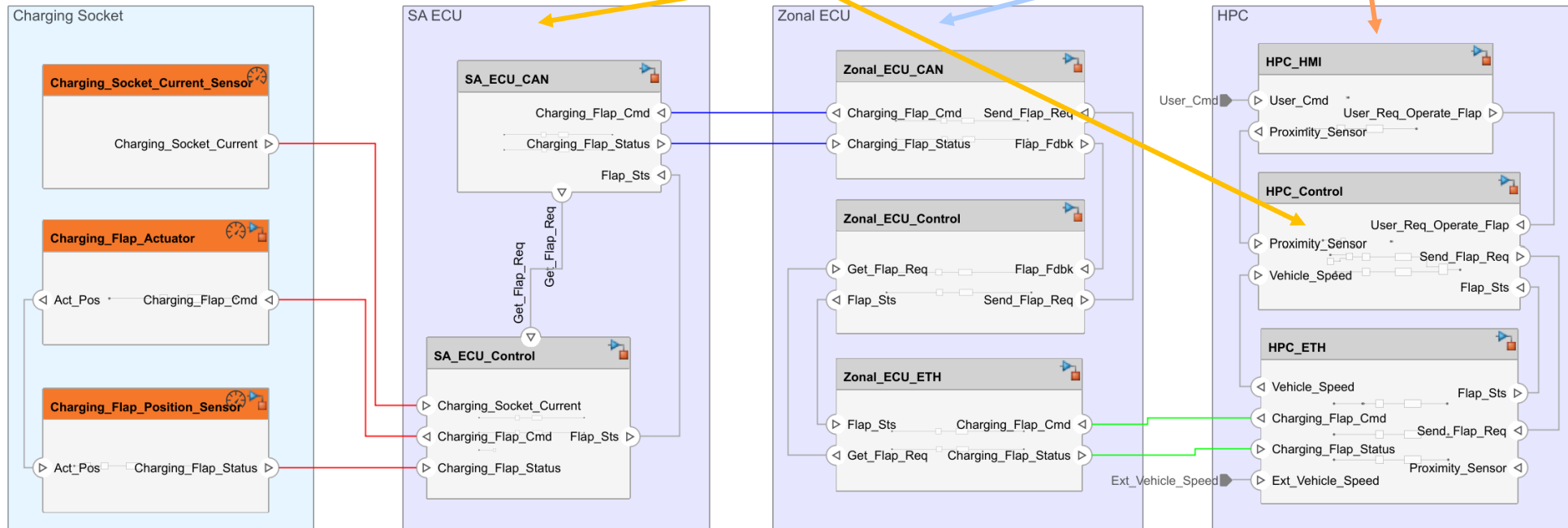
# Change on Physical Architecture

## Allocation Editor

Scenario 1	SystemArchitectureDiagram	Charging_Flap_Act	Charging_Flap_Pos	Charging_Socket_L	HPC_Control	HPC_ETH	HPC_HMI	SA_ECU_CAN	SA_ECU_Control	Zonal_ECU_CAN	Zonal_ECU_Control	Zonal_ECU_ETH
FunctionalArchitecture												
Charging_Flap_Operation												
Close_Charging_Flap												
Get_Charging_Flap_Status												
Measure_Charging_Socket												
Open_Charging_Flap												

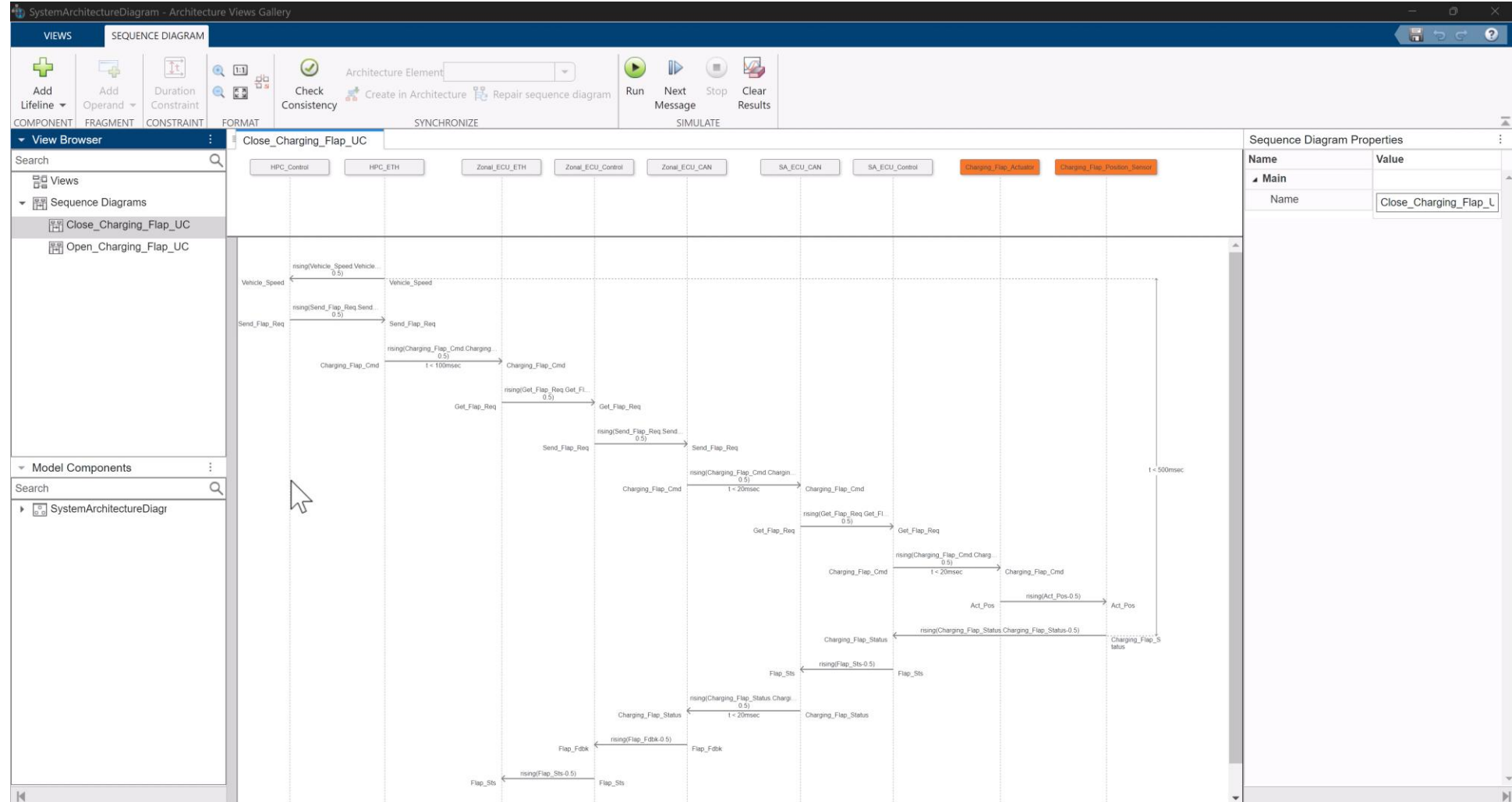


## SystemArchitectureDiagram



# Change on Physical Architecture

## Analyze the System with the Sequence Diagram



# Change on Physical Architecture

## Analyze the System with the Sequence Diagram



**New requirement:** “if open, the flap must be closed if the vehicle speed exceeds a given threshold. Closing shall take no longer than 500ms.”

# Change on Software Requirements

The screenshot shows the Requirements Editor window with the 'REQUIREMENTS' tab active. The interface includes a toolbar with icons for New Requirement Set, Open, Save, Import, Close, Load, Profile Editor, Add Requirement, Add Link, Show Requirements, Show Links, Search, Traceability Matrix, Traceability Diagram, Model Testing Dashboard, and Export. Below the toolbar is a table of requirements. Requirement SASW-12, 'Flap Close Request', is selected, and its details are shown in the Properties panel on the right. The Properties panel includes fields for Type (Functional), Index (34), Custom ID (SASW-12), and Summary (Flap Close Request). The Description tab is active, showing the text: 'When SW receives a command to close the charging flap, the SW shall read the proximity resistance value and the vehicle speed.'

Index	ID	Summary	Implemented	Verified
28	SASW-6	Flap Status		
29	SASW-7	Communication Requirement		
30	SASW-8	Flap Actuator		
31	SASW-9	Current Sensor		
32	SASW-10	Position Sensor		
33	SASW-11	Flap Close Response		
34	SASW-12	Flap Close Request		
35	SASW-14	Flap Close		
36	SASW-15	Flap Status		
37	SASW-16	Charging Flap Position		
38	SASW-17	Operate Charging Flap		
39	SASW-18	Socket Current		
40	SASW-19	Flap Action		

**Properties**

Type: Functional  
Index: 34  
Custom ID: SASW-12  
Summary: Flap Close Request

Description Rationale

When SW receives a command to close the charging flap, the SW shall read the proximity resistance value and the vehicle speed.

The screenshot shows the Requirements Editor window with the 'REQUIREMENTS' tab active. The interface is similar to the previous one, but requirement SASW-14, 'Flap Close', is selected. The Properties panel on the right shows details for SASW-14, including Type (Functional), Index (35), Custom ID (SASW-14), and Summary (Flap Close). The Description tab is active, showing the text: 'When SW reads the proximity resistance value, if the proximity resistance value is less than 5 ohms, or the vehicle speed is above a calibratable threshold, SW shall command the flap actuator to close the flap.'

Index	ID	Summary	Implemented	Verified
27	SASW-5	Flap Open Status		
28	SASW-6	Flap Status		
29	SASW-7	Communication Require...		
30	SASW-8	Flap Actuator		
31	SASW-9	Current Sensor		
32	SASW-10	Position Sensor		
33	SASW-11	Flap Close Response		
34	SASW-12	Flap Close Request		
35	SASW-14	Flap Close		
36	SASW-15	Flap Status		
37	SASW-16	Charging Flap Position		
38	SASW-17	Operate Charging Flap		
39	SASW-18	Socket Current		
40	SASW-19	Flap Action		

**Properties**

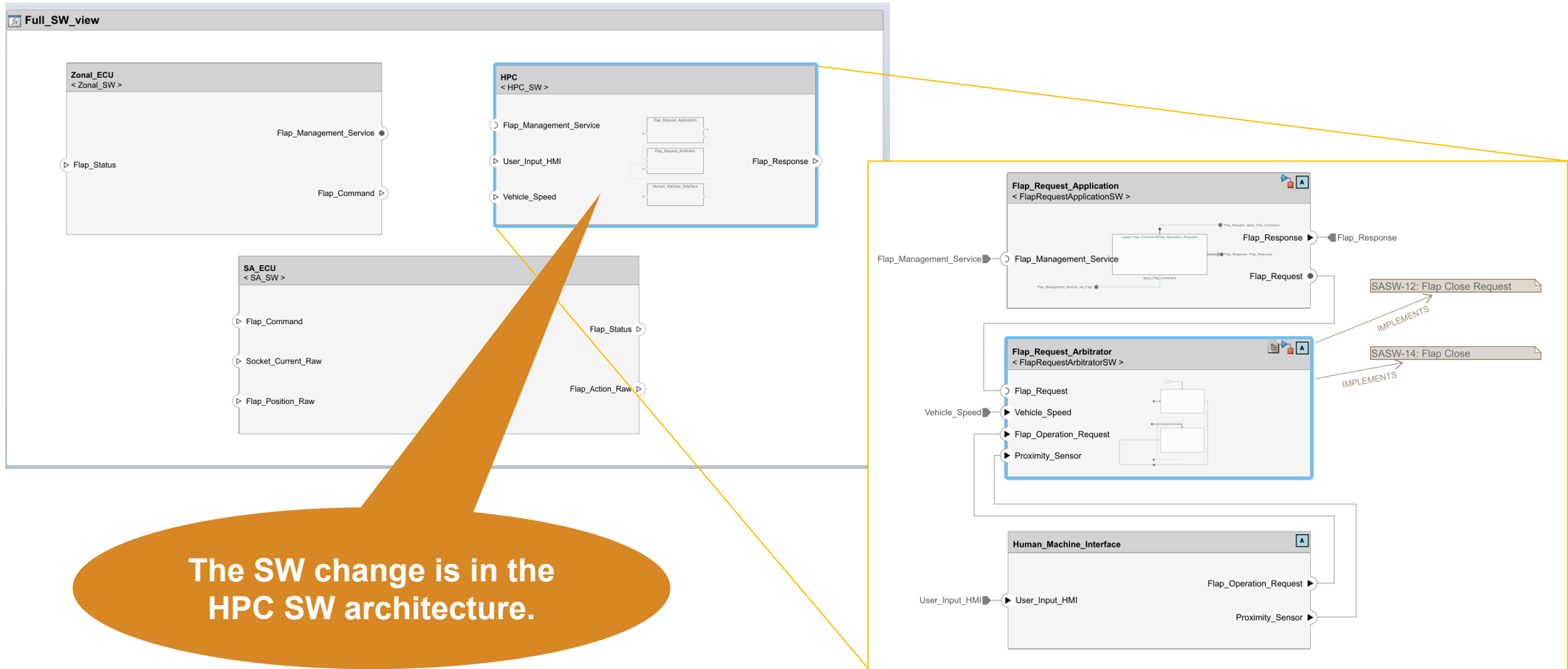
Type: Functional  
Index: 35  
Custom ID: SASW-14  
Summary: Flap Close

Description Rationale

When SW reads the proximity resistance value, if the proximity resistance value is less than 5 ohms, or the vehicle speed is above a calibratable threshold, SW shall command the flap actuator to close the flap

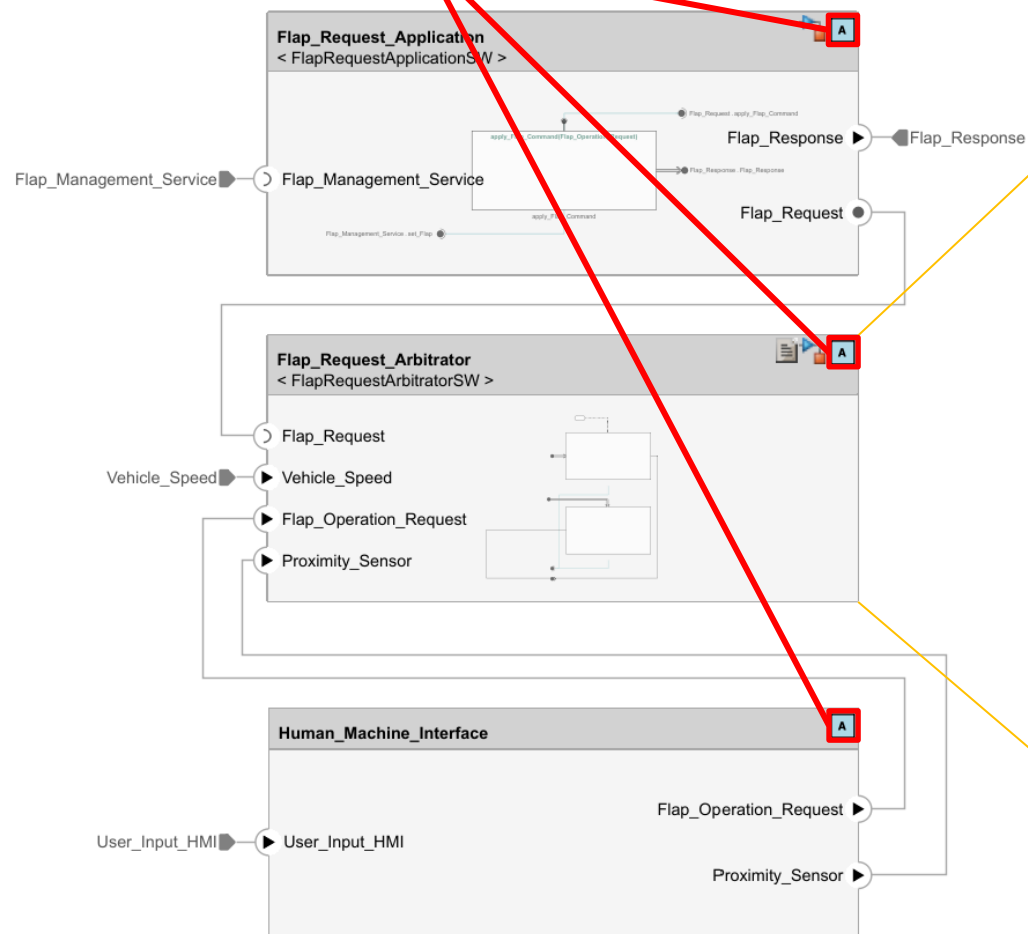


# Change on Software Architecture

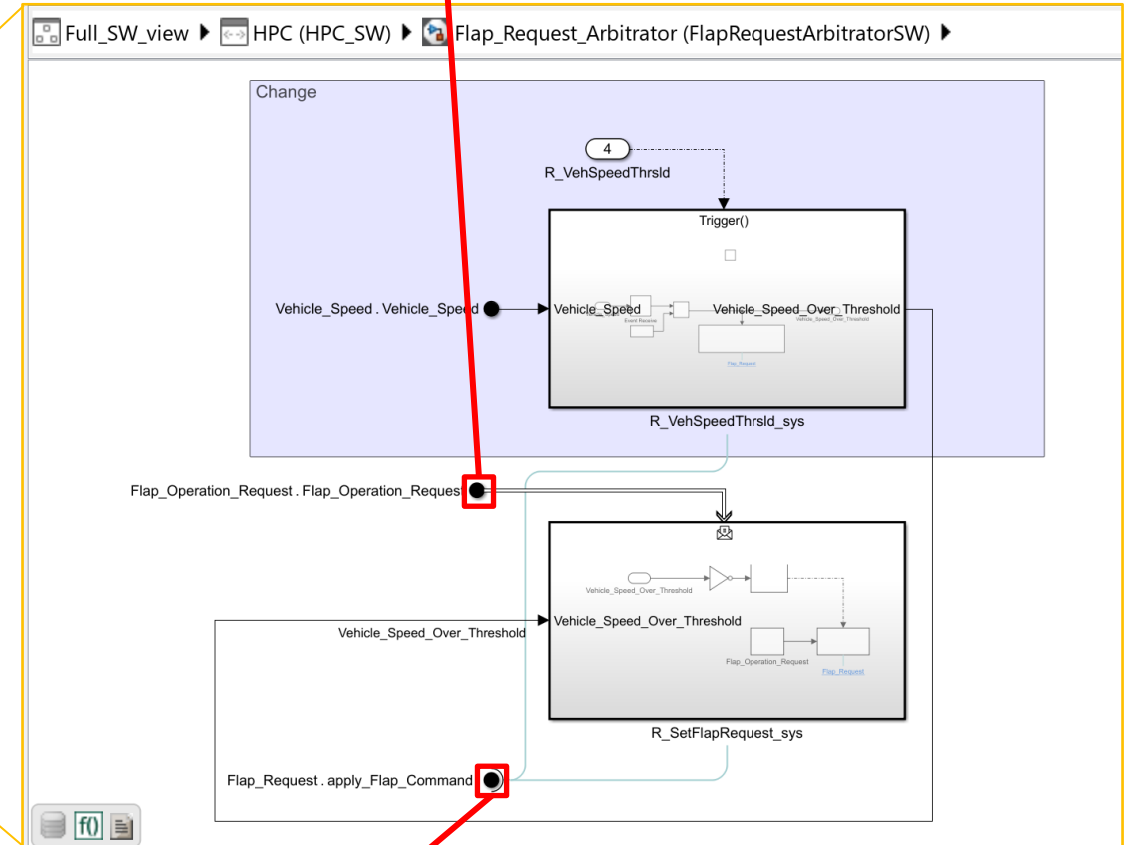


# Change on Software Component

## AUTOSAR Adaptive



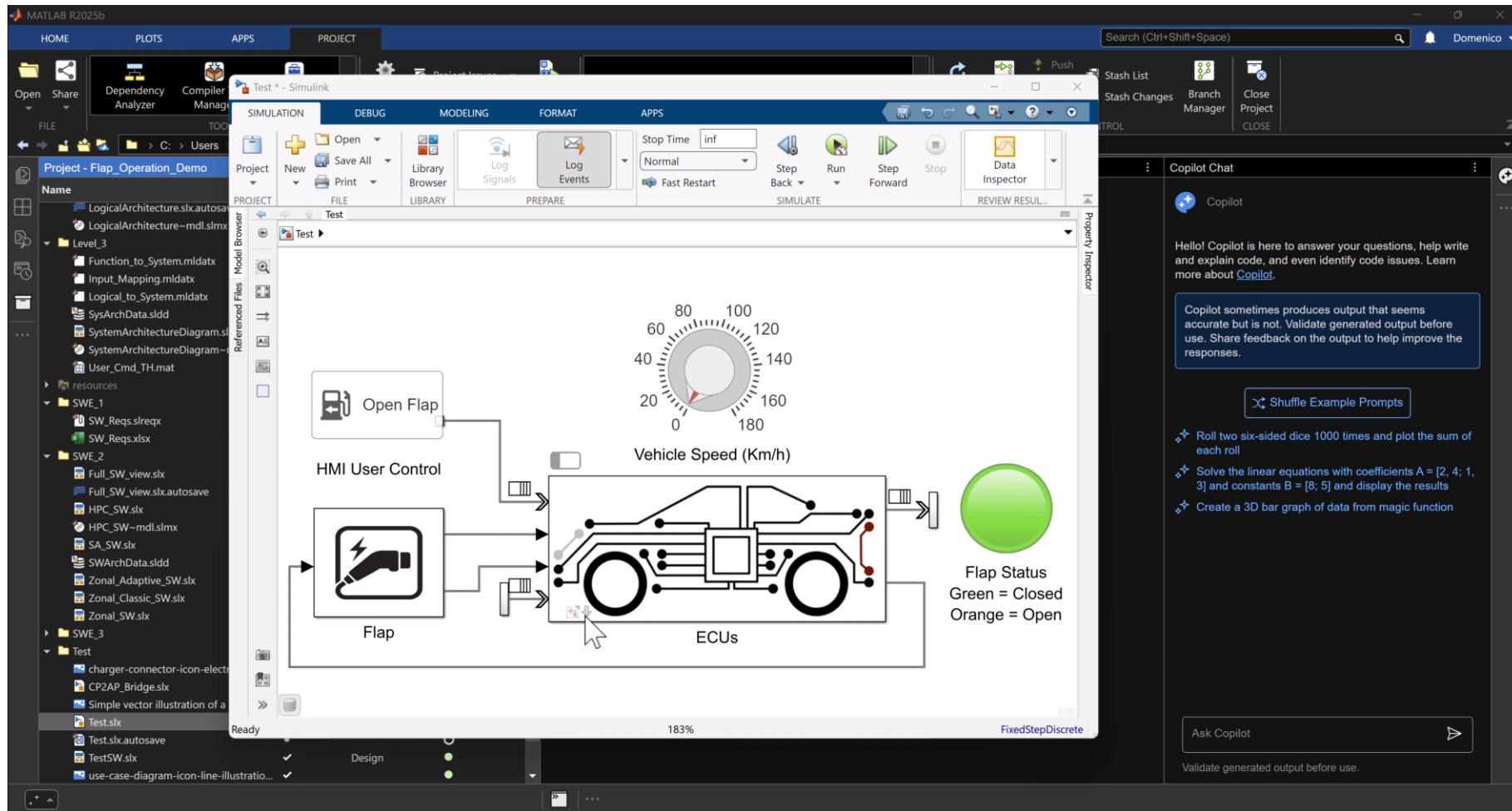
## Events based functions



## Methods

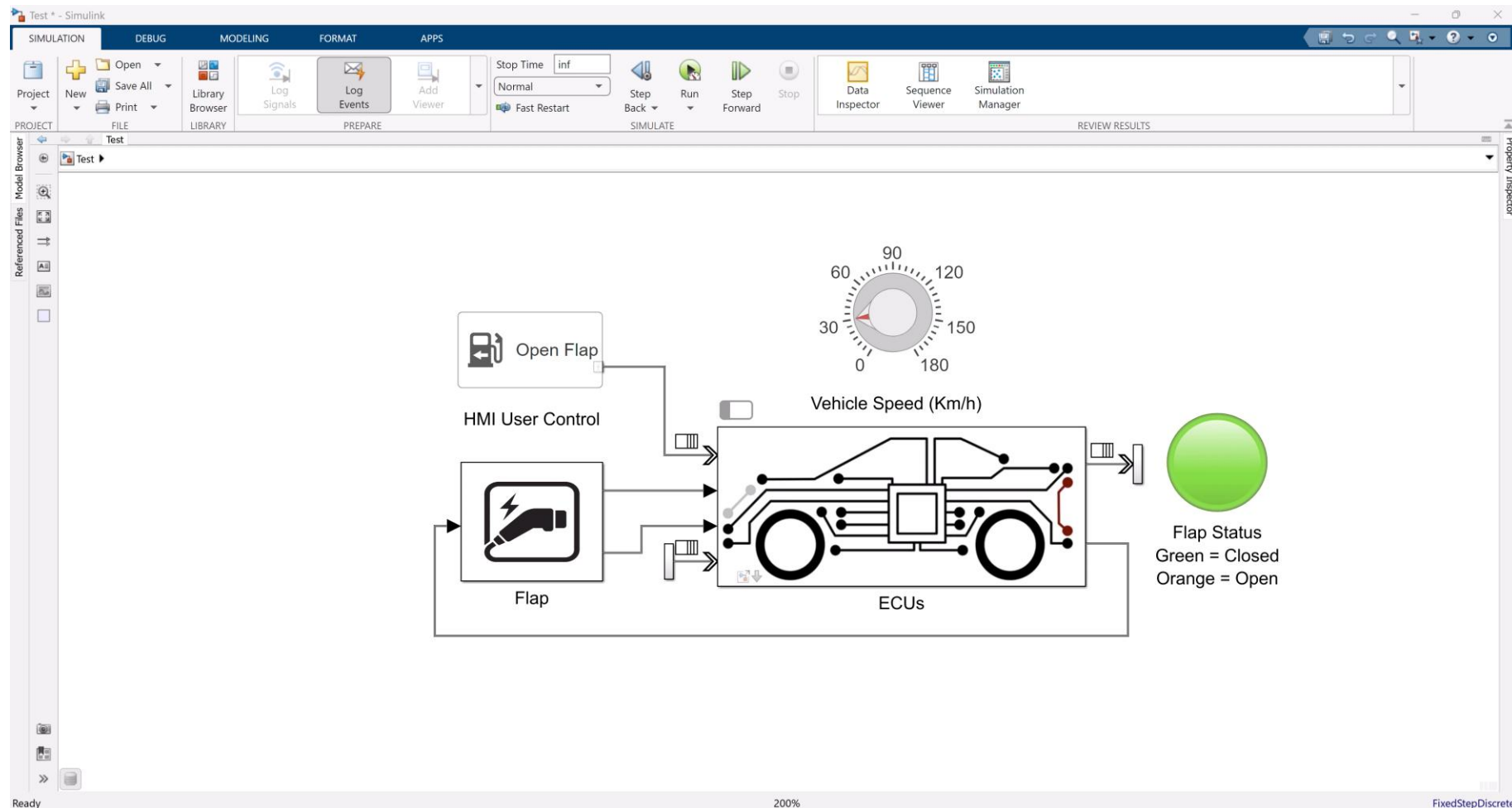
## Perform full Software Architectures closed-loop simulation

## Perform full Software Architectures closed-loop simulation



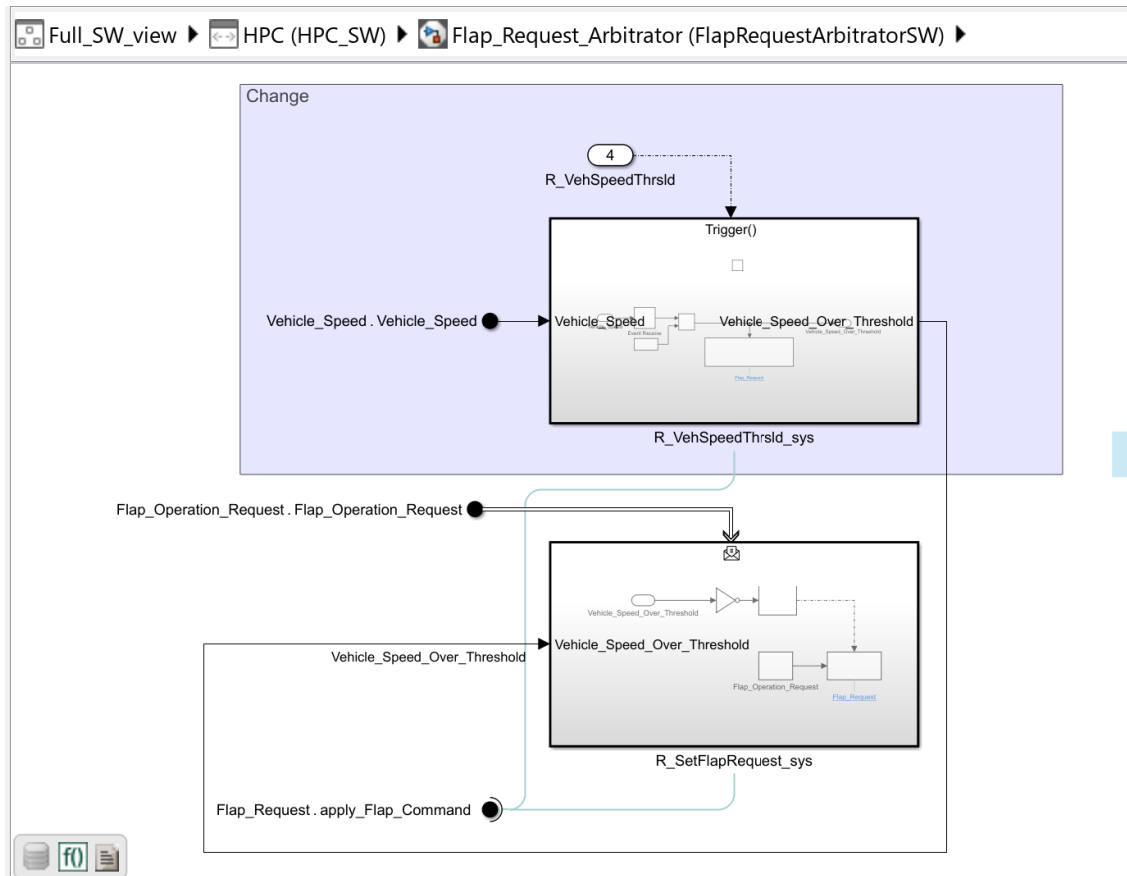
# Simulate the change

Perform full Software Architectures closed-loop simulation



# Automatic Code Generation

Generate the code of the feature, ready for deployment



Generate  
Code

```

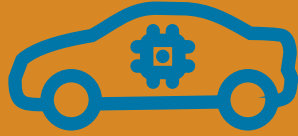
Code
FlapRequestArbitratorSW.cpp
1 //
2 // FlapRequestArbitratorSW.cpp
3 //
4 // Code generation for model "FlapRequestArbitratorSW".
5 //
6 // Model version : 5.29
7 // Simulink Coder version : 25.2 (R2025b) 28-Jul-2025
8 // C++ source code generated on : Wed Feb 25 14:55:09 2026
9 //
10 // Target selection: autosar_adaptive.tlc
11 // Embedded hardware selection: Intel->x86-64 (Windows64)
12 // Code generation objectives:
13 // 1. Execution efficiency
14 // 2. RAM efficiency
15 // Validation result: Not run
16
17
18 #include "FlapRequestArbitratorSW.h"
19 #include <stdint.h>
20 #include "zero_crossing_types.h"
21
22 void FlapRequestArbitratorSW::Vehicle_SpeedVehicle_SpeedReceive(ara::com::
23 SamplePtr< proxy::events::Vehicle_Speed::SampleType const > elementPtr)
24 {
25 // Receive: '<S2>/Event Receive'
26 rtDW.EventReceive = *elementPtr;
27 }
28
29 void FlapRequestArbitratorSW::
30 Flap_Operation_RequestFlap_Operation_RequestReceive(ara::com::SamplePtr< proxy::
31 events::Flap_Operation_Request::SampleType const > elementPtr)
32 {
33 // Outputs for Function Call SubSystem: '<Root>/R_SetFlapRequest_sys' incorporates:
34 // TriggerPort: '<S1>/Flap_Operation_Request'
35
36 rtDW.Flap_Operation_Request_h = *elementPtr;

```

# Conclusions and Key Take-Aways

## Key Challenges

ArchitecturalShift

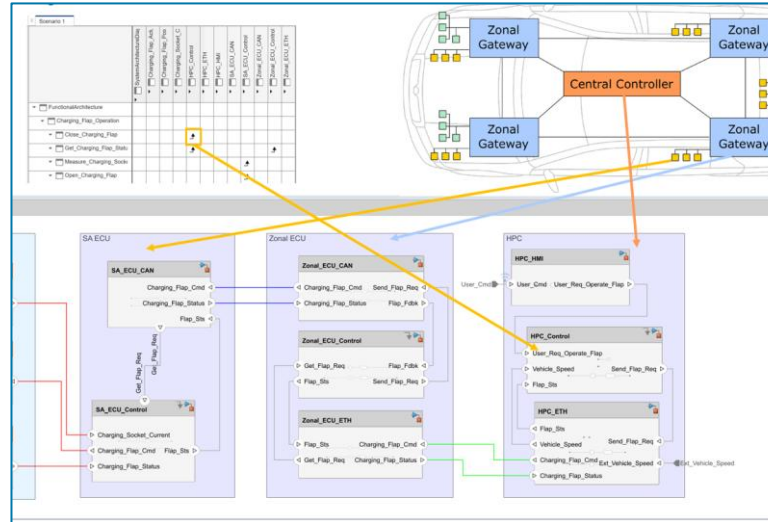


IncreasedSoftwareSize

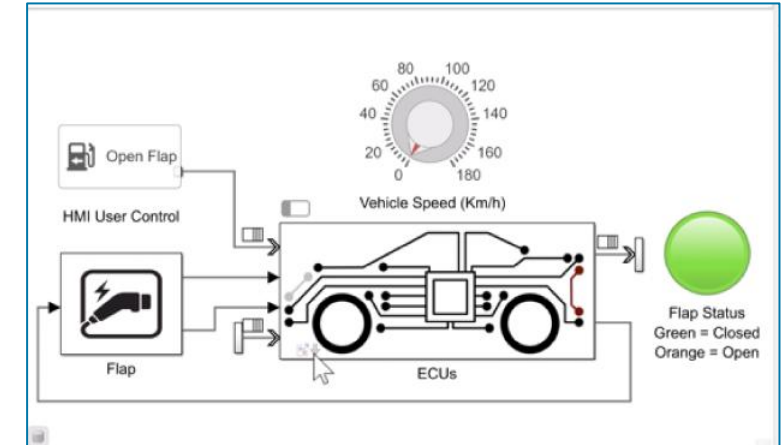
EnsuringSafetySecurity

Quality

Transition to SDV poses key development challenges



Unified workflow and traceability are key for impact analysis and collaboration at speed



Development teams can understand, trace, simulate, and validate system and software behavior early

Are you able to fully trace the impact of a change?



# Call to action: partner with KPIT and MathWorks in your SDV journey

**Learn more and visit:**



[www.mathworks.com/sdv](http://www.mathworks.com/sdv)



[www.kpit.com/software-defined-vehicle/](http://www.kpit.com/software-defined-vehicle/)

**Contact us:**



[sdv@groups.mathworks.com](mailto:sdv@groups.mathworks.com)



[alliances@kpit.com](mailto:alliances@kpit.com)

Thank you!