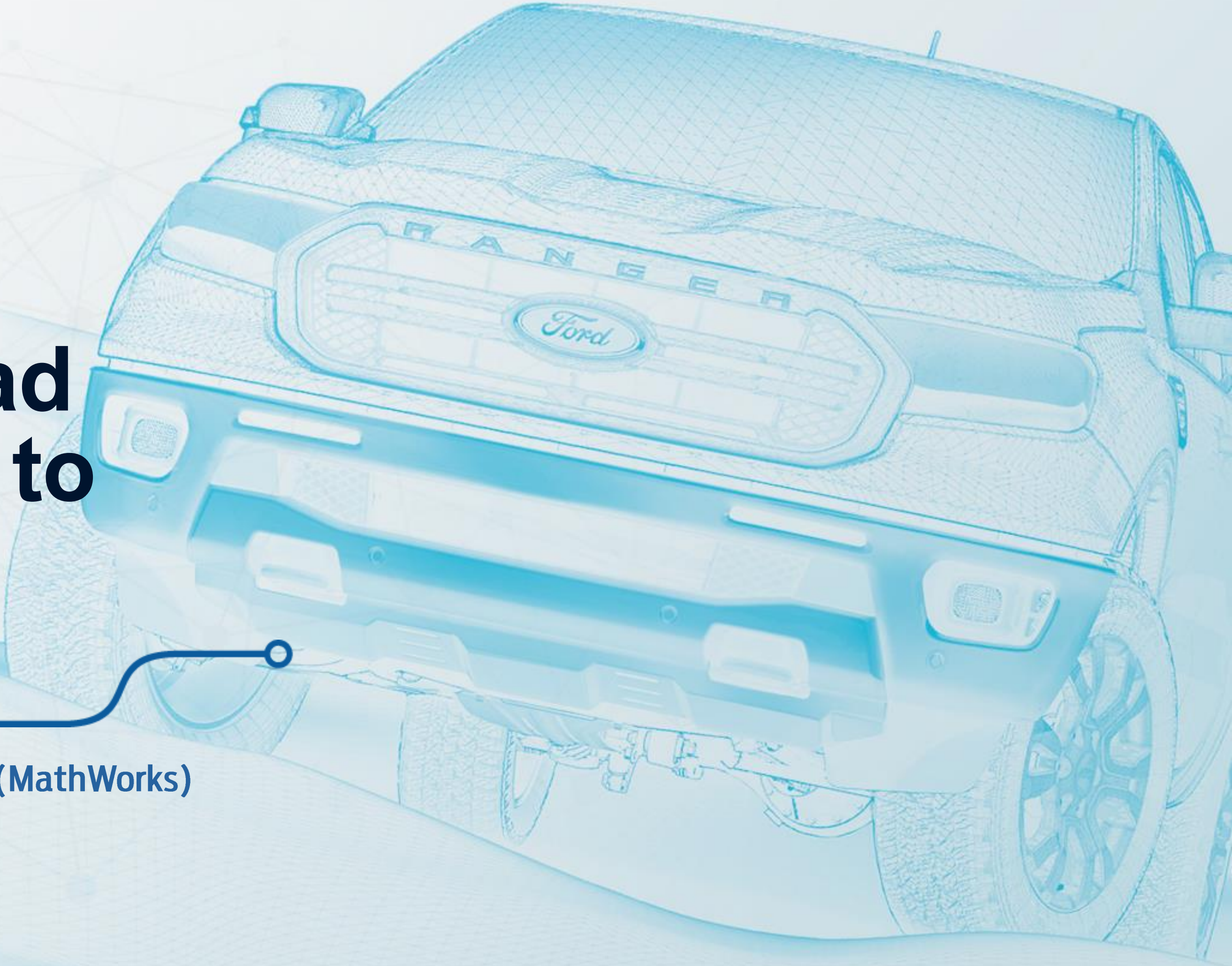
Vehicle Software &
Electronic Solutions

Building the Digital Thread between MBSE and MBD to Meet ISO26262 for Embedded Software



Authors: Joshua McCready (Ford), Hans Gangwar (Ford), Josh Kahn (MathWorks)

Problem Statement

Assessing ISO26262 Part 6 and ASPICE compliance for new and existing Ford In House software developed with Model Based Design software has demonstrated the need for additional best practices.

These best practices are needed to achieve connectivity to the System Engineering process and to allow for traceability and thread pulling of SW development artifacts*

*System and software requirements, model and data dictionary, implementation, test cases

Summary of Gaps found in Assessing ISO26262 Part 6

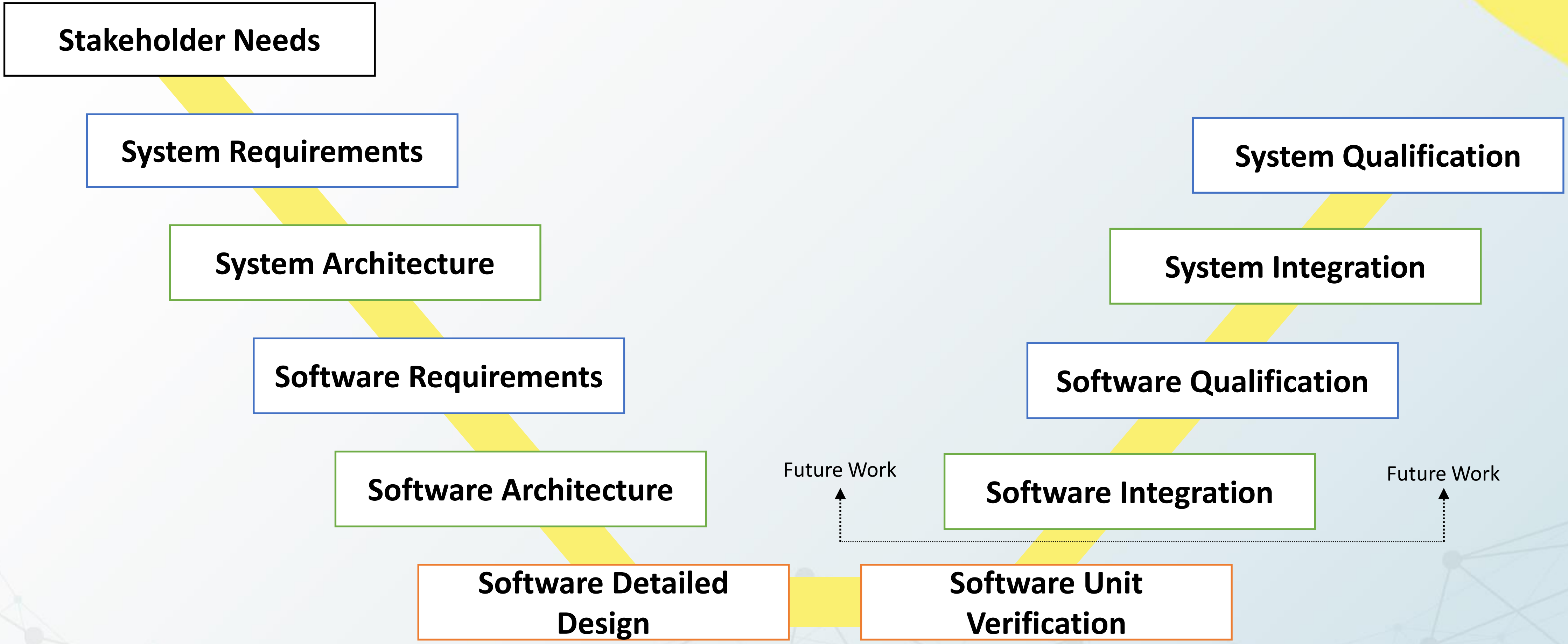
The following pain points were identified and targeted:

- Architecture models and implementation models were maintained in separate tools resulting in a poor connection between them
- Requirements were previously maintained in Microsoft Word with implicit linking to the Simulink implementation models resulting in the need for manual traceability
- Change tracking/impact analysis in models was difficult because one file contained all the subsystems
- Traceability between requirements, models, and tests was maintained in a Microsoft Excel spreadsheet resulting a labor-intensive process change management process
- Relationships between high-level requirements, implementation requirements, implementation, and test cases were implicit making validation of high-level requirements difficult

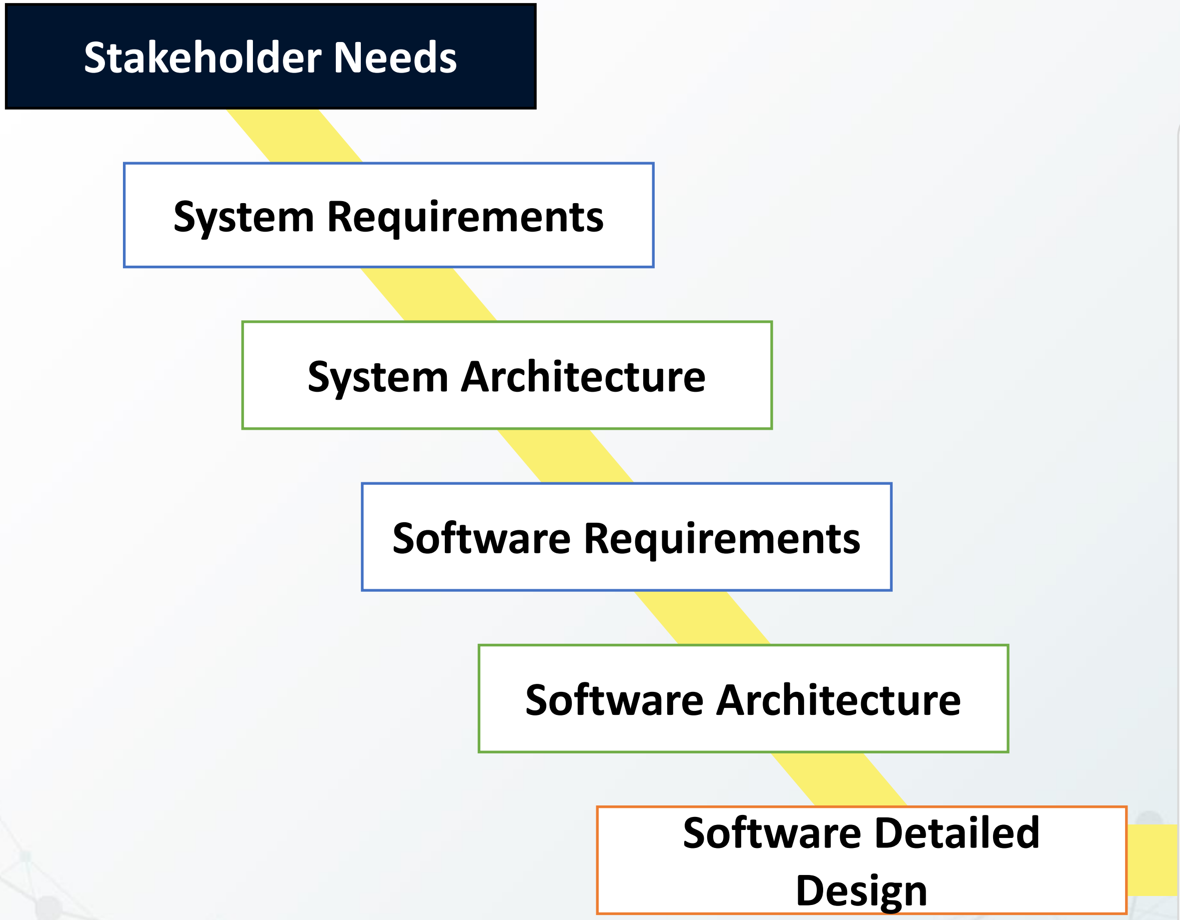
Solution

- Adopted an Integrated MBSE – MBD workflow to better connect system and software design artifacts
  - Created software functional architecture from required system functions via functional decomposition, allowing for focus on main SW function inputs and outputs upfront
  - Created software technical architecture that connects to system technical architecture and production model, allowing for nesting up and down the System V
- Limited the duplication of sources of truth
- Used a requirements management tool enabling requirements being machine readable, have relationships between requirements, and traceability to other System V artifacts
- Adopted a componentized modeling style (Model Reference and Reference Data Dictionary) enabling impact analyses upon changes and traceability to other System V artifacts
- Continued use of Simulink Test to perform requirements-based SW V&V with machine readable requirements, enabling impact analyses upon changes and traceability to other System V artifacts

Process Overview



Process Overview – Stakeholder Needs



Stakeholder Needs

Organization-level requirements are captured as Stakeholder Needs and Concept of Operations then decomposed into the System Requirements.

Process Overview – System Requirements

Vehicle Software & Electronic Solutions

Stakeholder Needs

System Requirements

System Architecture

Software Requirements

Software Architecture

Software Detailed Design

Structured System Requirements

System requirements are maintained in a tool outside MathWorks and split into three categories:

- Functional Safety Requirements
- Technical Safety Requirements
- System Functional Requirements

System Integration

Software Qualification

Software Integration

Software Unit Verification

Process Overview – System Architecture

Vehicle Software & Electronic Solutions

Stakeholder Needs

System Requirements

System Architecture

Software Requirements

Software Architecture

Software Detailed Design

Implementation of System Requirements

The System Architectures are implemented in either an outside tool or System Composer and split into a Functional Architecture and a Technical Architecture.

System Functional Architecture

Supports Failure Mode Analysis, Safety Goals, System Functional Requirements

Allocations

System Technical Architecture

Supports Functional and Technical Safety Requirements, System Functional Requirements

Process Overview – Software Requirements

Stakeholder Needs

System Requirements

System Architecture

Software Requirements

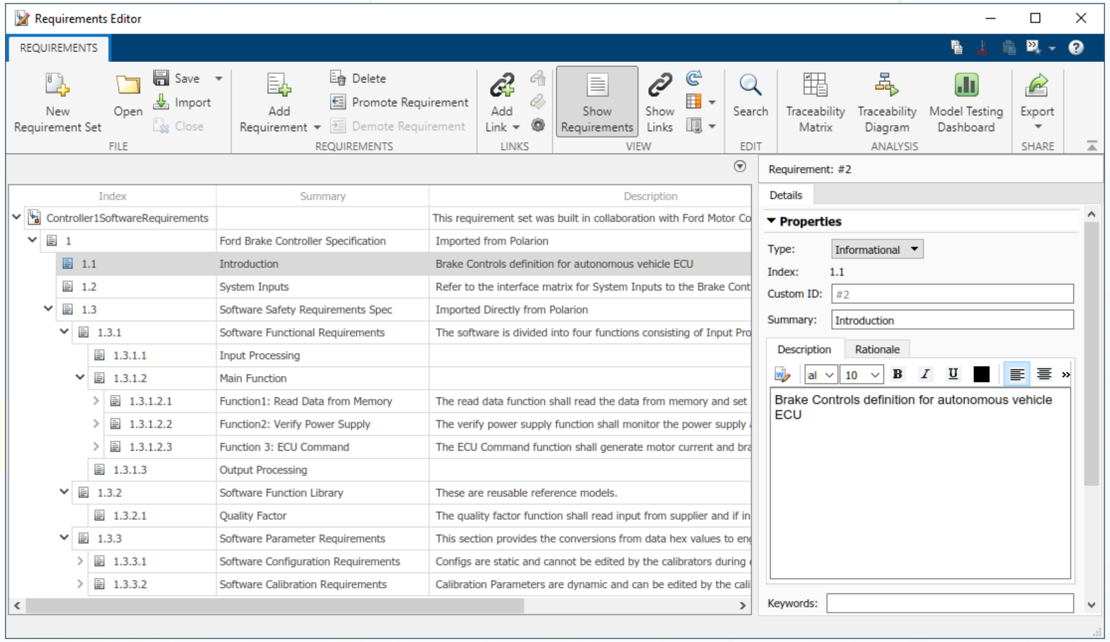
Software Architecture

Software Detailed Design

Structured Software Requirements

The software requirements are decomposed from the System Requirements and maintained in an outside tool. Then, they are imported into Simulink Requirements via ReqIF* to establish traceability within the MathWorks toolchain.

All software requirements can be considered as Software Safety Requirements, some simply being QM if they support no Technical Safety Requirement.



*Requirements Interchange Format

Process Overview – Software Architecture

Vehicle Software & Electronic Solutions

- Stakeholder Needs
- System Requirements
- System Architecture
- Software Requirements
- **Software Architecture**
- Software Detailed Design

Software Architecture Fits Requirements Structure
The Software Architecture is built in System Composer and matches the structure of the Software Requirements.

Artifacts can be shared between the Software Architecture and the detailed production software model such as interfaces between different SW components.

Software Functional Architecture*

Supports Failure Mode Analysis and Safety Goals
*Future Work

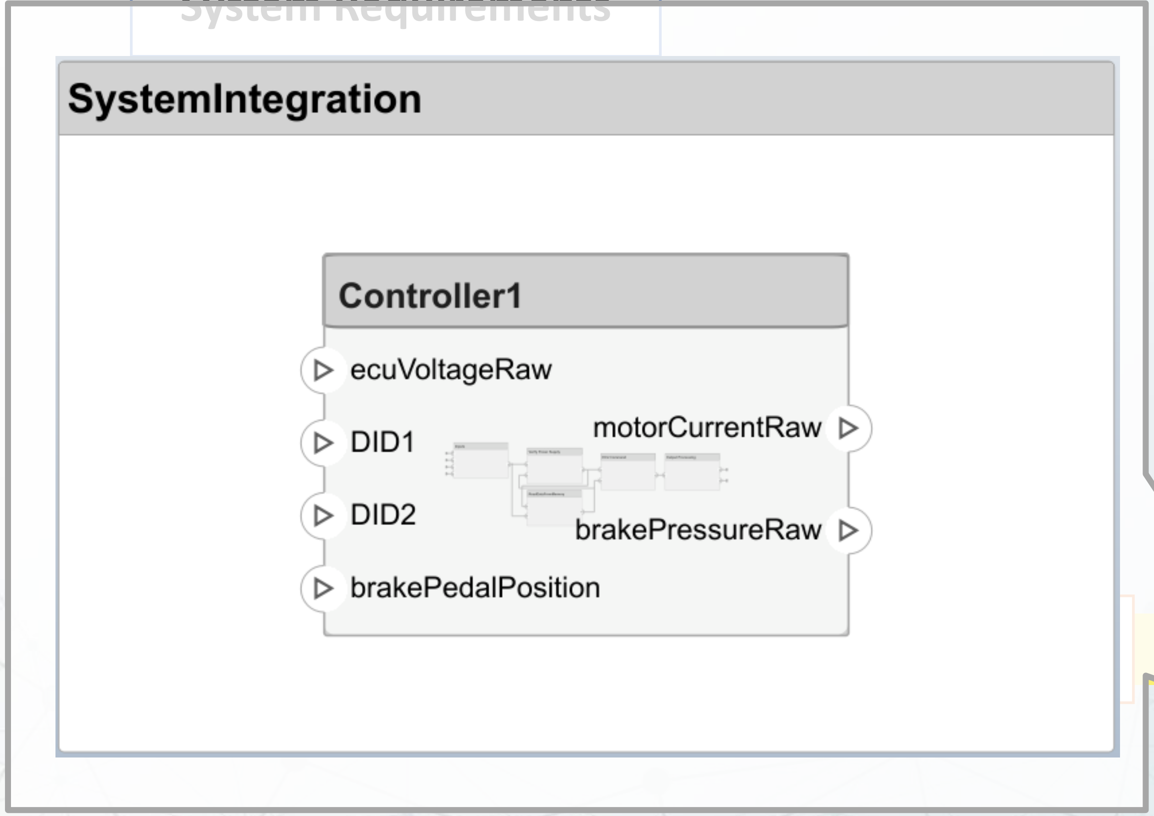
Allocations

Software Technical Architecture

Supports Technical Software Safety Requirements

Process Overview – Software Requirements and Architecture

Stakeholder Needs



Software Architecture Fits Requirements Structure

The Software Architecture is built in System Composer and matches the structure of the Software Requirements.

Artifacts can be shared between the Software Architecture and the detailed production software model such as interfaces between different SW components.

Software Functional Architecture*

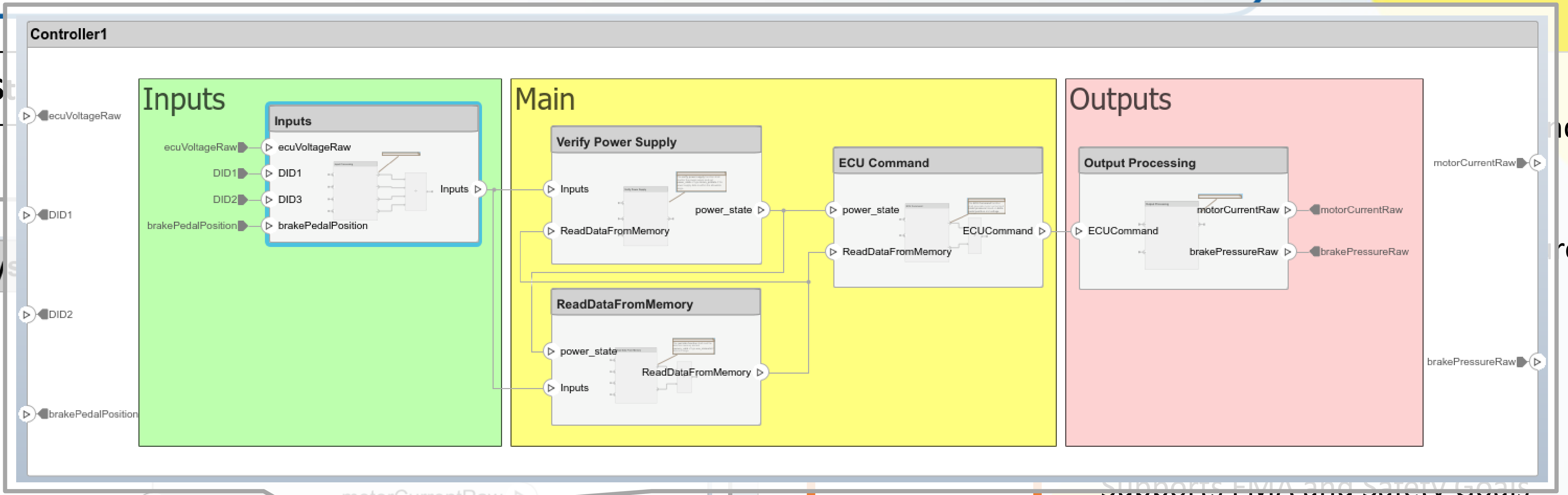
Supports FMA and Safety Goals
*Future Work

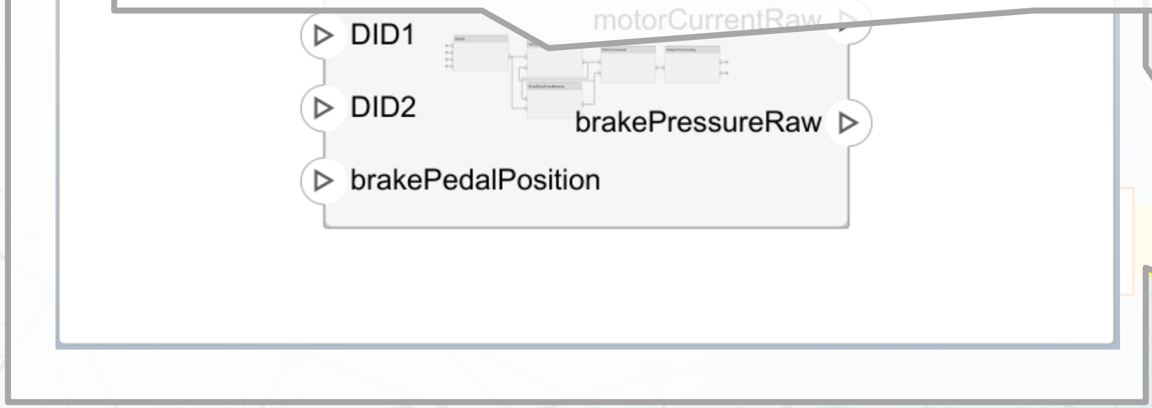
Allocations

Software Technical Architecture

Supports Technical Software Safety Requirements

Process Overview – Software Detailed Design – 3 Pillars





Functional Architecture*

Supports FMA and Safety Goals

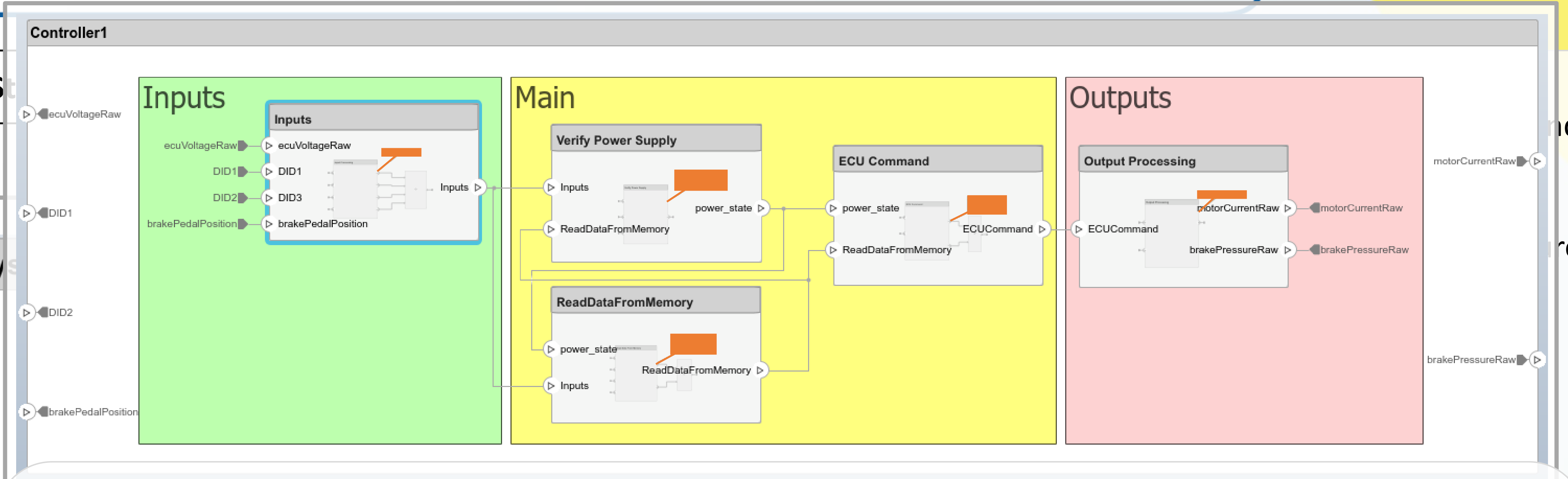
*Future Work

Allocations

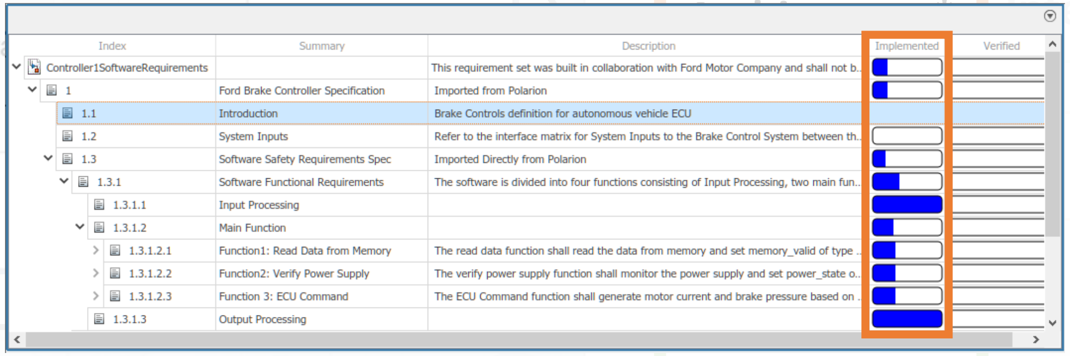
Software Technical Architecture

Supports Technical Software Safety Requirements

Process Overview – Software Detailed Design – Native Requirement Linkage



Requirements are linked from Simulink Requirements to their associated architectural elements for direct traceability.

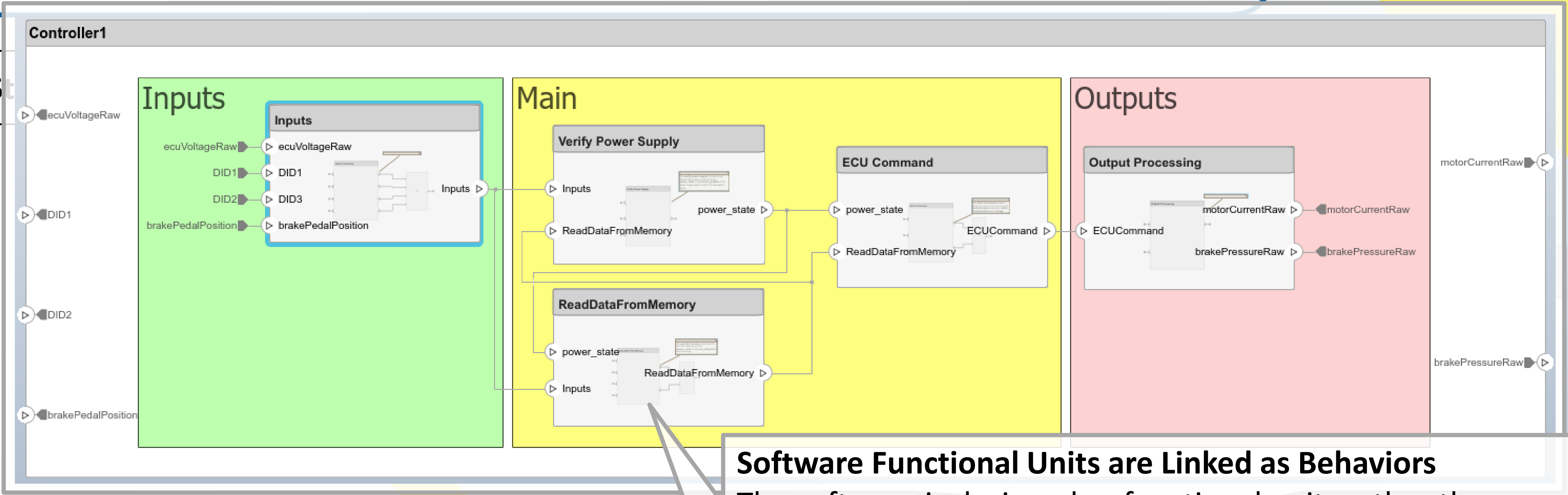


Index	Summary	Description	Implemented	Verified
Controller1SoftwareRequirements		This requirement set was built in collaboration with Ford Motor Company and shall not b...		
1	Ford Brake Controller Specification	Imported from Polarion		
1.1	Introduction	Brake Controls definition for autonomous vehicle ECU		
1.2	System Inputs	Refer to the interface matrix for System Inputs to the Brake Control System between th...		
1.3	Software Safety Requirements Spec	Imported Directly from Polarion		
1.3.1	Software Functional Requirements	The software is divided into four functions consisting of Input Processing, two main fun...		
1.3.1.1	Input Processing			
1.3.1.2	Main Function			
1.3.1.2.1	Function1: Read Data from Memory	The read data function shall read the data from memory and set memory_valid of type ...		
1.3.1.2.2	Function2: Verify Power Supply	The verify power supply function shall monitor the power supply and set power_state o...		
1.3.1.2.3	Function 3: ECU Command	The ECU Command function shall generate motor current and brake pressure based on ...		
1.3.1.3	Output Processing			

Supports Technical Software Safety Requirements

Process Overview – Software Detailed Design – Model Reference

Vehicle Software & Electronic Solutions

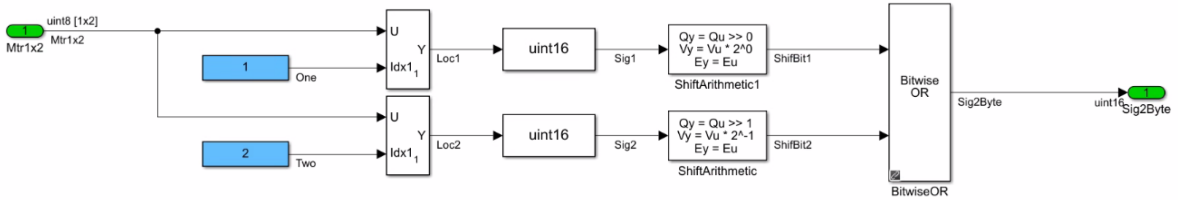


Software Architecture

Software Detailed Design

Software Functional Units are Linked as Behaviors

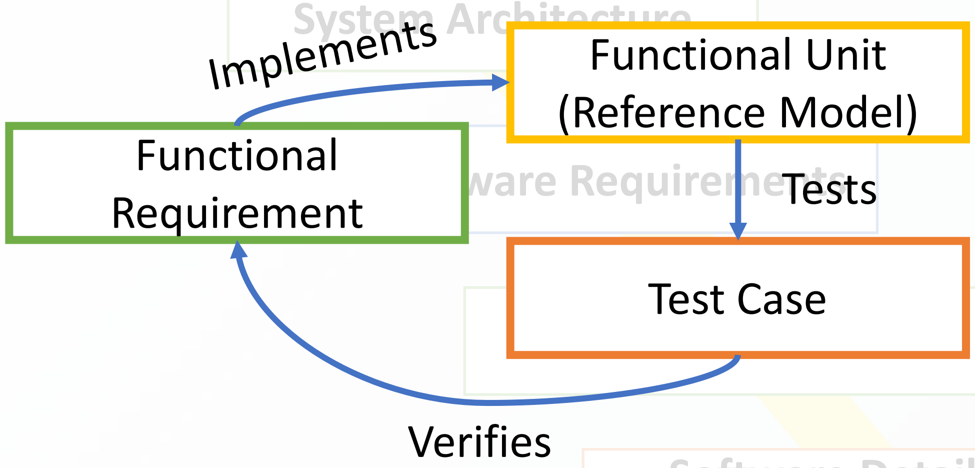
The software is designed as functional units rather than one large model, facilitating work-split, piece-wise integration, and impact analysis through **Model Reference**. These units exists as separate .slx files and are collected into a parent .slx file.

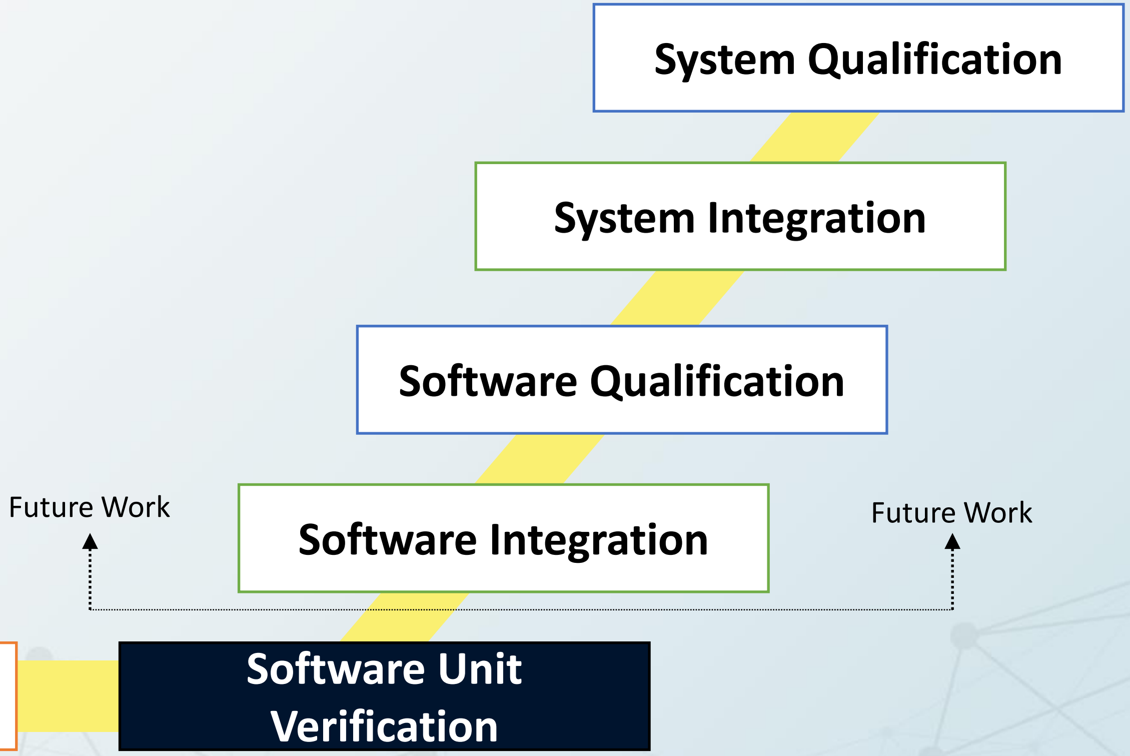


Process Overview – Software Unit Verification

Software Unit Verification Underway

Functional software requirements are in the process of being linked to Simulink Test Cases for verification and coverage analysis.

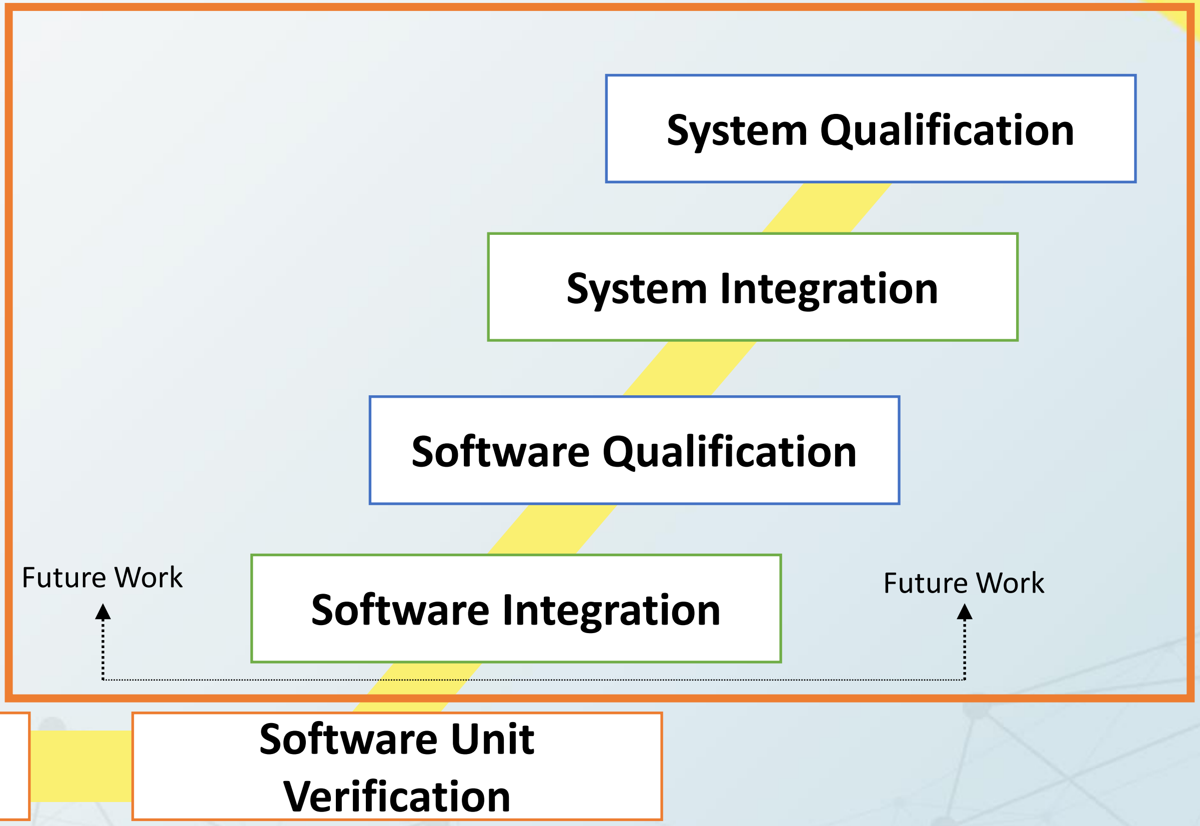




Process Overview – Ongoing Work

Next Steps

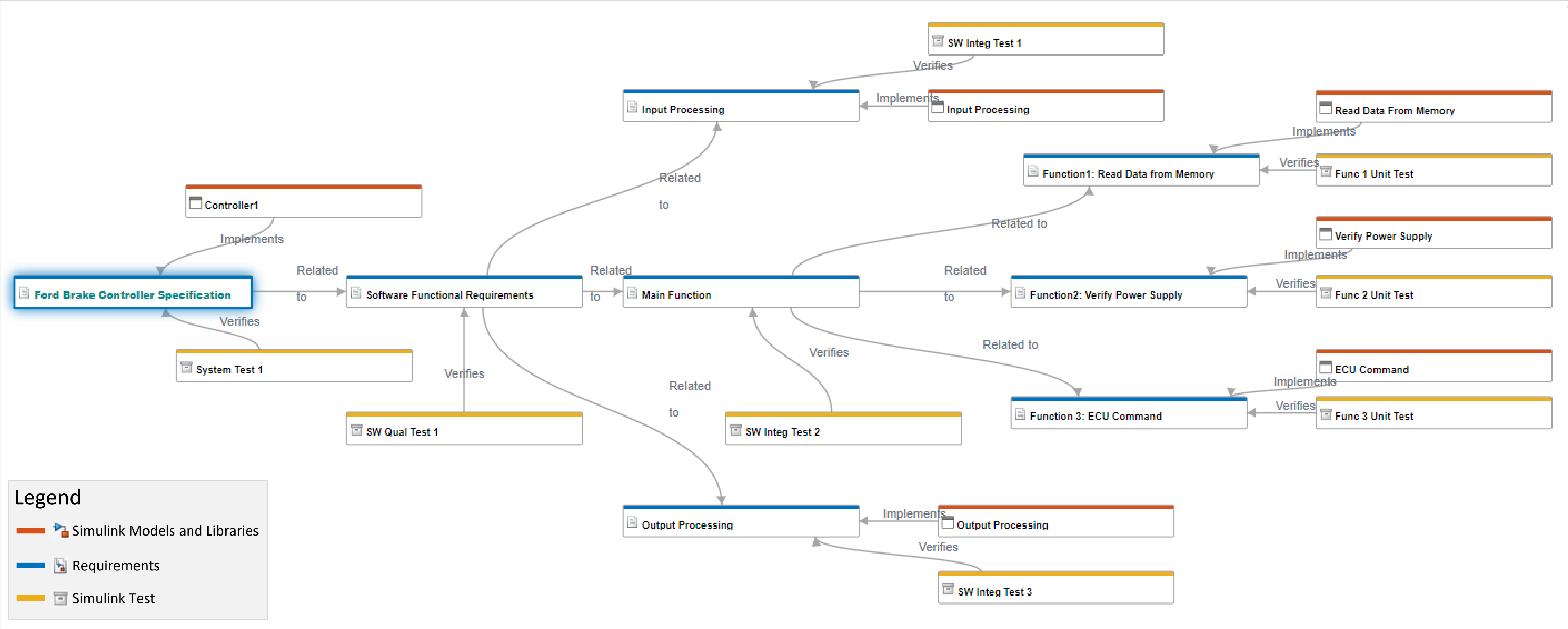
- Continually feedback Software Detailed Design to Software Architecture
- Create Design Verification Methods
- Link test cases to Design Verification Methods
- Create the Software Integration and Qualification Test Suites
- Identify dependencies of software integration and qualification testing and how to establish traceability across the project artifacts
- Develop System Integration and Qualification Tests
- Integrate Software Architecture with System Architecture



Analyzing Traceability

Thread-Pulling Using Traceability Diagram

The Traceability Diagram feature of Simulink Requirements (introduced in R2021b) is planned to be used for thread-pulling activities



Conclusion

- Using a requirements management tool enabled machine readable requirements allowing for greatly improved linking of artifacts
- Creating a software technical architecture model helped develop software implementation requirements and key artifacts can be shared between it and a production model that implements the detailed software design
- Applying a system engineering approach to create a software functional architecture improves ability for up front design
- Adopting a Model Reference and Reference Data dictionary modeling style enables easier impact analysis and makes generated code easier to read when paired with use of non-virtual buses
- Thread pulling of Technical Safety Requirements is done automatically with Traceability Diagrams in Simulink Requirements/Views in System Composer, enabling review that the Technical Safety Requirements are met and fully validated

Linked Library versus Model Reference MBD Comparison

Vehicle Software & Electronic Solutions

Linked Library File Structure

Main.slx

- Main_functions.slx (linked library)
- Reuse_units.slx (linked library)

Main.sldd

Calibration.sldd (imported from header file)

Model Reference File Structure

Main.slx

- Submain_function1.slx
  - unit_function1.slx
  - unit_function2.slx
  - unit_function3.slx
- Submain_function2.slx >
- Submain_function3.slx >
- Submain_function4.slx >
- Submain_function5.slx >

Main.sldd

- Config.sldd (imported from header file)
- Calibration.sldd (imported from header file)
- NonVirtualBus.sldd
  (creates bus objects that appear in generated code)

Thank You

Thank you for joining us today.

Please direct any follow-up questions to:

Joshua McCready
jmccrea8@ford.com

Josh Kahn
joshkahn@mathworks.com