MathWorks
AUTOMOTIVE CONFERENCE 2023
Europe

Agile Behavior-Driven and Test-Driven Development with Model-Based Design

Marc Segelken, MathWorks

MathWorks

Agenda

- Definition Behavior-Driven and Test-Driven Development
- How can I write good requirements?
- How do BDD and TDD work with Model-Based Design?
- Example of BDD with MBD

Key Takeaway: How to apply BDD and TDD in Model-Based Design

Introduction
Problem statement

- Textual requirements are often incomplete, inconsistent, incorrect and ambiguous.
- Testing often happens very late during a development cycle and it is rushed due to deadlines.

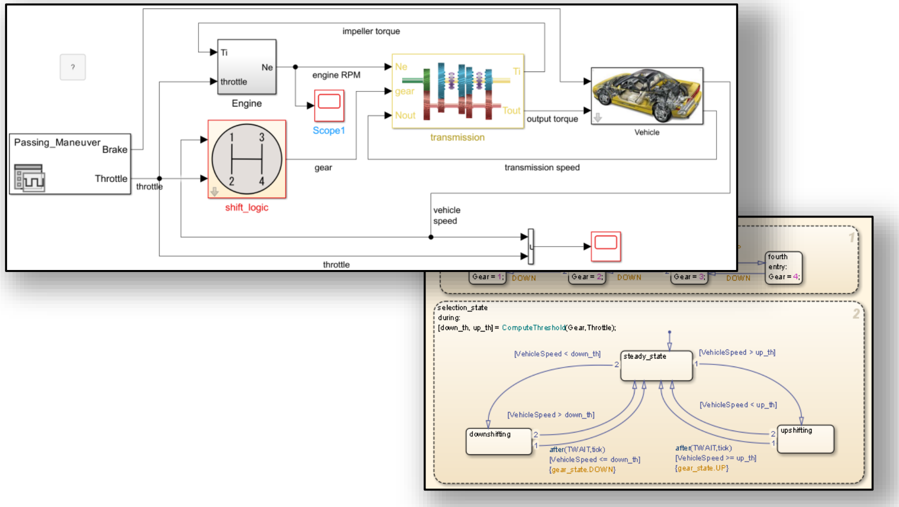
How can you prevent this?

The Scaled Agile Framework (SAFe) suggests the “test-first” approaches of Behavior-Driven Development and Test-Driven Development.

How can BDD and TDD be practiced in the context of Model-Based Systems Engineering and Model-Based Design?

Introduction
Model-Based Design

- Simulink and Stateflow referenced in standards for safety-critical system development such as, e.g., ISO 26262
- Higher level of abstraction
- Enables early validation of requirements through simulation
- Generate documentation and production code automatically, saving effort and eliminating manual error



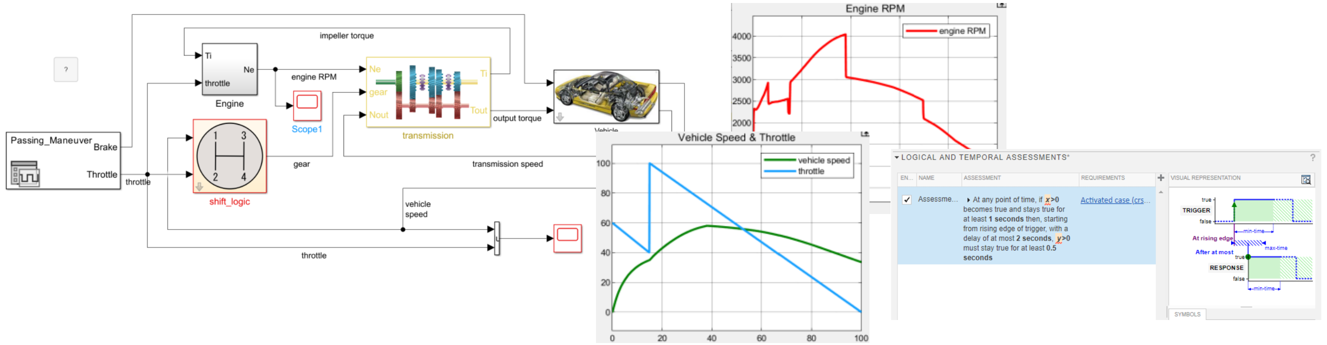
Benefit Hypothesis:

***I believe** that*
Model-Based Design
with system simulation
and production code
generation
***will result in** increased*
productivity and efficiency
in large-scale software
projects,
as measured by
correctness of
requirements and
quality of generated
production code.

Introduction
Behavior-Driven Development (BDD)

"BDD is a collaborative process that creates a shared understanding of requirements between the business and the agile teams." (SAFe)

- Requirements elicitation is team effort, leads to understanding of
  - **what** needs to be done
  - **why** it should be done
  - **when** it will be good enough
- Easier to see what behavior is needed with examples in action
- BDD with system simulation is an effective way to get a baseline for requirements with acceptance criteria.



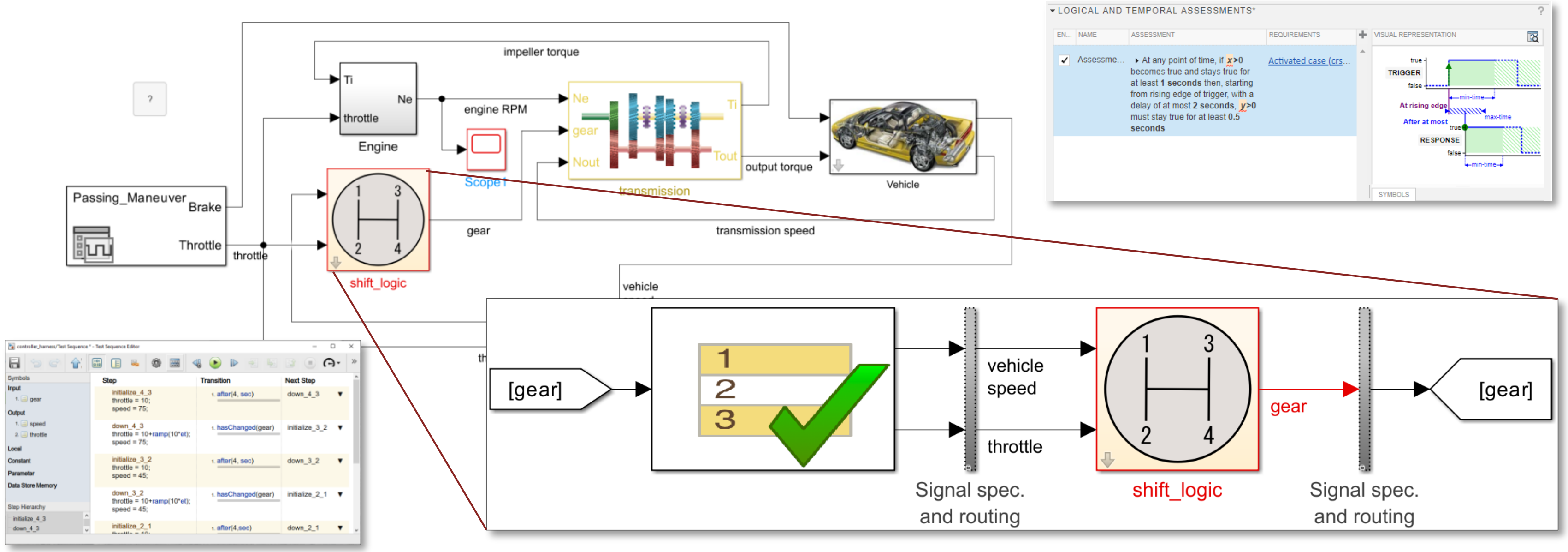
Benefit Hypothesis:

***I believe that** Behavior-Driven Development (BDD) in the combination with system simulation capabilities of Model-Based Design*
***will result in** built-in quality*
***as measured by** the efficient elicitation of correct system requirements and early validation of correct behavior of the system.*

Introduction
Test-Driven Development (TDD)

"[..] building and executing tests before implementing the code or a system component" (SAFe)

- Simulink simplifies creating test-frames including assessments, and mocking environments
- re-use the tests for generated code.

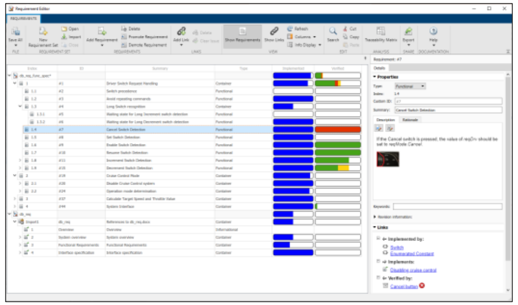


Benefit Hypothesis:

***I believe** that Test-Driven Development (TDD) in the combination with the executable specification (simulation) capabilities, model-level verification, and code generation capabilities of Model-Based Design **will result in** built-in quality, **as measured by** verified correct behavior of models and generated code on a component level.*

The Role of Requirements

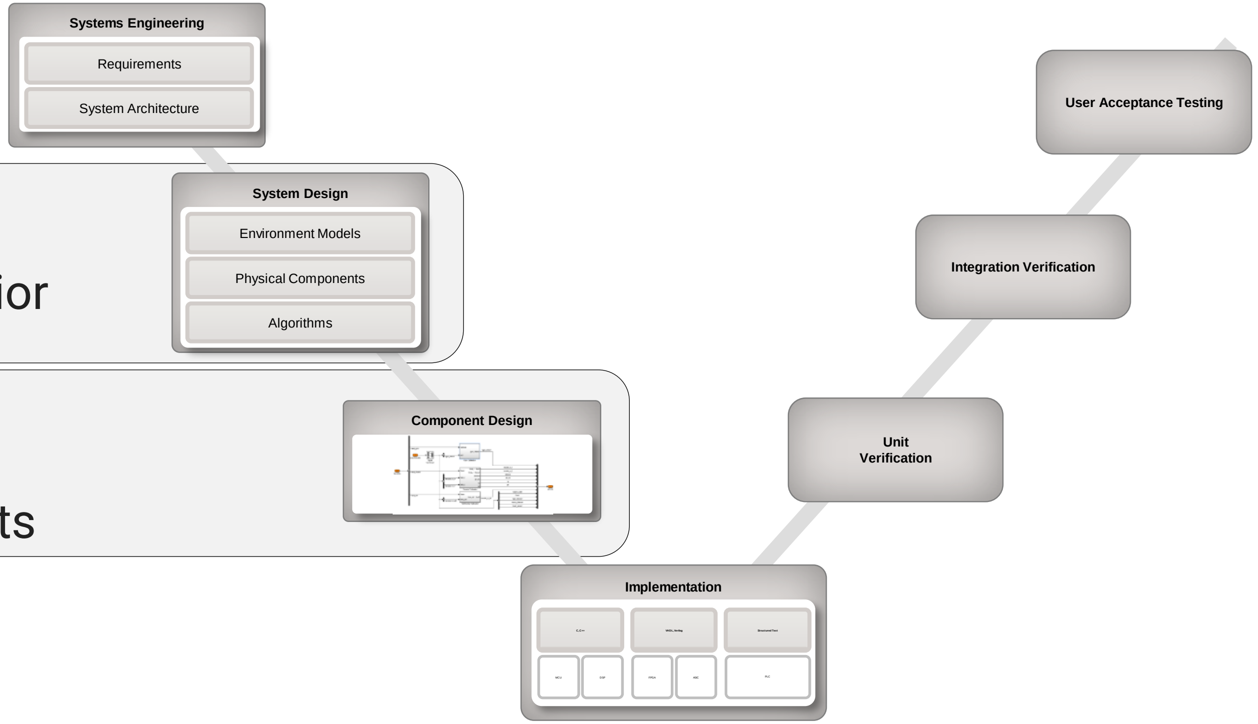
The Importance of Good Requirements



Starting point for BDD and TDD:
Testable requirements with acceptance criteria

BDD
desired functionality or system behavior

TDD
more detailed specification derived
from the system behavior requirements



The Role of Requirements in an Agile Framework

How can I write good requirements?

Benefit Hypothesis	I believe that <capability>, Will produce <outcome>, As measured by <metric>.	To communicate business and customer relevance	WHY?
User Stories	As a <role>, I want to <need>, So that <goal>.	To give context	WHAT?
Acceptance Criteria	Given <precondition>, When <trigger action>, Then <expected result>.	To communicate quality assurance criteria	WHEN WILL IT BE GOOD ENOUGH?

The Role of Requirements

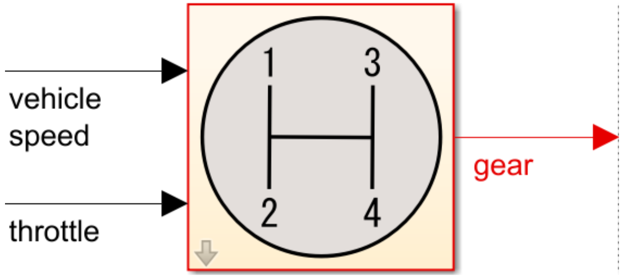
How Can I Write Good Requirements? Example

Requirement	**Issue**
The vehicle shall support DC charging of the battery.	This requirement is vague and incomplete with no clear acceptance criteria.
The vehicle shall support DC fast charging at 50kW…300kW.	The requirement now is clearer, but it is still incomplete.
As an electric vehicle driver, **I want to** have the option to charge my electric vehicle fast, **So that I** can get additional driving range in a short amount of time. **Given the battery's state of charge is below 50%,** **And the vehicle is plugged into a DC fast charging station capable of up to 320 kW,** **When the battery is charged for 5 minutes,** **Then the vehicle shall be able to travel for at least an additional 100 km.**	The end-user is put into focus and context is given by the user-story. The acceptance criteria is formulated in such a way that a test could be derived from it easily without too much interpretation.

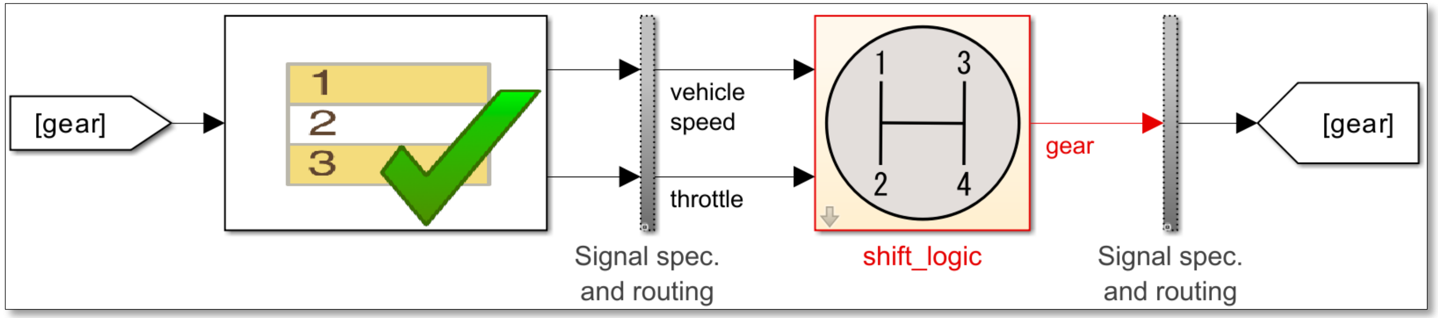
How do BDD and TDD work with Model-Based Design?

- BDD and TDD follow "test-first" approach:

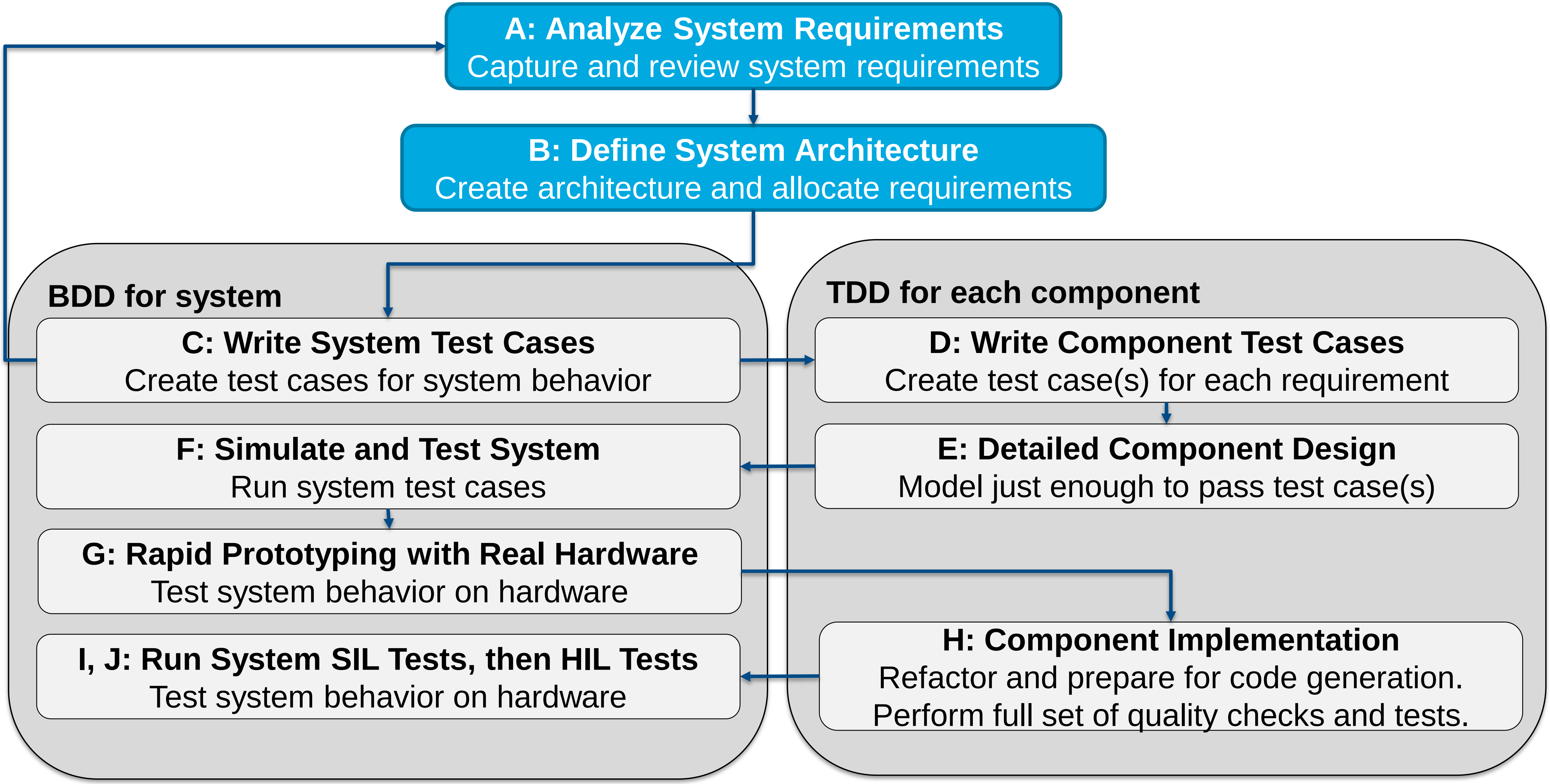
1. Have **clear requirements** and **clear acceptance criteria**
2. A **minimum architecture or design**



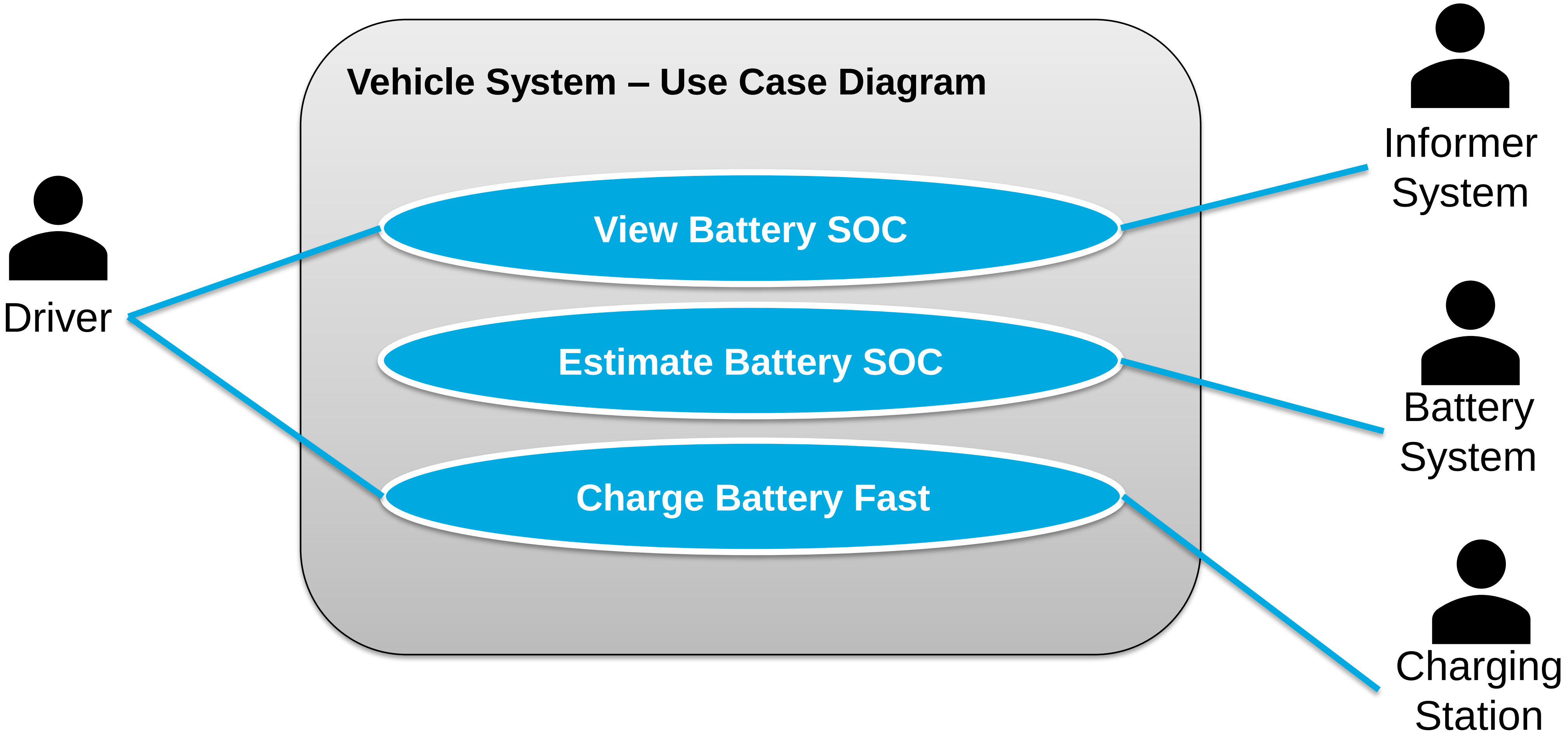
3. Only then start **building test harnesses and test cases with assessments.**



BDD and TDD lifecycles incorporating the benefits of MBD



A: Analyze System Requirements
Capture and review system requirements – Use Case Diagram



A: Analyze System Requirements

Capture and review system requirements - "WHY?"

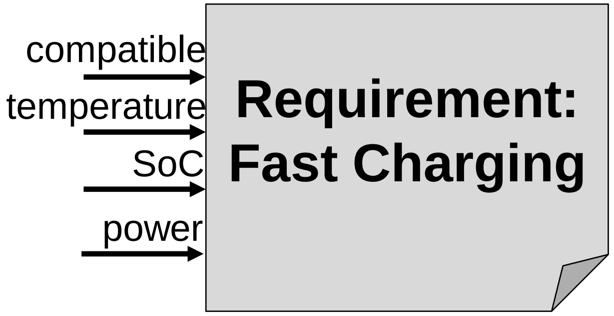
- Benefit Hypothesis: ("WHY?")
  - **I believe**, that DC fast charging in the range of 50 kW to 320 kW,
  - **will result in** greater adoption of electric vehicles due to more flexibility in charging options,
  - **as measured by** #users indicating DC fast charging as a significant reason for choosing the electric vehicle.

A: Analyze System Requirements

Capture and review system requirements – “WHAT? WHEN?”

- User Story: (“WHAT?”)
  - **As an** electric vehicle manufacturer,
  - **I want to** integrate DC charging stations with my EV charging system,
  - **So that** the vehicles can charge fast, safe and efficiently.

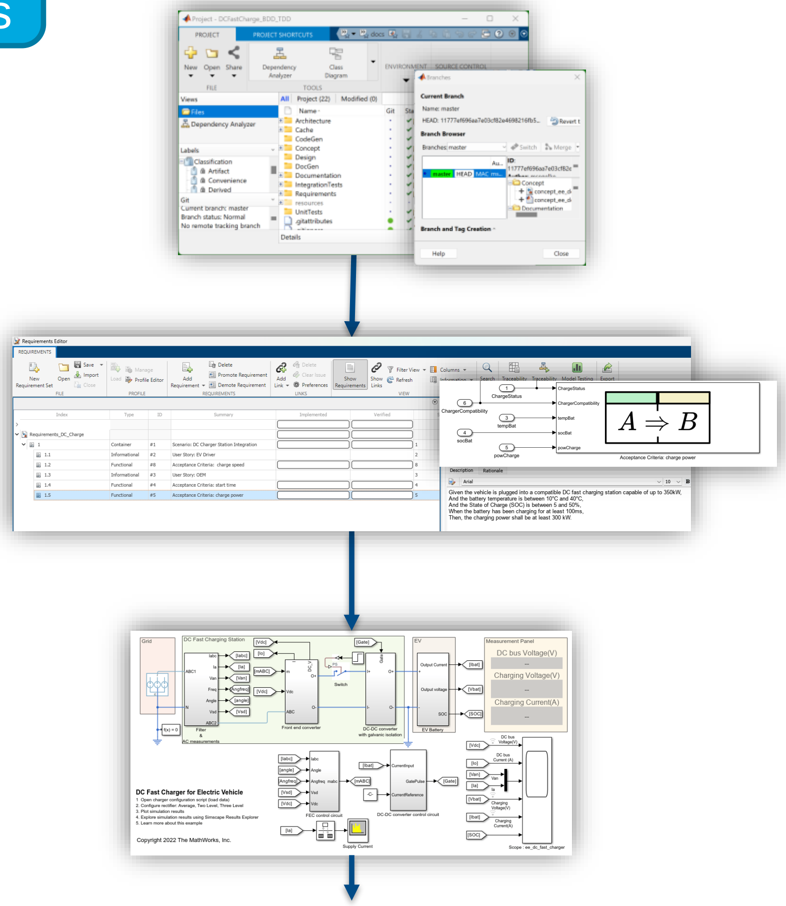
- Acceptance Criteria: (“WHEN IS IT GOOD ENOUGH?”)
  - **Given** the vehicle is plugged into a compatible DC fast charging station capable of up to 320kW,
  - **And** the battery temperature is between 10°C and 40°C,
  - **And** the State of Charge (SOC) is between 5 and 50%,
  - **When** the battery has been charging for at least 100ms,
  - **Then**, the charging power shall be at least 300 kW.



A: Analyze System Requirements
Capture and review system requirements

A: Analyze System Requirements
Capture and review system requirements

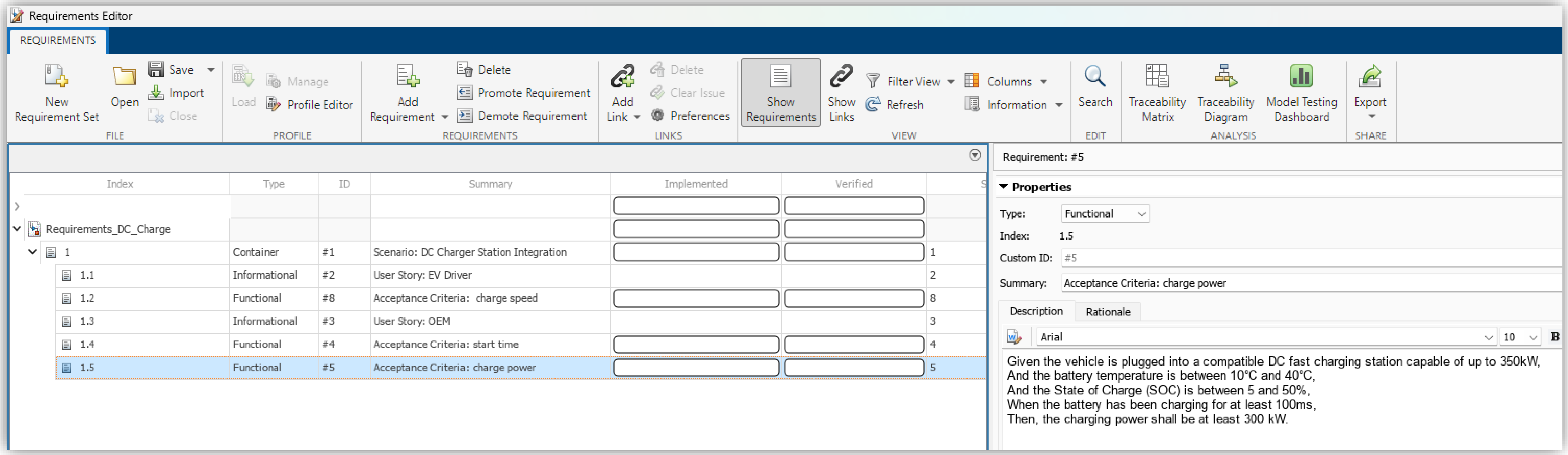
- Work in a structured way:
  - Create a MATLAB Project
  - Use Version Control,
    e.g., GIT that is integrated with MATLAB Project
  - Create a Requirement Set
  - Create a Requirements Model
  - Create Virtual Prototype Model
  - Do systematic analysis



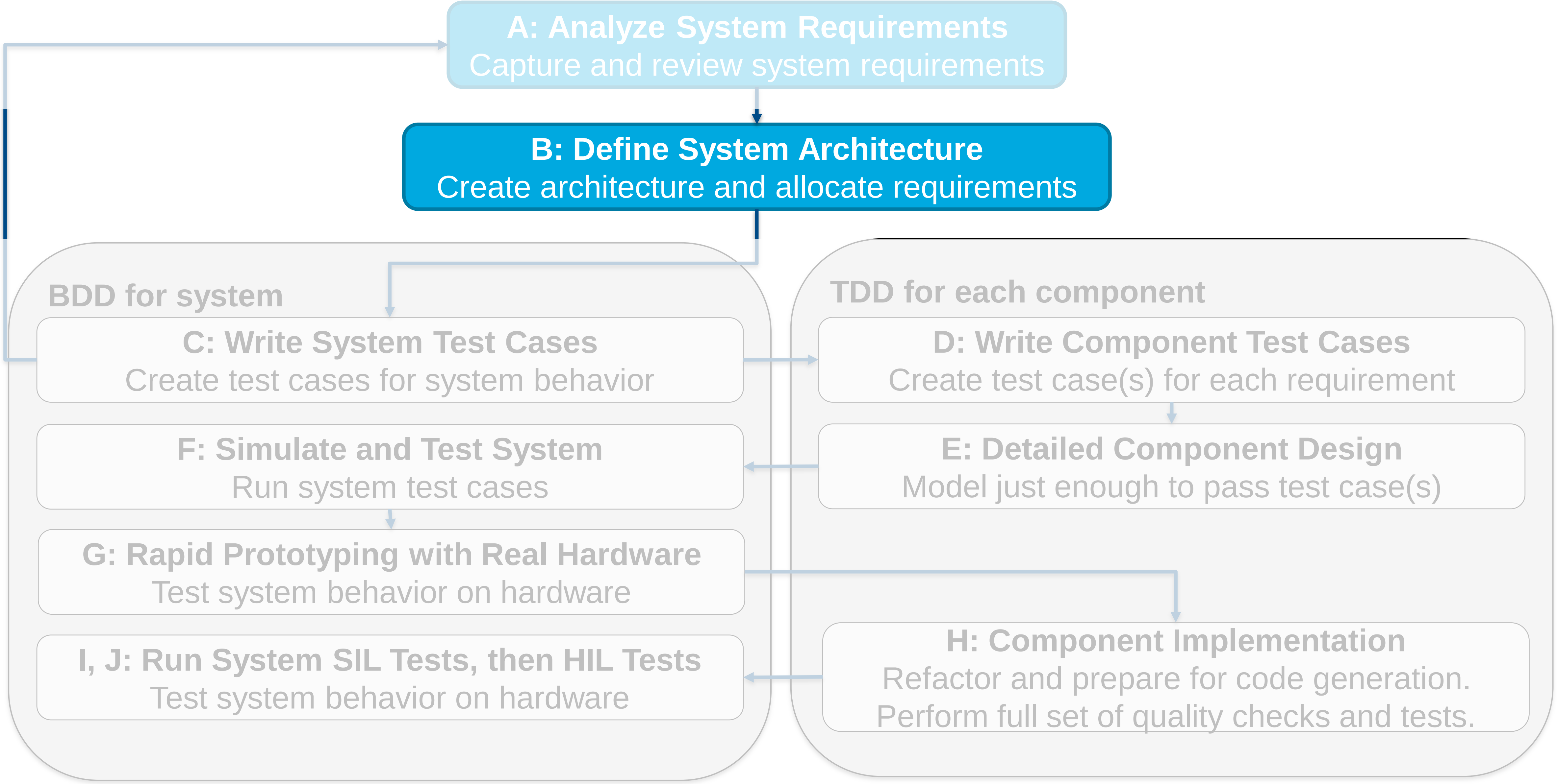
A: Analyze System Requirements

Capture and review system requirements

- Create a Requirement Set
- Requirements Editor (Create / Import / Update Requirements)
- Keep track of changes and do change impact analysis



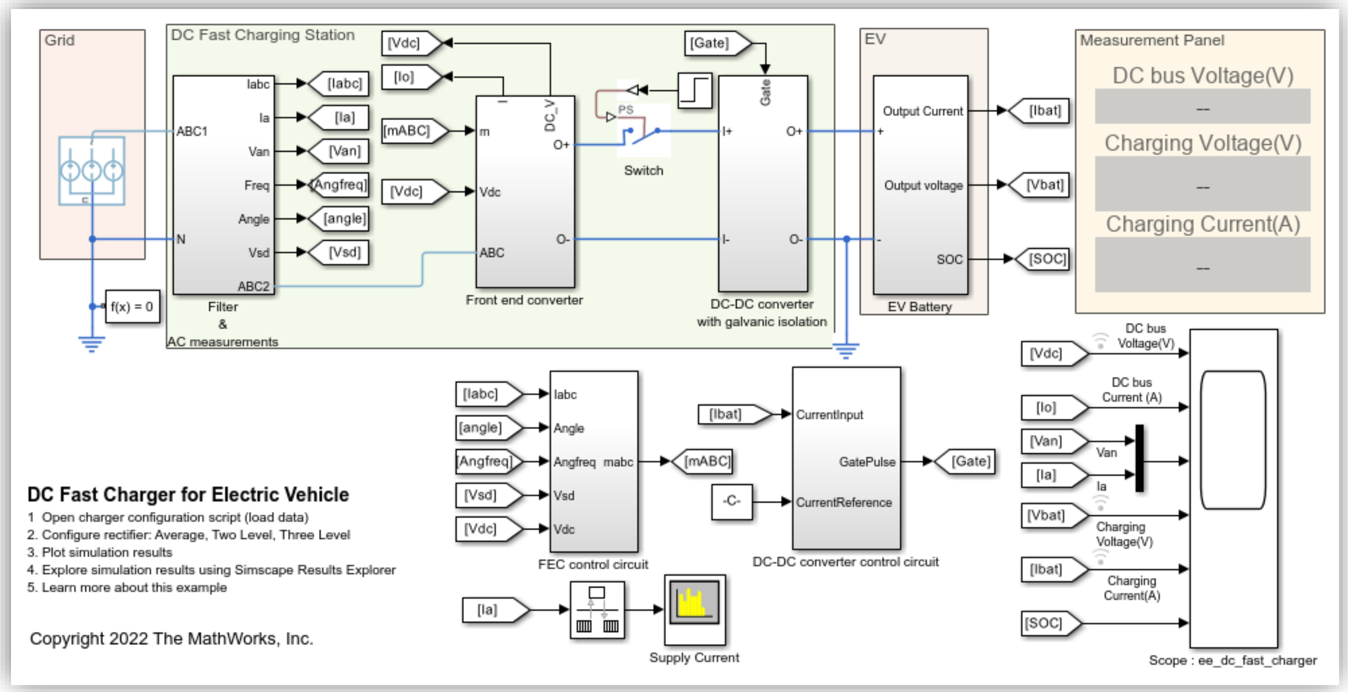
BDD and TDD lifecycles incorporating the benefits of MBD

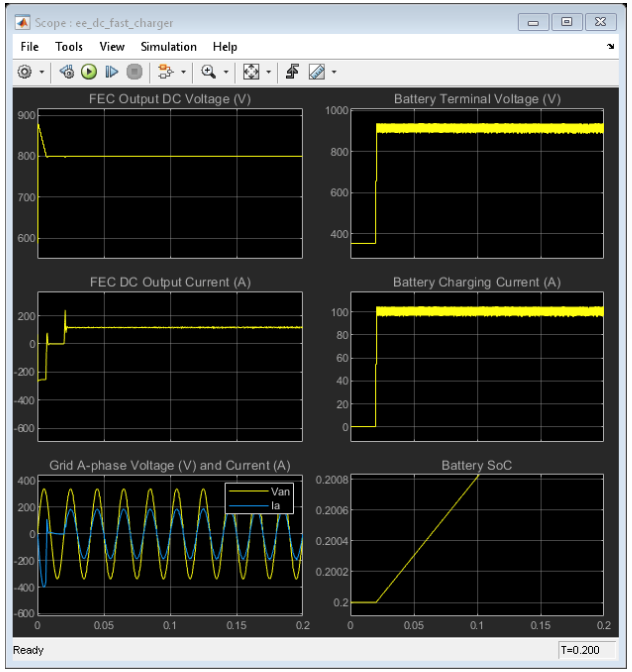


B: Define System Architecture

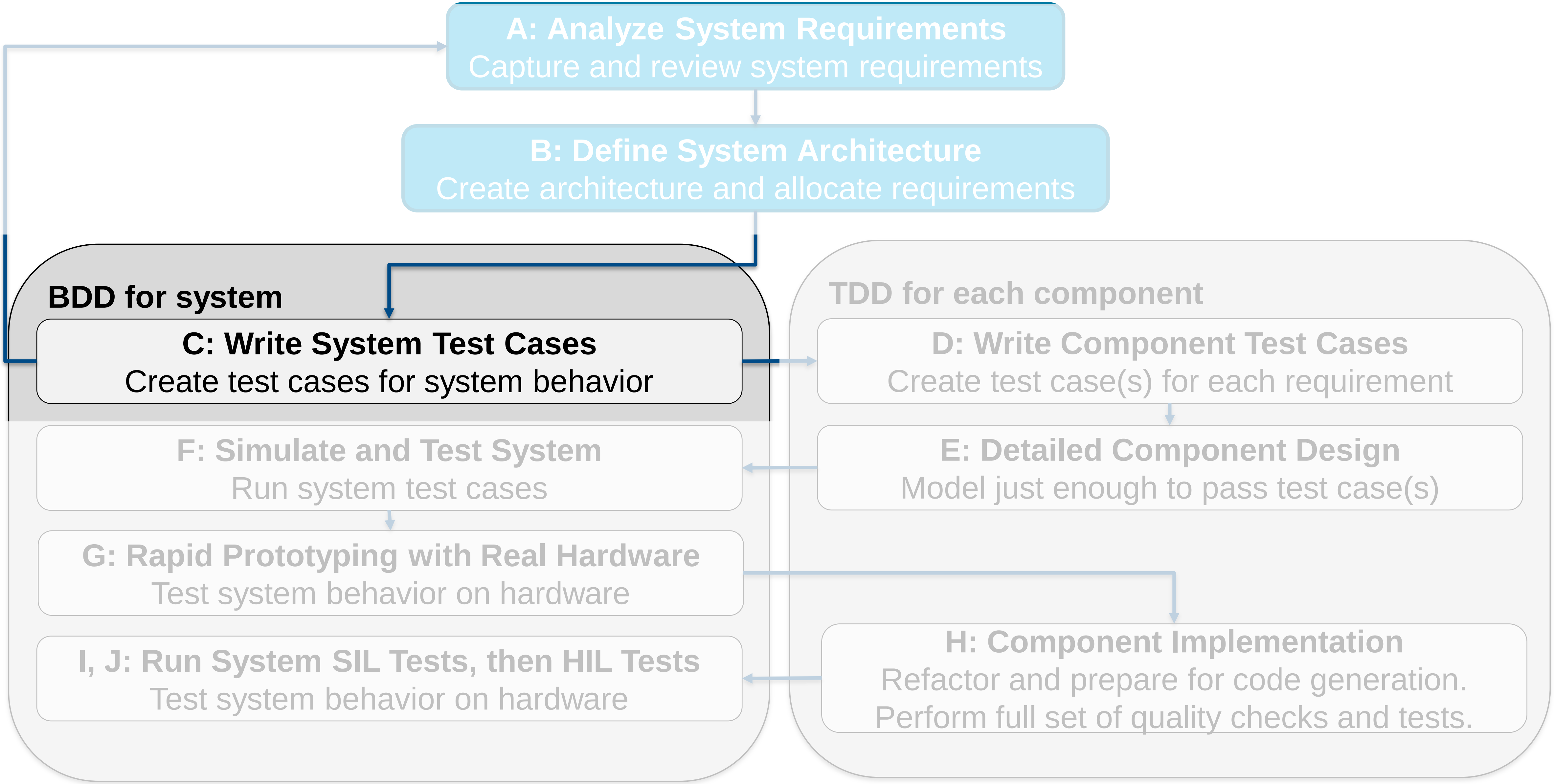
Concept phase: use virtual prototypes to gain better understanding

- Use a virtual prototypes
- Get started quickly making use of reference examples, e.g.,
- [DC Fast Charger for Electric Vehicle]()





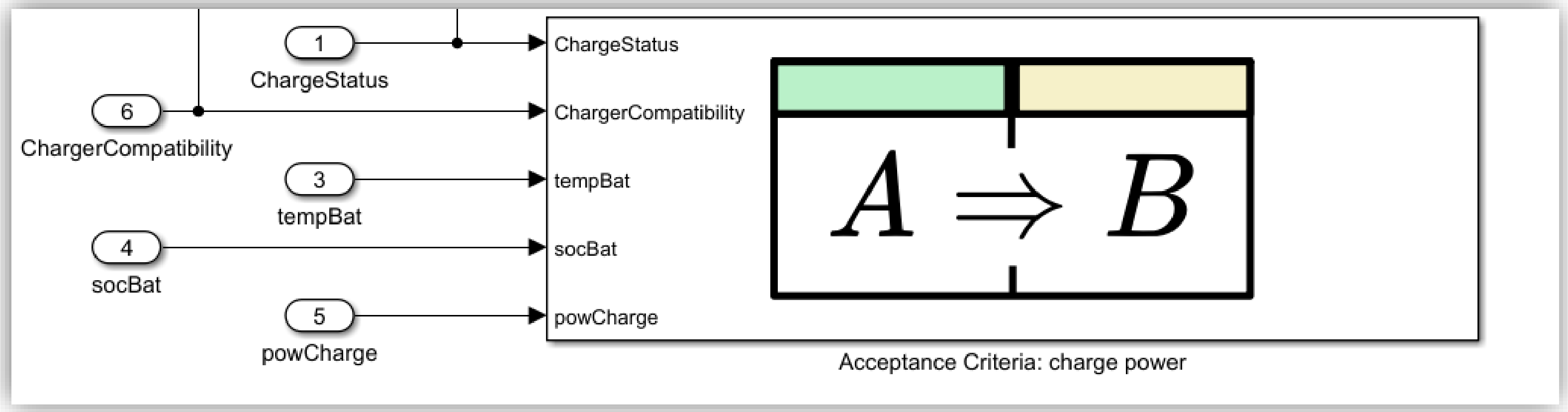
BDD and TDD lifecycles incorporating the benefits of MBD



C: Write System Test Cases

Create test cases for system behavior: Formalize Requirements

- Test-Case: System Under Test + Input Stimuli (Scenario) + Assessments
- Formalize requirements (acceptance criteria) == Assessments
- Use, for example, a “Requirements Table” in Simulink



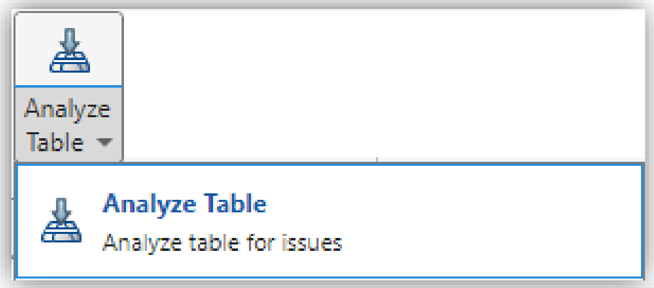
Requirements | Assumptions

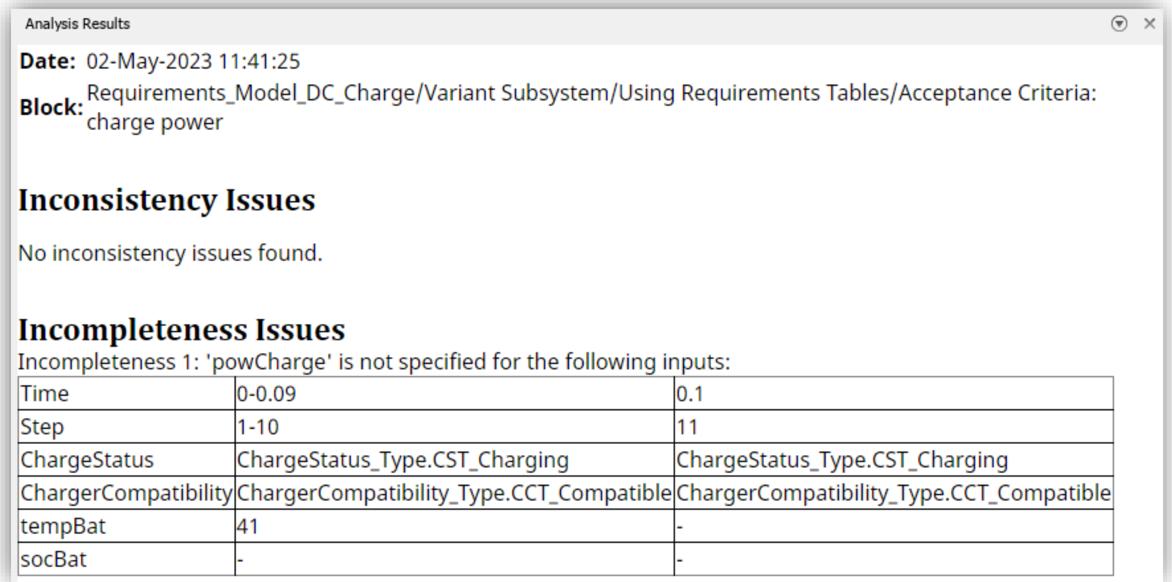
Index	Summary	Precondition				Duration	Postcondition
		ChargerCompatibility	socBat	tempBat	ChargeStatus		powCharge
1	Acceptance Criteria: charge power	CCT_Compatible	[5,50]	[10,40]	CST_Charging	0.1	> 300

A: Analyze System Requirements

Analyze Requirements for Complete and Consistency

- Requirements Tables assist you to analyze for completeness and consistency using advanced formal methods





Analysis Results

Date: 02-May-2023 11:41:25

Block: Requirements_Model_DC_Charge/Variant Subsystem/Using Requirements Tables/Acceptance Criteria: charge power

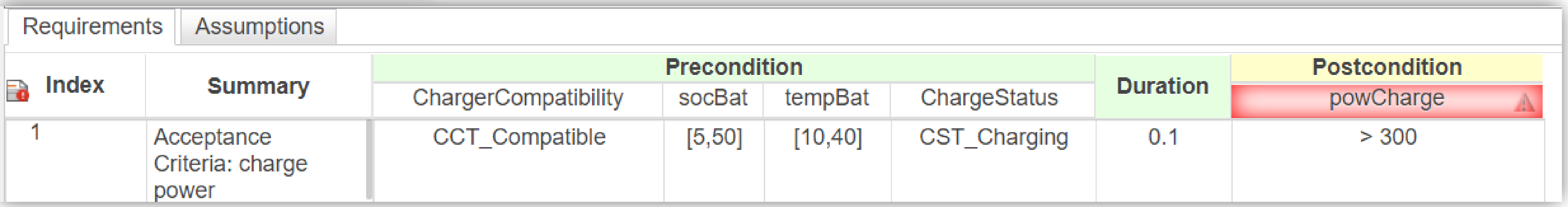
Inconsistency Issues

No inconsistency issues found.

Incompleteness Issues

Incompleteness 1: 'powCharge' is not specified for the following inputs:

Time	0-0.09	0.1
Step	1-10	11
ChargeStatus	ChargeStatus_Type.CST_Charging	ChargeStatus_Type.CST_Charging
ChargerCompatibility	ChargerCompatibility_Type.CCT_Compatible	ChargerCompatibility_Type.CCT_Compatible
tempBat	41	-
socBat	-	-

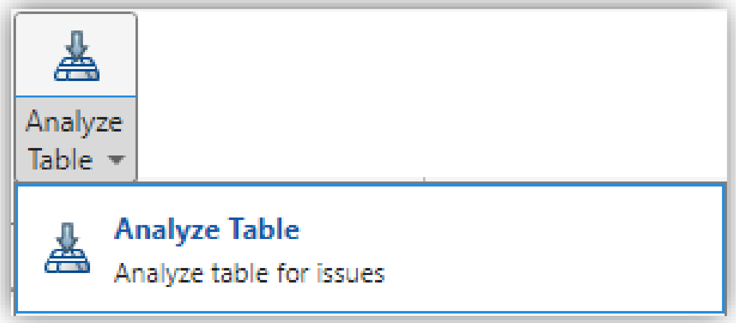


Requirements | Assumptions

Index	Summary	Precondition				Duration	Postcondition
		ChargerCompatibility	socBat	tempBat	ChargeStatus		powCharge
1	Acceptance Criteria: charge power	CCT_Compatible	[5,50]	[10,40]	CST_Charging	0.1	> 300

A: Analyze System Requirements
Analyze Requirements for Complete and Consistency

- Requirements Table needs to be complete to be used in simulation as Assessments



Analysis Results

Date: 02-May-2023 11:36:43

Block: Requirements_Model_DC_Charge/Variant Subsystem/Using Requirements Tables/Acceptance Criteria: charge power

Inconsistency Issues

No inconsistency issues found.

Incompleteness Issues

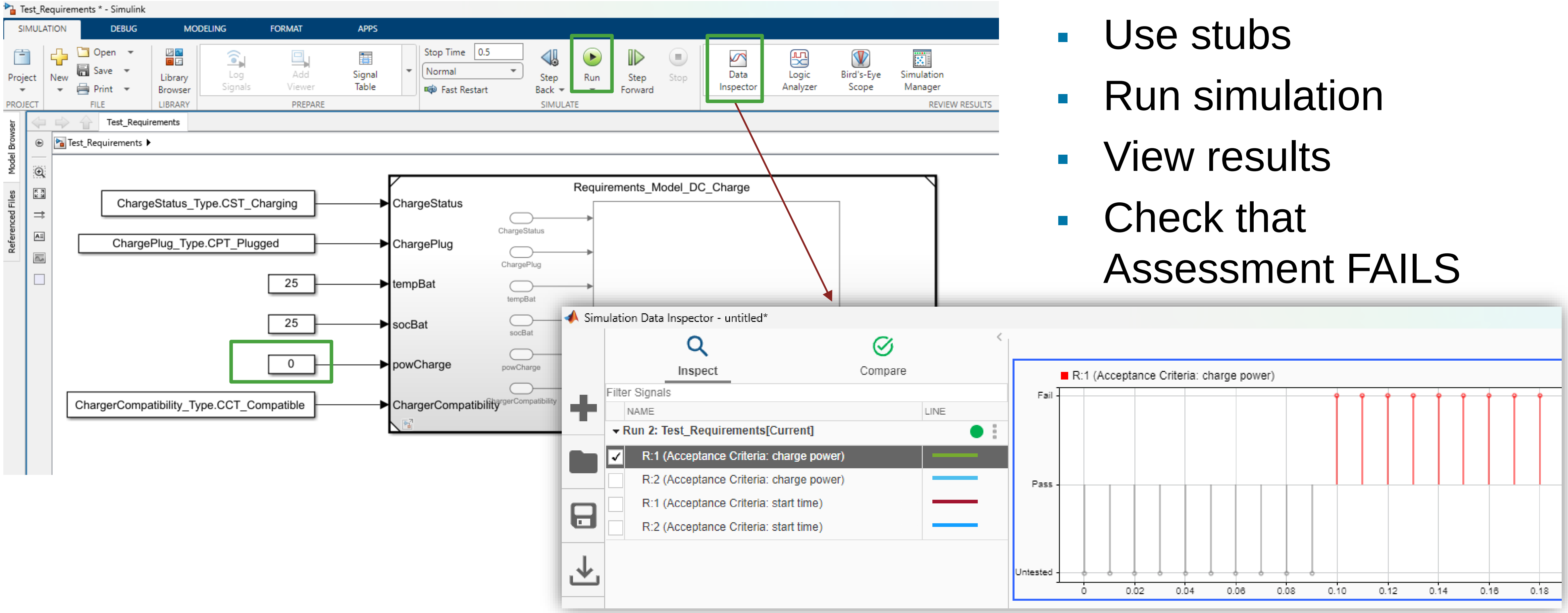
No incompleteness issues found.

Requirements | Assumptions

Index	Summary	Precondition				Duration	Postcondition
		ChargerCompatibility	socBat	tempBat	ChargeStatus		powCharge
1	Acceptance Criteria: charge power	CCT_Compatible	[5,50]	[10,40]	CST_Charging	0.1	> 300
2 D	Don't care	Else					[-2000,2000]

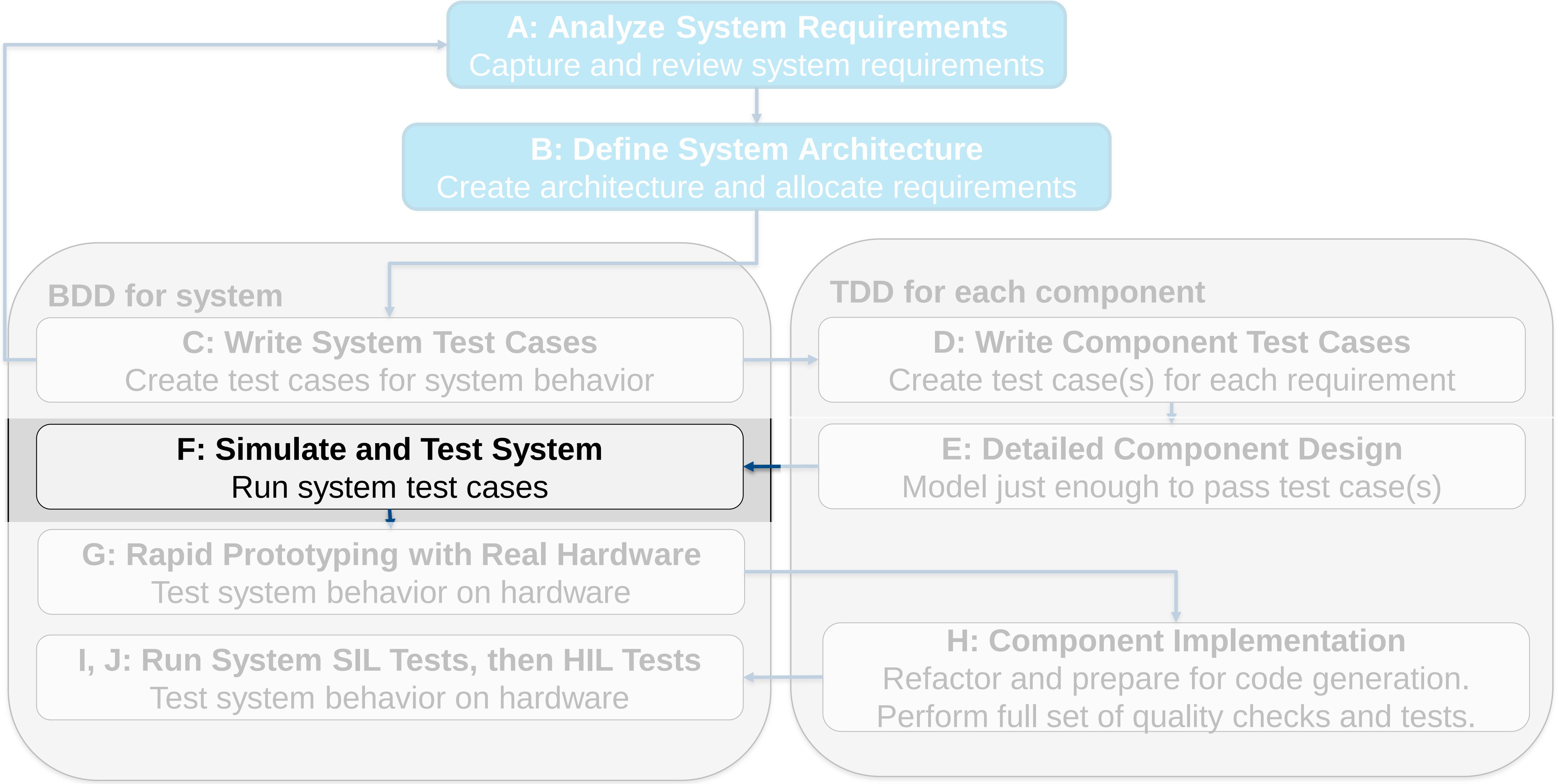
C: Write System Test Cases

Check that Assessments FAIL as expected



- Use stubs
- Run simulation
- View results
- Check that Assessment FAILS

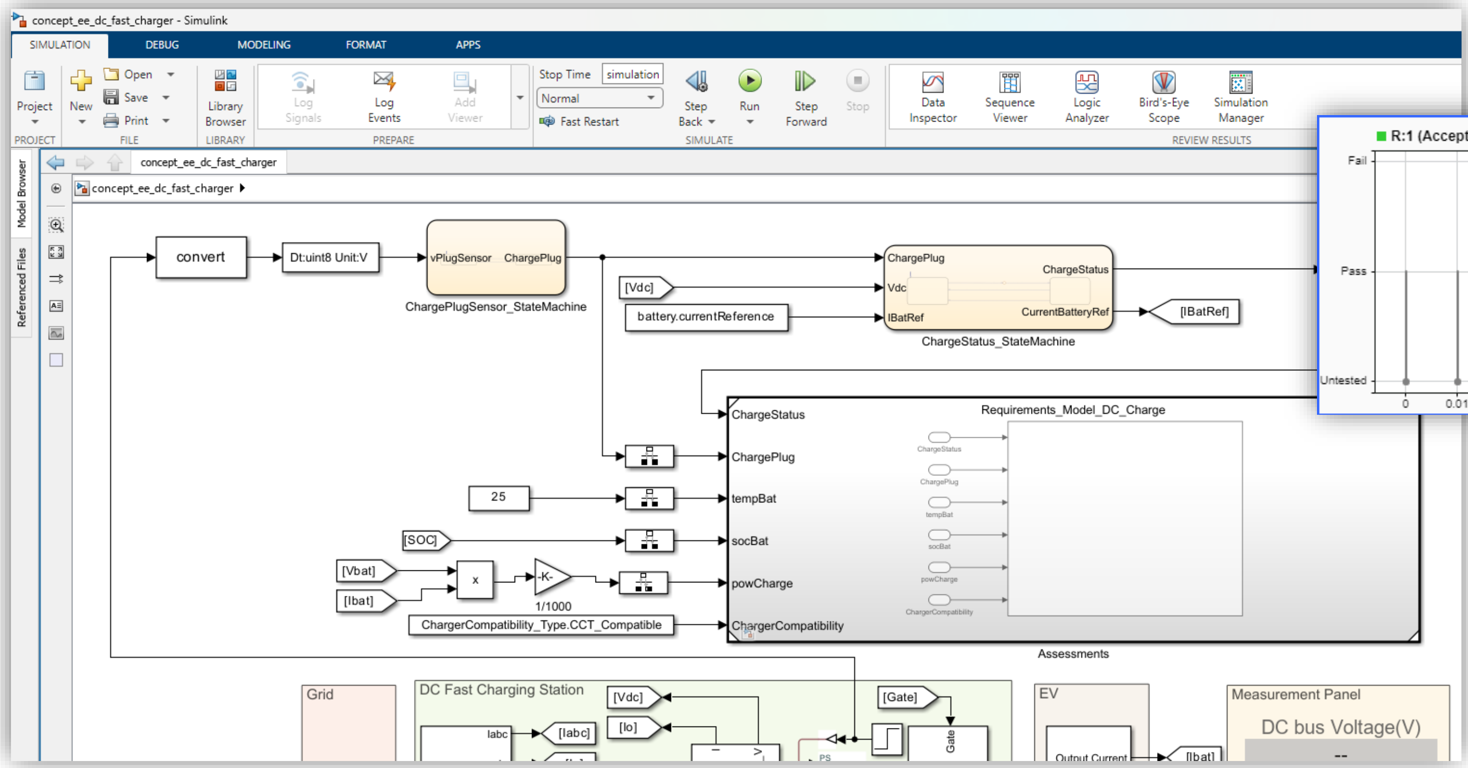
BDD and TDD lifecycles incorporating the benefits of MBD

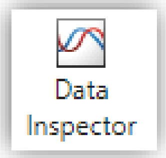


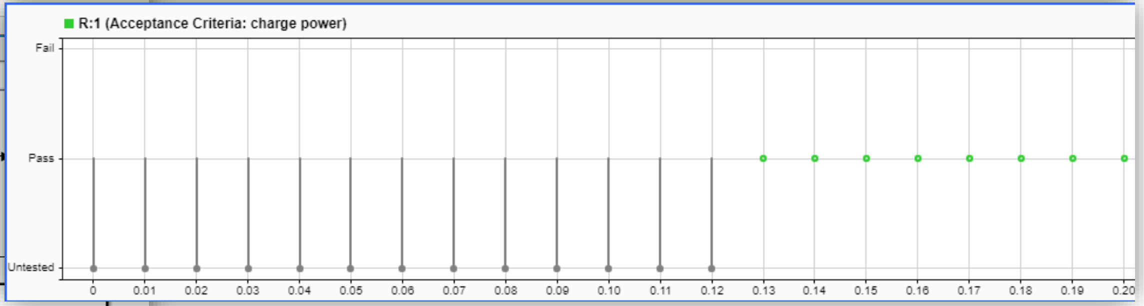
F: Simulate and Test System

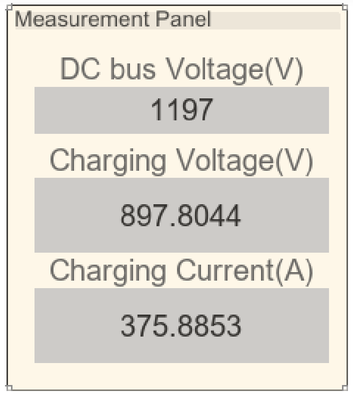
Run system test cases, Check that Assessments pass as expected

- Add mocking logic and stubs to virtual prototype
- Run Simulation including Assessments, view results



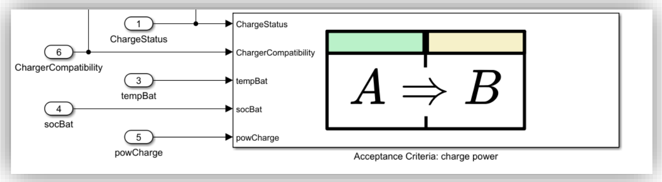


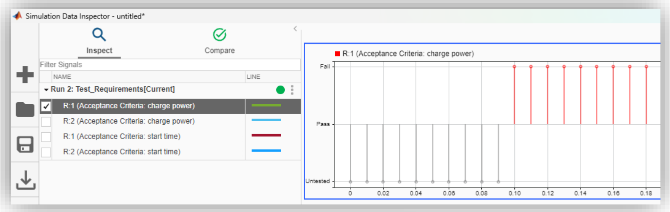


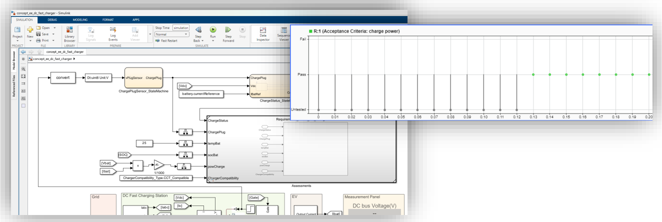


Conclusions

- Model-Based Design fits very well with BDD and TDD
- Refine requirements –
  make them complete, consistent, correct, testable
- Follow test first approach –
  Use test-cases, catching missing or wrong functionality
- Use virtual prototypes and test-benches
  to validate requirements
- Avoid unnecessary work –
  Gain confidence in requirements, architecture, and designs before implementation







Thank you!

For more information on this topic, please visit this page: