INCOSE

ASIA-OCEANIA SYSTEMS ENGINEERING CONFERENCE

# Quantifying and Managing System Complexity with a Model-Based Impact Analysis Engine

## A Case Study in 2-Wheeler EVs

INCOSE AOSEC · 2025

**AUTHORS**

Sankalp Kumar, and Prerana Kulkarni

**PUBLICATION DATE**

July, 2026

**SUBMISSION #**

63

INCOSE PUBLICATIONS

# Quantifying and Managing System Complexity with a Model-Based Impact Analysis Engine: A Case Study in 2-Wheeler EVs

**Sankalp Kumar**
**Intelligent Systems Department**
**Bajaj Auto Technology Limited**
**Pune, India, 411044**
**withsankalpkumar@gmail.com**

**Prerana Kulkarni**
**Intelligent Systems Department**
**Bajaj Auto Technology Limited**
**Pune, India, 411044**
**pvkulkarni240@gmail.com**



## Abstract

Complex engineered systems evolve rapidly, and small requirement changes can cascade across architectures and behaviors, yet practitioners lack a concise, repeatable way to quantify those ripples. This paper presents a model-based impact-analysis framework built on SysML v1 artifacts and an executable Python pipeline. We formalize a compact metric set — function and flow counts, functional density, Dependency Structure Matrix (DSM) coupling measures, behavioral cyclomatic complexity for state machines, and provide algorithms to extract them from SysML v1 models - Functional Architecture (FA), Logical Architecture (LA), State Machine Diagram (SMD), Sequence Diagram (SD) - and ancillary trace matrices. The framework compares baseline and post-change model snapshots, computes delta profiles, generates a Dependency Structure Matrix, and ranks affected elements to guide testing, verification, and architecture decisions. This paper demonstrates the approach in the case of a 2-wheeler electric vehicle: replacing mechanical key entry with a wireless keyfob and dual-authentication design. Quantitative results show targeted increases in functional flows and functional density, among others, and identify coupling hotspots (e.g., CAN bus) that concentrate risk and verification effort. The implementation of this engine makes use of Python scripts and SysML v1 models. The artifacts and tooling emphasize portability to common MBSE toolchains. The paper concludes with sensitivity analyses, limitations tied to model granularity, and a roadmap for integrating the engine into industrial MBSE workflows.

## Keywords

Model-Based Systems Engineering (MBSE), SysML v1, Complexity Metrics, Impact Analysis, Traceability, Electric Vehicles, Digital Engineering.

# I. Introduction

Today, engineering systems, particularly in domains such as automotive electrification, autonomous mobility, and connected infrastructure, are growing in functional capability and interdependence. This growth delivers new value, but simultaneously increases the probability that a single requirement change will produce far-reaching and often unanticipated consequences across architecture, behavior, and verification scope. Systems engineers therefore face two pressing needs: (1) objective, repeatable measures of system complexity that are tied to concrete models, and (2) an operational process that translates those measures into actionable impact assessments for design, test, and risk mitigation. Model-Based Systems Engineering (MBSE) and SysML v1 provide a natural substrate to meet these needs: functional decompositions, logical allocations, state machines, and interaction sequences can all be captured as authoritative artifacts. However,

current practice often stops at visualization and informal traceability. Prior research has proposed complexity taxonomies and single-representation metrics (e.g., DSM analysis, McCabe cyclomatic counts, and function-count heuristics) (Browning, 2001; McCabe, 1976; Sheard & Mostashari, 2010; Steward, 1981) but lacks an integrated, model-driven pipeline that (a) extracts quantitative metrics across FA/LA/SMD/SD artifacts, (b) computes baseline vs. post-change deltas, and (c) produces ranked impact summaries that directly inform verification and architectural choices.

This paper addresses that gap. We present a model-based impact-analysis framework built on SysML v1 artifacts and an executable Python toolchain. Our approach formalizes a compact set of metrics: function and flow counts, functional density, DSM coupling measures, and behavioral cyclomatic complexity and defines deterministic extraction and delta-scoring algorithms. Contributions are threefold: (1) a formal, multi-representation metric set with precise extraction rules from SysML v1 models; (2) an automated pipeline (model export → Python ingestion → metric computation → DSM/trace difference → impact ranking) validated on an industrially-relevant 2-wheeler EV case; and (3) empirical insights into how a single requirement change (mechanical key → key-fob dual authentication) quantitatively affects complexity, testing scope, and coupling hotspots.

The paper is organized as follows. Firstly, the paper situates our work in related literature. Secondly, it defines metrics and formal methods for extraction. Thirdly, it describes the toolchain and implementation. The paper, then, presents the EV case study and results. Later, industrial utility, and limitations for engineering practice are discussed, and finally the paper concludes with a roadmap for integration into industrial MBSE workflows.

# II. Related Work

Traceability and model-based impact analysis are well-established research and industrial practices (INCOSE, 2023; Object Management Group (OMG), 2012). Prior work has studied complexity metrics for engineering models (McCabe, 1976; Potts et al., 2020; Sheard & Mostashari, 2010) , and DSM-based analyses are standard for interface concentration detection (Browning, 2001; Steward, 1981; Yassine et al., 2003). Our framework synthesizes these strands: we compute simple, interpretable metrics from SysML XMI, propagate changes via model linkages (traceability), and use DSM analyses to highlight hotspots. Unlike many heavy commercial offerings, this method emphasizes portability and repeatability via open tooling (Papyrus and Python) and XMI as a lingua franca.

# III. Our Approach

This section situates the proposed impact-analysis framework within three relevant research strands: (a) complexity measurement in engineered systems, (b) DSM and impact-analysis methods, and (c) MBSE tooling and model-driven metrics.

**A. Complexity measurement:**

Foundational work on behavioral complexity (McCabe, 1976) introduced cyclomatic complexity as a path-based measure of behavioral branching, that remains widely used for software and state-based models; it provides a rigorous path-centric view of verification burden and motivates our use of cyclomatic counts on state-machine diagrams. Broader typologies and taxonomies (e.g., Sheard & Mostashari) classify sources of complexity in systems engineering—interfaces, interactions, emergent behavior—and argue for multi-dimensional measures rather than single-number heuristics (Sheard & Mostashari, 2010). More recent empirical studies examine complexity in engineered systems from multi-model perspectives and highlight the need for metrics that map directly to artifacts engineers already author (Potts et al., 2020).

**B. Dependency analysis and DSMs:**

The Design Structure Matrix (DSM) literature (Steward, 1981) and later applications in product architecture research show how dependency concentration predicts propagation of change and integration risk. DSM methods have been adapted to software, hardware, and socio-technical contexts to identify hotspots and schedule risk (Browning, 2001; Eppinger & Browning, 2016; Yassine et al., 2003). Our use of DSM maps and degree-based coupling scores follows this tradition but extends it by linking DSM entries directly to SysML model elements and to computed metric deltas.

**C. Model-based impact and MBSE tooling:**

Prior MBSE efforts demonstrate the value of formal models (FA/LA/SMD/SD) for traceability and verification (INCOSE, 2023; Object Management Group

(OMG), 2012); tools such as Capella, Papyrus, and other SysML workbenches support these representations and manual trace analyses. Industry solutions and research prototypes for impact analysis exist (e.g., vendor/consultancy tools that compute trace-based change lists), but they often operate within a single representation or require heavy manual configuration. There is limited published work that (1) systematically extracts the same set of quantitative metrics across FA, LA, SMD and SD, (2) computes baseline $\rightarrow$ post-change deltas, and (3) produces ranked, explainable impact scores usable for test-planning and architectural decisions.

**D. Gap and our position:**

In summary, the literature provides strong foundations—cyclomatic theory for behavior, DSM for coupling, and MBSE for authoritative artifacts—but lacks a compact, reproducible pipeline that unifies these pieces into an automated, multi-model impact engine. Our contribution fills that gap by (a) formalizing a small, well-defined metric set across commonly used SysML artifacts, (b) automating extraction with a portable Python toolchain, and (c) combining delta analysis with DSM/topology metrics to produce ranked, actionable impact summaries.

# IV. Definitions and Formal Metrics

**A. Definitions:**

*System* - a set of interacting elements organized to perform one or more functions.

*Functional Architecture (FA)* - hierarchical set of Functions and Flows capturing what the system does. A Function is an atomic unit of behavior at the chosen granularity. A Flow is a directed logical dependency (information, control, power) between functions.

*Logical Architecture/Physical Architecture (LA/PA)* - Blocks (components) and Connectors that realize functions; LA is domain-independent allocation; PA is component/implementation allocation.

*State-Machine Diagram (SMD)* - a set of States and Transitions describing behavioral modes and guarded changes. Only atomic (leaf) states are counted as states.

*Sequence Diagram (SD)* - ordered lifelines and messages showing interaction sequences between components/actors.

*Traceability Matrix* - mapping between Requirements and Model elements (Functions, Blocks, States, Messages).

*Dependency Structure Matrix (DSM)* - N×N matrix representing directed dependencies among components/blocks.

**B. Granularity Rule:**

**Baseline granularity:** The lowest decomposition level was chosen such that further splitting of a function (or state) does not change the engineering decisions the metric should influence. Practically: we used leaf-level functions that map to a single test case or work package; used atomic states (no internal substates) for SMD metrics.

**C. Formal Metrics:**

***Function Count***

$$N_F = \{f \mid f \in \text{FA\_leaf\_functions}\}$$

Unit: count.

Interpretation: estimate of functional decomposition.

***Flow Count***

$$I_F = \{\phi \mid \phi : f_i \rightarrow f_j\}$$

Unit: count.

Interpretation: number of directed logical interactions.

***Functional Density (per function) - Normalised interaction intensity***

$$D_F = I_F / N_F$$

Unit: flows per function.

Interpretation: higher $D_F$ means more interactions per function (greater coordination cost).

***State Count and Transition Count***

$$S = \mid atomic\ states \mid,\ T = \mid transitions \mid$$

Units: counts.

***Cyclomatic Complexity (State Machine) - For a state graph***

$$M = T - S + 1$$

Unit: count of independent execution paths.

Interpretation: minimal number of test cases to cover all paths.

***Lifeline Count (Sequence Diagram)***

$L$ = number of participants.

***Message Count (Sequence Diagram)***

$M_s$ = number of messages.

***Interaction Depth (Sequence Diagram)***

$D$ = maximum parallel depth.

***DSM / Coupling Metrics***

Let A be the adjacency matrix (the Dependency Structure Matrix (DSM) that is of interest) among *n* components (blocks), where $A_{ij}$ = 1 if *i* depends on *j* (optionally weighted).

*Out-degree of* ***i****:*

$$d_i^{out} = \sum_j A_{ij}$$

Interpretation: how many components *i* depends on. Change originating at node *i* may propagate outward.

*In-degree of* ***i****:*

$$d_i^{in} = \sum_j A_{ji}$$

Interpretation: how many components depend on *i*. Changes elsewhere may converge on node *i*.

*Global Density:*

$$\rho = (\sum_{i,\,j} A_{ij}) \,/\, n(n-1)$$

Interpretation: tells that nodes with high total degree are change propagation hotspots (high risk).

***Traceability Coverage***

Given R requirements and E elements:

$$C_{trace} = ((R,E)\,|R\,traces\,to\,E\,)\,/\,R$$

Unit: fraction (0 - 1).

Interpretation: fraction of requirements with model.

**D. Delta & Impact scoring**

We compare **baseline** (subscript 0) and **updated** (subscript 1) snapshots and compute deltas:

$$\Delta X = X_1 - X_0$$

$for\ metrics\ X\varepsilon\{N_F, I_F, D_F, S, T, M, L, M_s\}.$

Report absolute and percent delta:

$$\%\Delta X = 100 * \Delta X / X_0\ (when\ X_0 \neq 0)$$

Interpretation: rank elements by delta change; high scores -> prioritize test, review, or refactor.

Rationale for deltas: function/flow deltas measure structural changes; cyclomatic delta measures behavioral branching changes.

**E. Units, Thresholds, & Practical Defaults**

***Suggested Alert Thresholds:***

- $\%\Delta N_F > 10\%$ means moderate structural growth.
- $\%\Delta I_F > 15\%$ means increased coordination risk.
- $\Delta M \geq 3$ means behavioral complexity warranting expanded testing.
- Node degree $d_i \geq 0.5 \times \max(d)$ means hotspot (high coupling).

The thresholds mentioned here are empirical and should be calibrated as per one's organization's products and practices.

**F. Assumptions & Caveats**

- **Model fidelity:** metrics reflect the model as authored which means that inaccurate models yield misleading metrics.
- **Granularity sensitivity:** metrics change with decomposition level so, a documented baseline is required.
- **Semantic mapping:** behavioral semantics (e.g., composite states, orthogonal regions) must be reduced to atomic states per baseline rule before counting.
- **Normalization:** normalized scores require consistent normalization method.

The selected metrics and extraction rules are grounded in established behavioral, structural, and systems-complexity literature and are intentionally conservative to preserve interpretability and reproducibility (McCabe, 1976; Potts et al., 2020; Sheard & Mostashari, 2010; Steward, 1981).

# V. Toolchain & Python Implementation

DSM analysis, and model traversal were implemented using established Python libraries (Hagberg et al., 2008; Kolovos et al., n.d.).

We implemented a pipeline that consumes SysML v1 XMI exports and produces automated metrics, DSMs and ranked impact lists. The pipeline stages are: (1) Ingest XMI exports of system models. (2) Parse XMI exports with python. (3) Metrics (compute counts, densities, cyclomatic), (4) DSM & Trace (build matri-

ces and compute coupling), (5) Delta & Impact (compare snapshots and score elements), and (6) Visualize and Report (JSON, CSV, charts).

### A. Inputs

Authoritative SysML v1 model files exported as XMI from the modeling tool (Papyrus, Cameo, Enterprise Architect, Capella). XMI is required because it preserves element types, stereotypes and behavioral structure (states, transitions, guards) better than diagram images.

### B. Parsing XMI

Most SysML v1 XMI files produced by Papyrus/Cameo can be read using python. Models' XMI exports expose metamodel classes, relationships and applied stereotypes reliably. This preserves state ownership, transitions, and trace links.

### C. Visualization and Reports

Calculated metrics are exported as .csv and .json files. DSM trace, and charts for comparison between initial system models and models after requirement changes ingestion into system are also obtained.

*Key advantages:* robust traversal, direct access to SysML metaclasses, easier mapping of guards and ports.

## VI. Case Study: 2-Wheeler EV

### A. System context and change request

This case study examines a representative subsystem of a 2-wheeler battery-operated electric vehicle (EV) that is typical of current production and near-production platforms. In the baseline configuration, vehicle access and enablement are achieved through a purely mechanical handlebar lock–unlock mechanism actuated using a physical key. The baseline system behavior and architecture therefore reflect a low-coupling, low-branching entry mechanism with minimal electronic authentication logic.

The core system capabilities modeled for this study include vehicle entry, vehicle mobility (forward drive, reverse drive, and limp-home operation), battery charging, ECU software flashing, and telemetry/data analytics. These capabilities were selected because they represent a realistic cross-section of functional, logical, and behavioral concerns in a modern EV, and because they span both safety-critical and user-facing behaviors.

The evaluated change request replaces the mechanical key entry with a wireless keyfob-based access flow that introduces dual authentication. The updated design requires: (1) successful detection and authentication of a wireless keyfob within a defined proximity threshold ($\leq 10\ m$), and (2) subsequent authentication of the Electronic Handle Lock (EHL) mediated by the Supervisory Control Unit over the CAN bus. To address security and misuse concerns, the system introduces an attempt counter; after three consecutive failed authentication attempts, the system must notify the vehicle owner via the telematics channel.

**Objective:** quantify how this change affects structural and behavioral complexity (FA, LA, SMD, SD), identify propagation hotspots, and provide ranked, actionable impact guidance for verification and architecture work.

### B. Models used and snapshot description

We used authoritative SysML v1 artifacts (exported as XMI) as inputs to the pipeline.

- **FA (Functional Architecture)** — baseline FA (Figure 1) and updated FA (Figure 2). Leaf functions were chosen as per the granularity rule: each leaf maps to a single testable activity. The updated FA adds functions such as fob detection, fob authentication, EHL authentication, Increment Fail Count, and Notify Owner of vehicle.
- **LA (Logical Architecture)** — baseline LA (Figure 3) and updated LA (Figure 4). The updated LA introduces EHL-block with wireless communication and TCU (telemetry/owner notification) connectivity to GSM and CAN.
- **SMD (State Machine Diagram)** — baseline SMD (Figure 5) and updated SMD (Figure 6). Authentication sequence was introduced as a distinct submachine with fob authentication, EHL authentication, and an attempt counter state.
- **SD (Sequence Diagram)** — representative scenario (vehicle entry/startup): baseline SD (Figure 7). Updated SD (Figure 8) shows added lifeline TCU, alternative blocks for retries and owner notification.

### C. Metric extraction and summary results

The extraction pipeline was executed on both baseline and updated snapshots. Structural, behavioral, and interaction metrics were computed for each representation, and deltas were derived by direct comparison. Table 1 summarizes the baseline values, updated values, and resulting changes used for

subsequent impact analysis.

| Metrics | Baseline | Updated | Δ | %Δ |
|---|---|---|---|---|
| Function Count | 30 | 33 | (+)3 | 10.0% |
| Flow Count | 17 | 21 | (+)4 | 23.5% |
| Functional Density | 0.567 | 0.636 | (+)0.069 | 12.2% |
| SMD States | 16 | 18 | (+)2 | 12.5% |
| SMD Transitions | 29 | 35 | (+)6 | 20.7% |
| Cyclomatic | 14 | 18 | (+)4 | 28.6% |
| SD Lifelines | 5 | 6 | (+)1 | 20.0% |
| SD Messages | 24 | 27 | (+)3 | 12.5% |

**Table 1. Baseline and Updated Metrics for Impact Analysis**

Interpretation: the update increases both structural (functions/flows) and behavioral (states/transitions/cyclomatic) complexity. The cyclomatic increase (+4, +28.6%) signals more independent execution paths requiring test coverage. Functional density grows 12.2%, indicating more interactions per function and thus higher coordination cost.

### D. DSM & traceability results

We built DSMs from LA connector sets and generated trace matrices between requirements and model elements.

**Interpretation:** high in/out degree nodes are propagation hotspots. CAN's high degree implies that changes touching it are likely to cascade; EHL & TCU are new contributors to this propagation potential.

### E. Ranked Outcomes

Top impacted items:

- **Electronic Handle Lock (EHL)** — highest impact due to direct function/flow additions and medium-to-high DSM centrality.
- **Telematics Unit (TCU)** — new owner-notification path and wireless dependency increases its impact score.
- **CAN Bus (CAN)** — unchanged structurally but highest centrality → significant propagation risk.
- **Battery Management System (BMS)** — minor coupling increases (battery checks during start/charging) produce secondary impact.

### F. Recommended Engineering Actions

From the ranked impacts and DSM insights we recommend, in order:

- **Focused verification** — Prioritize verification activities for Electronic Handle Lock (EHL) authentication behavior. The increase of four independent execution paths (cyclomatic +4) requires explicit enumeration and coverage to maintain behavioral completeness.
- **Security hardening** — Strengthen security across the updated entry architecture by reviewing firmware integrity mechanisms, code-signing practices, and authentication strength for EHL, keyfob, and telematics interfaces.
- **CAN regression tests** — Execute targeted regression testing for subsystems communicating over the CAN bus, focusing on interfaces affected by the introduction of EHL and TCU dependencies.
- **Telemetry & monitoring** — Enable runtime logging and telemetry capture for authentication failures, retry counters, and owner-notification events. These mechanisms support post-deployment validation, fault diagnosis, and assessment of alignment between modeled behavior and observed field operation.
- **Trace updates & test-case mapping** — Update the requirements traceability matrix to reflect new functions, states, and interfaces introduced by the keyless entry change. Map verification artifacts and test cases to impacted elements.

Each recommended action links directly to engine outputs: ranked list (priority), DSM hotspot (scope), and metric deltas (test effort estimate).

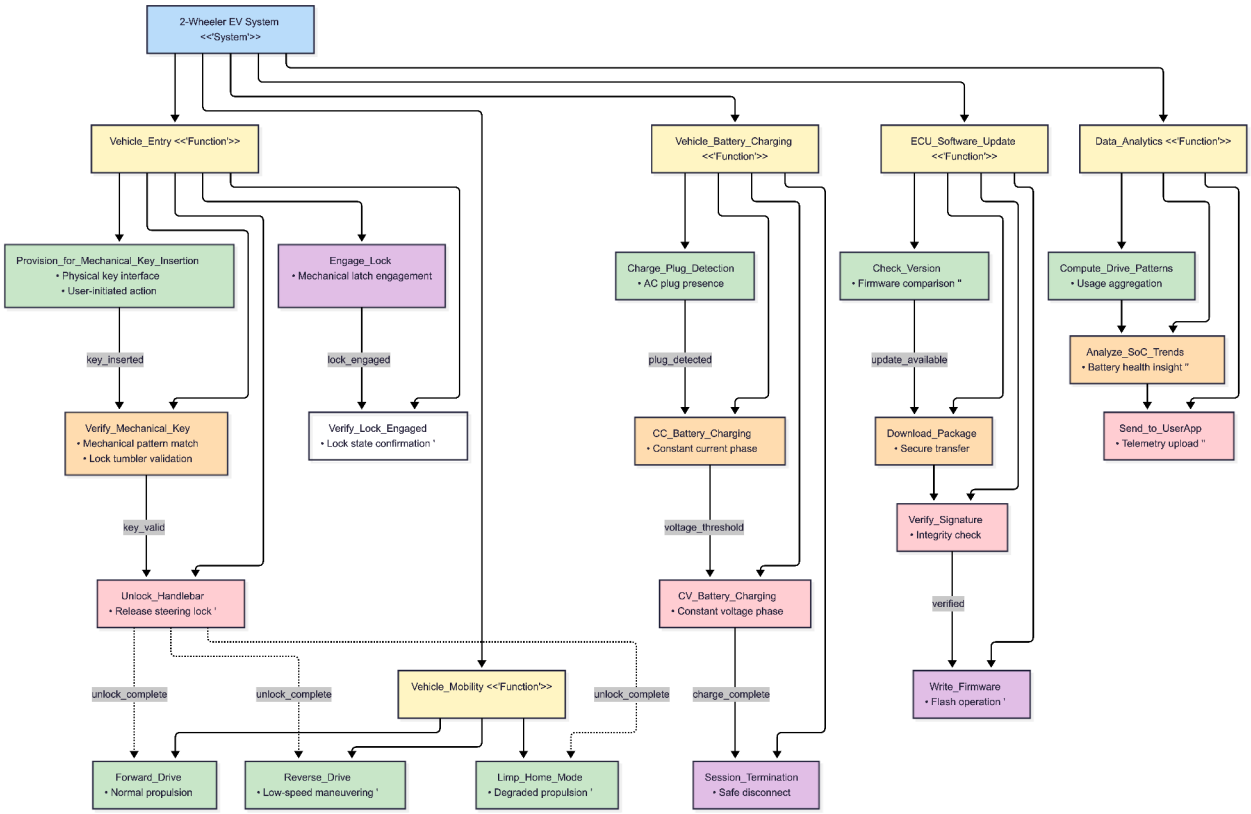


**Figure 1. Functional Architecture – Baseline.**

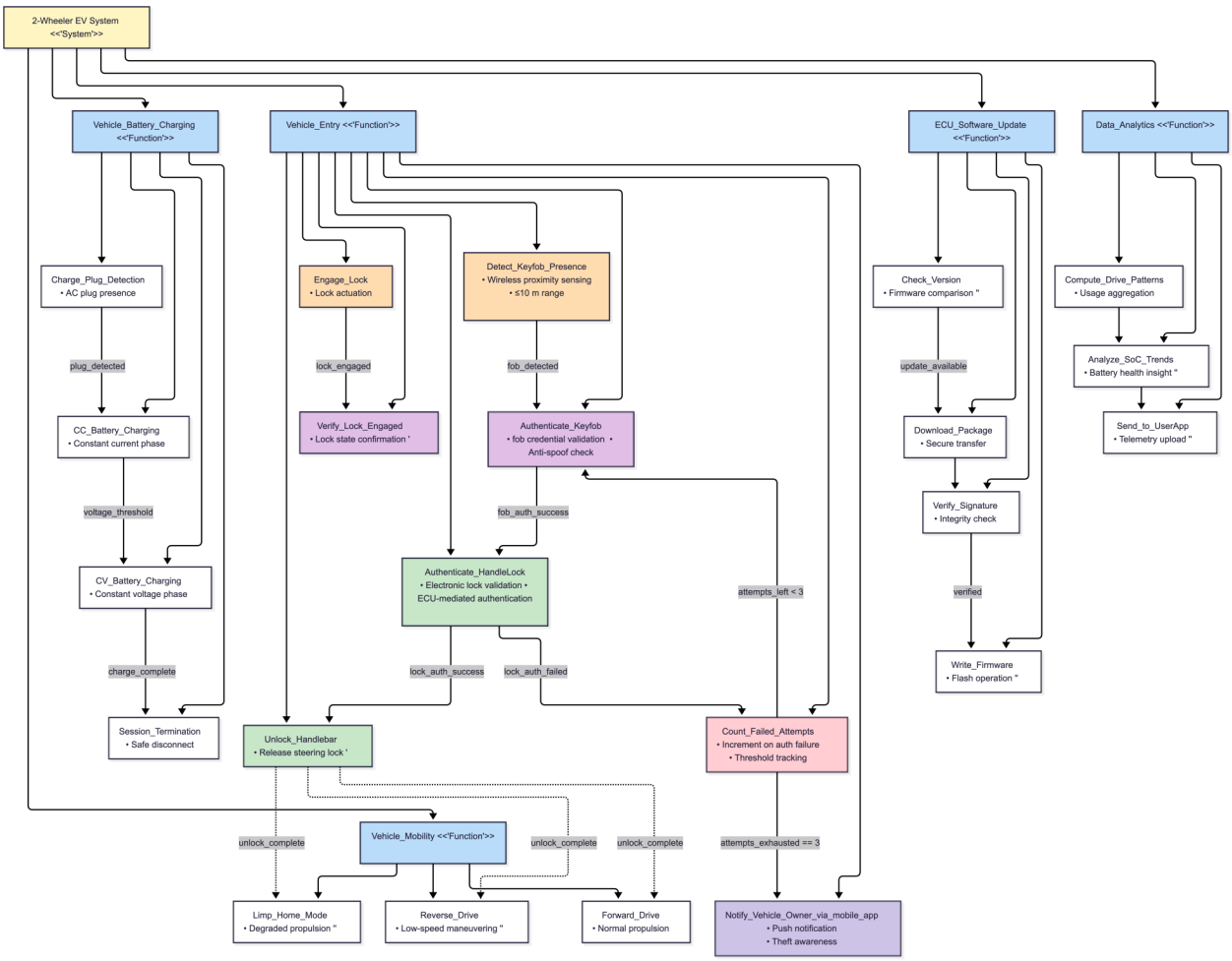


**Figure 2. Functional Architecture – Updated.**

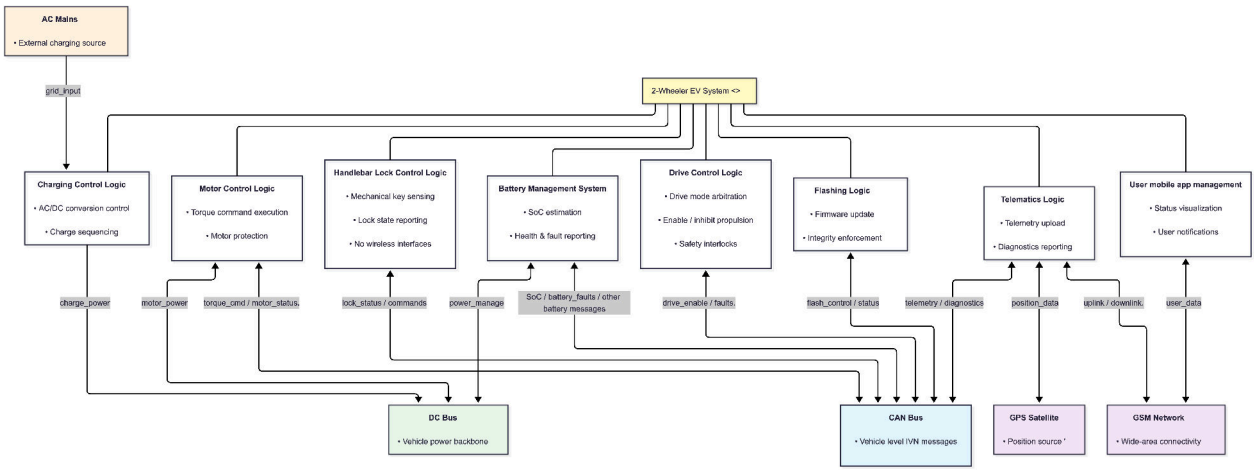


**Figure 3. Logical Architecture – Baseline.**

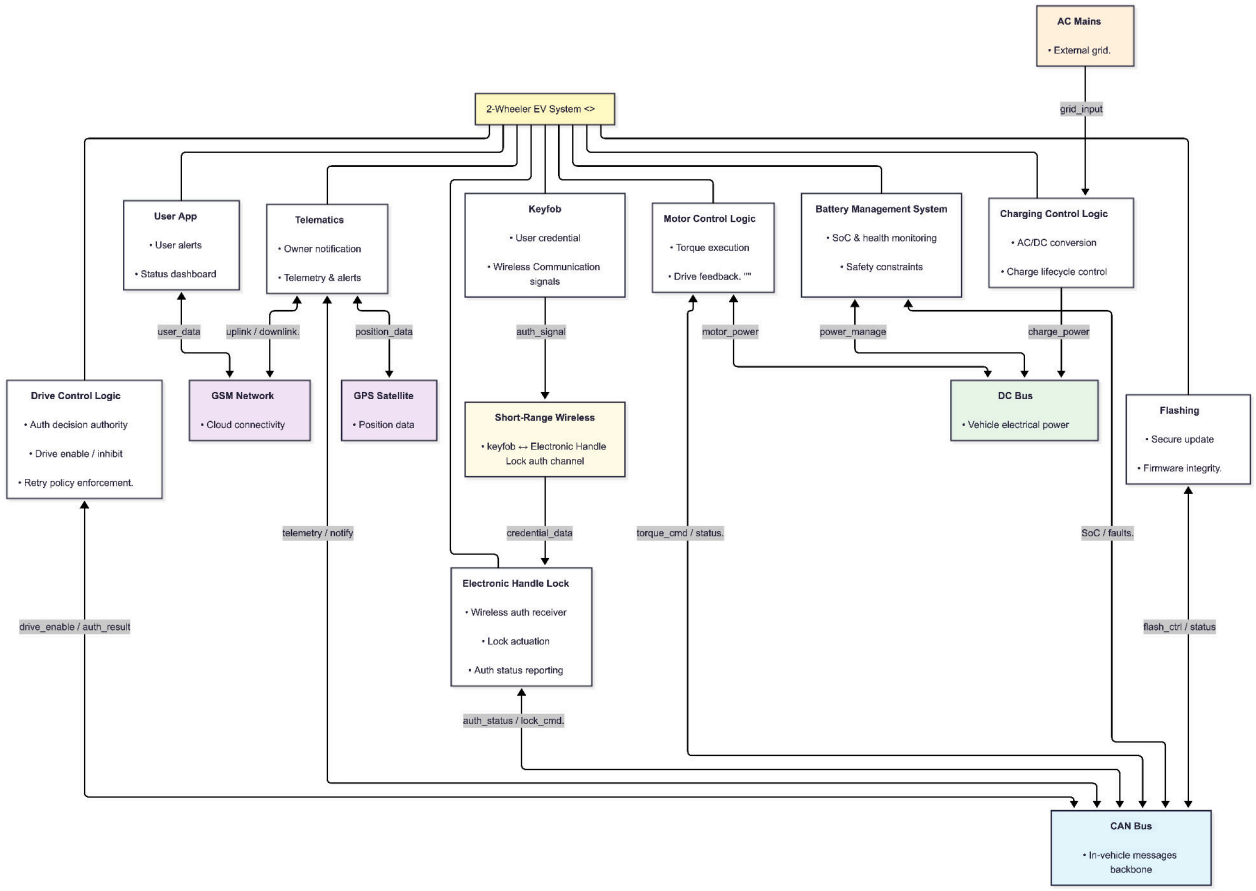


**Figure 4. Logical Architecture – Updated.**

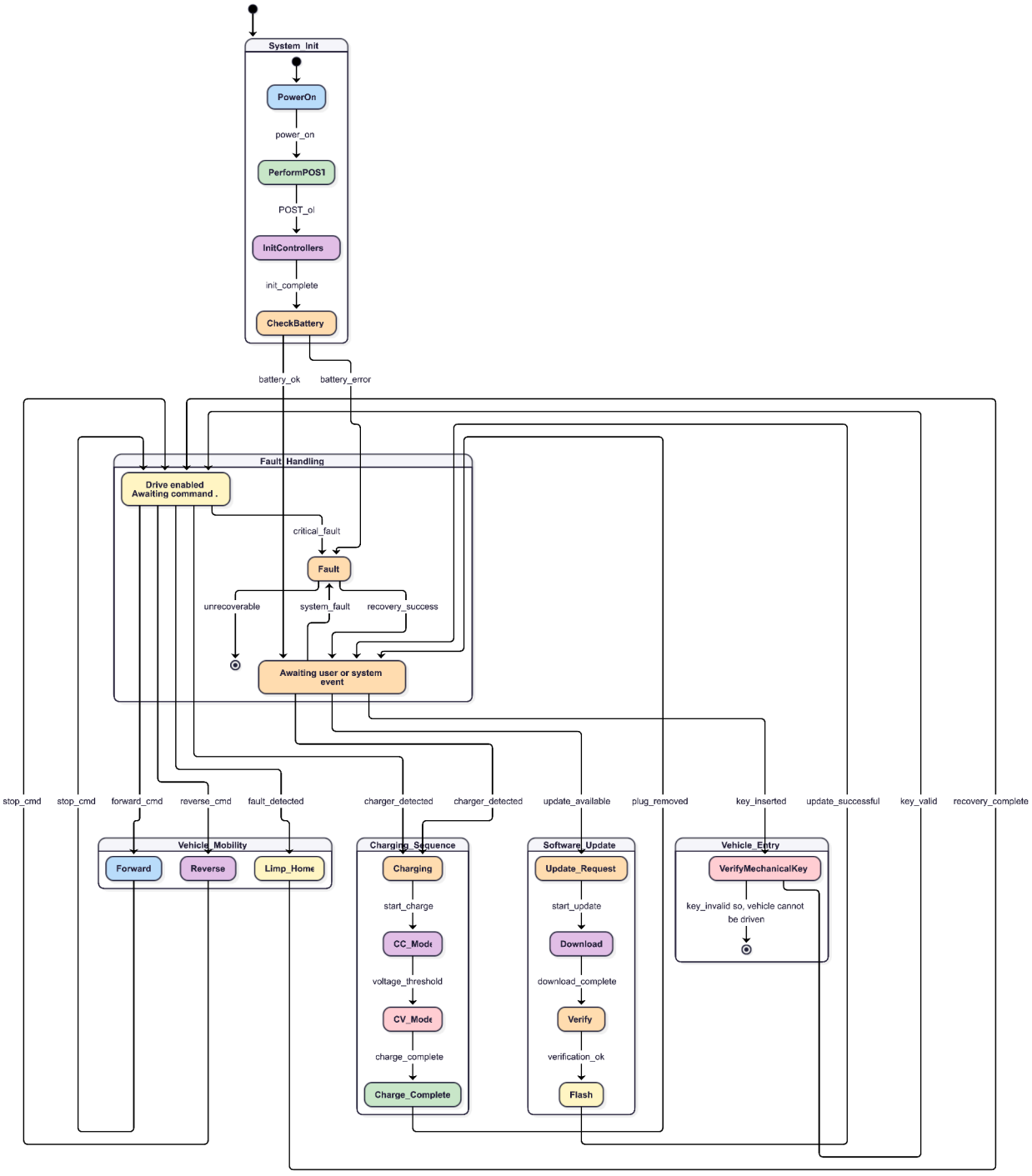


**Figure 5. State Machine Diagram – Baseline.**

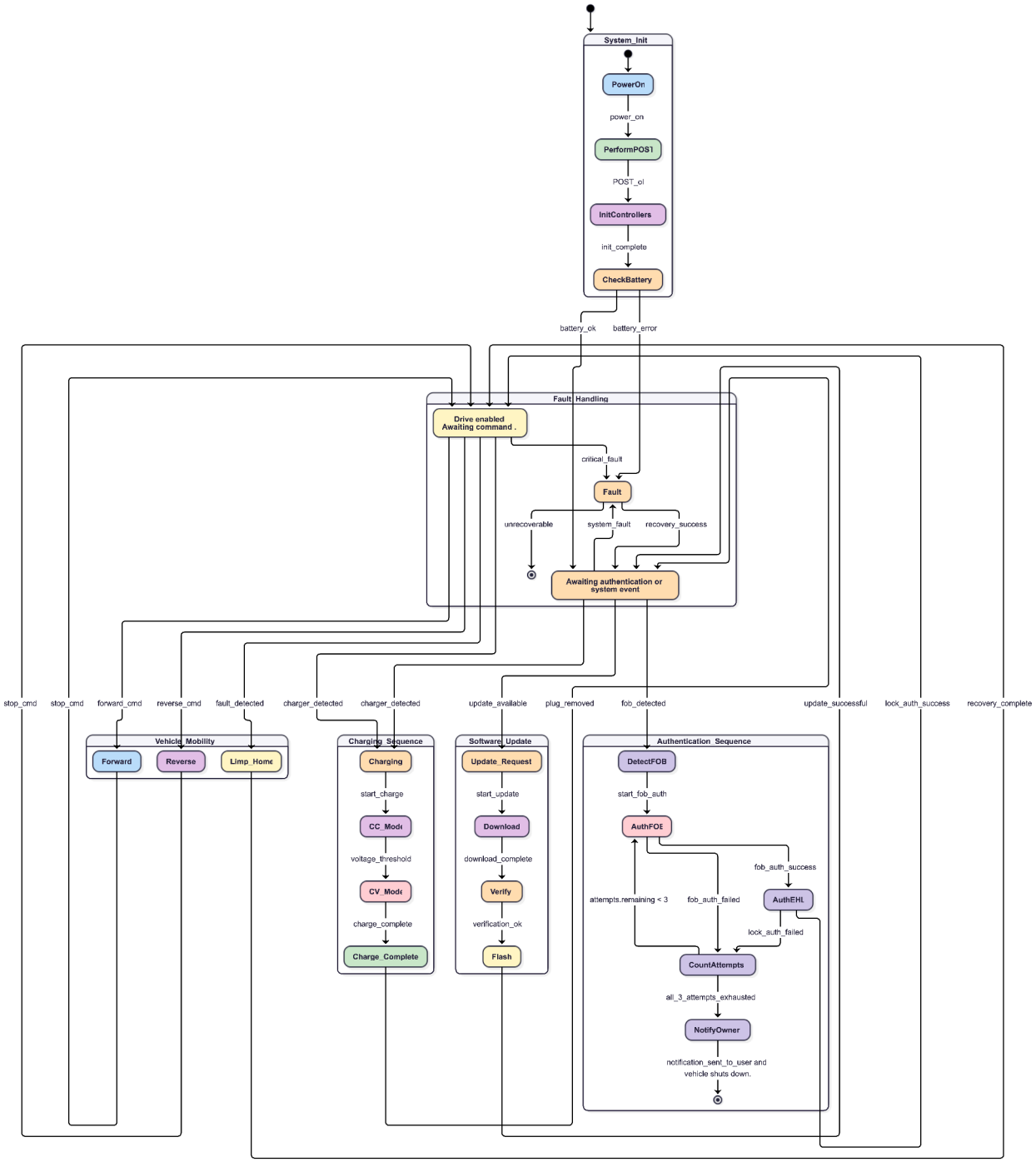


**Figure 6. State Machine Diagram – Updated.**

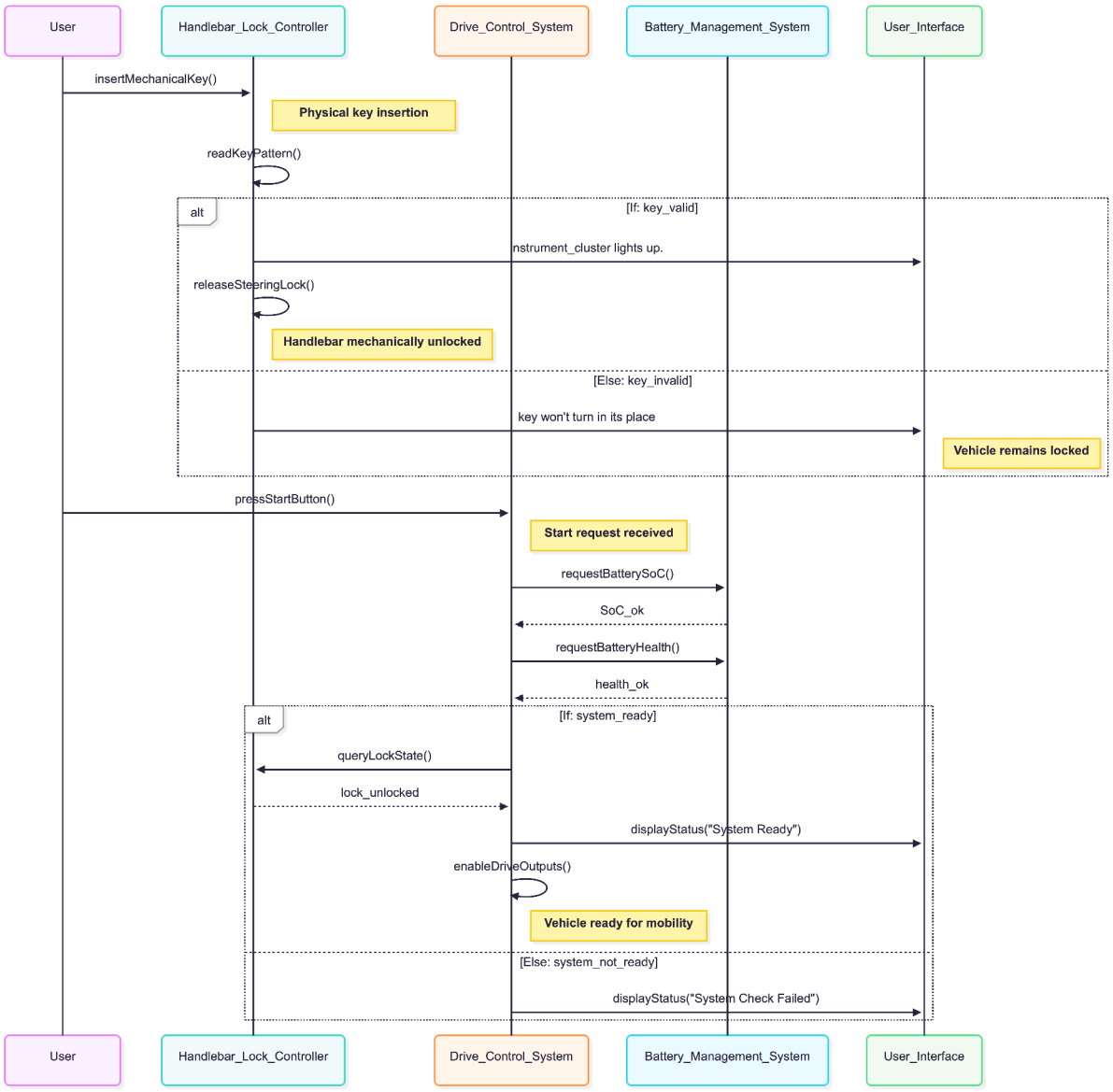


**Figure 7. Sequence Diagram – Baseline.**

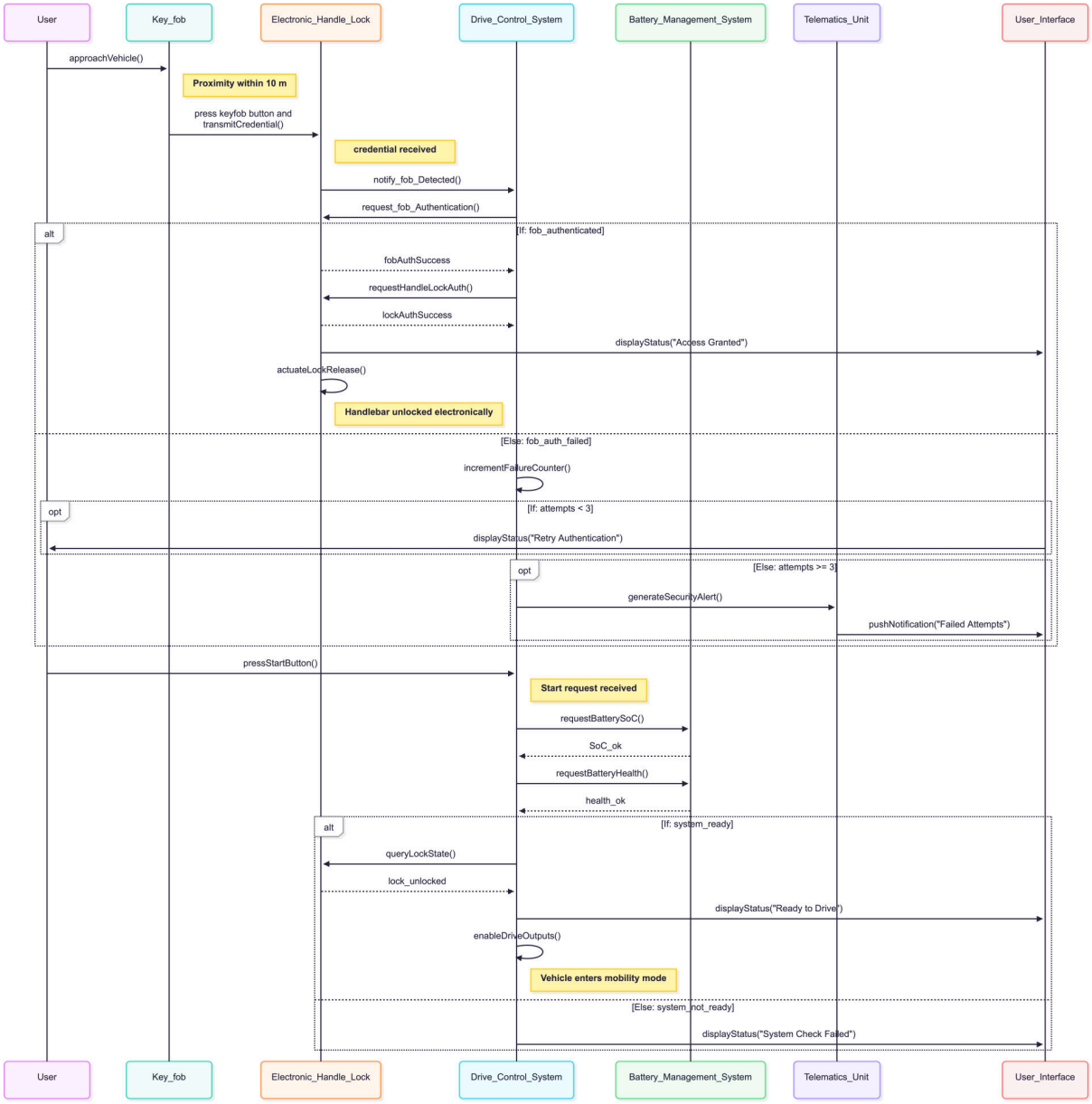


**Figure 8. Sequence Diagram – Updated.**

| Component ↓\→ | HLC | DCS | MC | BMS | CC | Flashing ECU | TCU | UA | CAN | DC | AC | GSM | GPS | Degree |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HLC | X | | | | | | | | CAN (communication) | | | | | 1 |
| DCS | Control | X | | | | | | | CAN (communication) | Power_DC (DC bus) | | | | 3 |
| MC | | | X | | | | | | CAN (communication) | Power_DC (DC bus) | | | | 2 |
| BMS | | | | X | | | | | CAN (communication) | Power_DC (DC bus) | | | | 2 |
| CC | | | | Control | X | | | | | | Power_AC (AC mains) | | | 2 |
| Flashing ECU | | | | | | X | | | CAN (communication) | | | | | 1 |
| TCU | | | | | | | X | | CAN (communication) | | | Wireless (GSM/GPS) | Wireless (GSM/GPS) | 3 |
| UA | | | | | | | | X | | | | Wireless (GSM/GPS) | | 1 |
| CAN | CAN (communication) | CAN (communication) | CAN (communication) | CAN (communication) | | CAN (communication) | CAN (communication) | | X | | | | | 6 |
| DC | | Power_DC (DC bus) | Power_DC (DC bus) | Power_DC (DC bus) | | | | | | X | | | | 3 |
| AC | | | | | Power_AC (AC mains) | | | | | | X | | | 1 |
| GSM | | | | | | | Wireless (GSM/GPS) | Wireless (GSM/GPS) | | | | X | | 2 |
| GPS | | | | | | | Wireless (GSM/GPS) | | | | | | X | 1 |

**Figure 9. Dependency Structure Matrix – Baseline**

**Dependency Structure Matrix (derived from updated LA)**

| Component ↓ \ → | EHL | DCS | MC | BMS | CC | Flashing ECU | TCU | UA | CAN | DC | AC | GSM | GPS | Degree |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **EHL** | X | | | | | | | | CAN (communication) | | | | | 1 |
| **DCS** | Control | X | | | | | | | CAN (communication) | Power_DC (DC bus) | | | | 3 |
| **MC** | | Power_DC (DC bus) | X | | | | | | CAN (communication) | Power_DC (DC bus) | | | | 2 |
| **BMS** | | | | X | | | | | CAN (communication) | Power_DC (DC bus) | | | | 2 |
| **CC** | | Control | | | X | | | | | | Power_AC (AC mains) | | | 2 |
| **Flashing ECU** | | | | | | X | | | CAN (communication) | | | | | 1 |
| **TCU** | | | | | | | X | | CAN (communication) | | | GSM/ GPS | GSM/ GPS | 3 |
| **UA** | | | | | | | | X | | | | GSM/ GPS | | 1 |
| **CAN** | CAN (communication) | CAN (communication) | CAN (communication) | CAN (communication) | | CAN (communication) | CAN (communication) | | X | | | | | 6 |
| **DC** | | Power_DC (DC bus) | Power_DC (DC bus) | Power_DC (DC bus) | | | | | | X | | | | 3 |
| **AC** | | | | | Power_AC (AC mains) | | | | | | X | | | 1 |
| **GSM** | | | | | | | | | | | | X | | 2 |
| **GPS** | | | | | | | | | | | | | X | 1 |

**Figure 10. Dependency Structure Matrix – Updated**

**Requirements Traceability Matrix - Baseline traceability matrix: trace_1**

| Req ID | Requirement | FA Function(s) | LA Component(s) | SMD State(s) | SD Interaction(s) |
|---|---|---|---|---|---|
| **R1** | Unlock via mechanical key | Provision_for_Mechanical_Key_Insertion<br>Verify_Mechanical_Key<br>Unlock_Handle_bar | Handlebar Lock Controller (HLC) | Locked → Mobility (via key_authenticated) | user→HLC: insertKey<br>HLC→HLC: verifyMechanicalKey<br>HLC→user: unlock_handlebar |
| **R2** | Vehicle mobility: forward, reverse, limp-home drive | Forward_Drive<br>Reverse_Drive<br>Limp_Home_Mode | Drive Control System (DCS) | Mobility → Driving (Forward/Reverse)<br>Driving → Limp_Home | user→DCS: pressStart<br>DCS→DCE: engage_drive_mode<br>DCS→user: enableDriveControls |
| **R3** | Battery charging (plug-in/CC/CV modes, session management) | Charge_Plug_Detection<br>CC_Battery_Charging<br>CV_Battery_Charging<br>Session_Termination | Battery Management System (BMS)<br>Charger | Idle → Charging (plug_in)<br>Charging → Idle (plug_out) | to be detailed in charging SD |
| **R4** | ECU software update (download, verify, flash) | Check_Version<br>Download_Package<br>Verify_Signature<br>Enter_Bootloader<br>Write_Firmware<br>Exit_Bootloader | Flasher ECU | Idle → SoftwareUpdate<br>SoftwareUpdate→ Idle | to be detailed in update SD |
| **R5** | Transmit ride data, battery SoC & analytics to user | Compute_Drive_Patterns<br>Analyze_SoC_Trends<br>Package_for_Transmission<br>Send_to_UserApp | Telematics Unit (TCU)<br>UserApp | not invoked in these core SMDs | to be detailed in data-analytics SD |

**Figure 11. Requirements Traceability Matrix – Baseline**

**Requirements Traceability Matrix - Updated traceability matrix**

| Req ID | Requirement | FA Function(s) | LA Component(s) | SMD State(s) | SD Interaction(s) |
|---|---|---|---|---|---|
| **R1'** | Vehicle Entry via wireless Key-FOB + dual auth + notify | Detect_KeyFOB_Presence<br>Authenticate_KeyFOB<br>Authenticate_HandleLock<br>Count_Failed_Attempts<br>Notify_Owner_Via_App<br>Unlock_Handlebar<br>Engage_Lock<br>Verify_Lock_Engaged | Electronic Handle Lock (EHL)<br>Drive Control System (DCS)<br>Telematics Unit (TCU)<br>UserApp | Locked → Authentication Sequence {AuthFOB, AuthEHL, Count}.<br>Fault (notify path) | user→EHL:sendUnlockSignal<br>EHL→DCS:reportFOB<br>DCS→EHL:requestAuth<br>EHL→DCS:sendAuthResponse<br>failure loops,DCS→TCU:notifyOwner<br>TCU→user:pushNotification |
| **R2** | Vehicle Mobility (unchanged) | Forward_Drive … Limp_Home_Mode | Drive Controller, MotorController | Mobility → Forward/Reverse/ Limp_Home | (unchanged Vehicle Start-Up group) |
| **R3** | Vehicle Battery Charging | Charge_Plug_Detection… SoC_Estimation | Battery Management System (BMS), Charger | Charging → CC/CV/Trickle | (unchanged Vehicle Start-Up group) |
| **R4** | ECU Software Update | Check_Version…Exit_Bootloader | Software Flashing, CAN_Bus | SoftwareUpdate → Download/Verify/ Flash | (unchanged Vehicle Start-Up group) |
| **R5** | Data Analytics & Telemetry | Data_from_Sensors…Send_to_UserApp | Telematics, UserApp | (unchanged) | (still not in SD2– coming in future sequences) |
| **R6** | In-Vehicle ECU-ECU communication (CAN) | — | CAN_Bus | (implicit) | implicit in SD2 message routes |

**Figure 12. Requirements Traceability Matrix – Updated**

### F. Summary of Case Study Findings

The keyless entry change measurably increased structural and behavioral complexity and concentrated propagation potential on a small set of components. The impact engine produced an interpretable ranking that directly maps to verification, security, and architecture actions. These outputs enable evidence-based prioritization that reduces wasted test effort and focuses engineering resources where they matter most.

## VII. Discussion & Industrial Utility

The engine translates model edits into measurable, traceable consequences. In the EV case the replacement of a mechanical key with a keyfob produced three observable effects:

- **Structural growth** — +3 functions and +4 flows—these reflect more responsibilities distributed across components and thus higher coordination cost (scheduling, interfaces, integration).
- **Behavioral branching** — +4 cyclomatic paths; each independent path implies additional tests and verification scenarios.
- **Concentrated propagation potential** — DSM shows CAN remains a high-degree hub; adding EHL/TCU increased local coupling. This reveals where regression testing and architecture hardening yield the largest return.

These concrete outputs enable engineers to move from intuition to prioritized action: focus verification and security resources where the engine ranks impact highest, and schedule architectural mitigation for high-degree nodes. We recommend the following minimal process to integrate the engine into an MBSE practice:

- **Model snapshot policy** — Define reviewable snapshots (milestones) with required SysML v1 exports.
- **Automated run:** — On each snapshot or change request, run the pipeline to produce metric deltas, DSM maps, and ranked impact lists. Publish CSV + figures to the project trace/regression board.
- **Triage board** — Systems engineers, architects and test leads review the ranked items weekly—accept/adjust weights and decide mitigation (test, refactor, monitor).
- **Feedback loop** — Use telemetry and verification outcomes to calibrate weights and validate prediction quality.

This workflow keeps the model the single source of truth, adds programmability via the pipeline, and embeds impact assessment into engineering cadence.

The approach is broadly applicable to cyber-physical systems (automotive, aerospace, healthcare, defense) that maintain SysML artifacts.

**Key strengths:**

- **Tool-agnostic metrics** — metrics are representation-centric (FA/LA/SMD/SD) and therefore portable.
- **Transparent and auditable** — all calculations come from exported model elements; results are reproducible.
- **Action-oriented outputs** — ranked remediation items map directly to test plans and architectural work.

**Limitations and caveats:**

- **Model fidelity dependence** — poor or incomplete modeling yields low-quality guidance. Enforcement of modeling standards and modeler training is necessary.
- **Granularity sensitivity** — while we demonstrated robustness under reasonable perturbations, organizations must standardize granularity rules to avoid metric drift.
- **Behavioral semantics complexity** — deeply nested/ orthogonal SMDs or complex sequence interactions require careful flattening and may demand manual review.
- **Toolchain maturity** — SysML v1 XMI semantics vary between tools; our mapping reduces friction but occasional manual mapping is required for site-specific exports.

**Business & organizational value:**

The engine creates measurable business value by:

- **Reducing verification waste** — by focusing tests on highest-impact elements, organizations can reduce redundant testing and shorten verification cycles.
- **Risk visibility** — DSM heatmaps and impact rankings expose hidden coupling and help prioritize architectural refactoring where it matters.

- **Faster change decisions** — engineers and managers can quantify the expected verification burden and make cost-benefit decisions about proposed features early.
- **Auditability for safety/regulation** — metric artifacts (CSV + DSM) support compliance and certification evidence by tracing requirement changes to verification scopes.

**Interpretation:** the framework is practically useful: it produces concrete, actionable intelligence from the models engineers already author. Adoption requires modest process changes (governance, tooling) but yields higher-confidence decisions, better prioritization, and measurable reductions in verification overhead.

# VIII. Conclusion

**Conclusion** — This paper introduced a reproducible, model-based impact-analysis framework that quantifies structural and behavioral complexity from SysML v1 artifacts and ranks the likely propagation of change using a combined metric + topology score. By formalizing a compact set of metrics (function/flow counts, functional density, cyclomatic complexity, DSM coupling) and operationalizing them in a Python pipeline, we demonstrated how a single requirement change (mechanical key $\rightarrow$ keyfob dual authentication) produces measurable increments in both structural interactions and behavioral paths, and concentrates risk on a small set of components (EHL, TCU, CAN). The outputs—metric deltas, DSM maps, ranked impact lists—are directly actionable for test planning, security hardening, and architectural mitigation.

**Closing remarks** — By making system complexity measurable, traceable, and operationally actionable from the authoritative SysML models that engineers already create and maintain, this work advances Model-Based Systems Engineering from a descriptive practice toward a prescriptive and decision-enabling engineering discipline. Rather than relying on intuition or post-hoc reasoning, the proposed framework allows teams to quantitatively assess the downstream consequences of requirement changes early in the lifecycle and to prioritize engineering effort where it delivers the highest value. This capability is particularly relevant for modern cyber-physical systems, where increasing connectivity and software-driven functionality amplify change-propagation risk. Embedding such an impact-analysis engine into routine MBSE workflows enables faster, evidence-based trade decisions, more focused verification strategies, and improved transparency for both technical and managerial stakeholders. Ultimately, the approach supports a shift toward more resilient architectures and more predictable development outcomes in complex engineered systems.

**Acknowledgments** — The authors would like to express thankfulness to Mr. Aniket Bhattacharya, Vice President, Intelligent Systems Department, Bajaj Auto Technology Limited, for his continued support, encouragement, and technical guidance in bringing out this research work. Any remaining errors or omissions are solely the responsibility of the authors.

# IX. References


McCabe, T. J. (1976). *A complexity measure*. IEEE Transactions on Software Engineering, 2(4), 308–320. https://doi.org/10.1109/TSE.1976.233837

Steward, D. V. (1981). *The design structure system: A method for managing the design of complex systems*. IEEE Transactions on Engineering Management, 28(3), 71–74. https://doi.org/10.1109/TEM.1981.6448589

Sheard, S. A., & Mostashari, A. (2010). *A complexity typology for systems engineering*. Proceedings of the INCOSE International Symposium. Available at: https://web.mst.edu/lib-circ/files/Special%20Collections/INCOSE2010/A%20Complexity%20Typology%20for%20Systems%20Engineering.pdf

Potts, M. W., Johnson, A., & Bullock, S. (2020). *Evaluating the complexity of engineered systems: A framework informed by a user case study*. Systems Engineering, 23(6), 707–723. https://doi.org/10.1002/sys.21558

INCOSE. (2023). *Systems engineering handbook: A guide for system life cycle processes and activities* (5th ed.). John Wiley & Sons.

Object Management Group. (2012). *Systems Modeling Language (SysML) specification* (Version 1.3). https://www.omg.org/spec/SysML/1.3/

Eppinger, S. D., & Browning, T. R. (Eds.). (2016). *Design structure matrix methods and applications*. MIT Press.

Browning, T. R. (2001). *Applying the design structure matrix to system decomposition and integration problems: A review and new directions*. IEEE Transactions on Engineering Management, 48(3), 292–306. https://doi.org/10.1109/17.946528

Yassine, A., et al. (2003). *Complex concurrent engineering and the design structure matrix*. IEEE Transactions on Engineering Management, 50(5).

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). *Exploring network structure, dynamics, and function using NetworkX*. Proceedings of the 7th Python in Science Conference, 11–15. https://networkx.org/

Kolovos, P. J., Polack, F. A. C., Paige, R. F., & Gray, J. A. G. (n.d.). *pyecore: Python implementation for EMF*. https://github.com/pyecore/pyecore

# Biography

**Sankalp Kumar**
Sankalp Kumar is an Embedded Software and Systems Engineer with industry experience in automotive electric vehicles and model-based systems engineering. He works in the Intelligent Systems Department at Bajaj Auto Technology Limited, where his work spans embedded software development, system architecture modeling, and system complexity analysis. His current interests include MBSE, impact analysis, architectural complexity metrics, and the application of digital engineering methods to large-scale cyber-physical systems. This paper reflects his ongoing work on quantifying and managing system complexity using executable model-driven pipelines.

**Prerana Kulkarni**
Prerana Kulkarni is a Systems Engineer in the Intelligent Systems Department at Bajaj Auto Technology Limited. Her work focuses on system architecture definition, functional and logical modeling, and requirements traceability for automotive embedded systems. She has experience applying systems engineering methods to vehicle subsystems, with particular interest in verification planning, system behavior modeling, and change impact assessment. Her contributions to this work center on model structuring, traceability analysis, and validation of the proposed framework on an industrial EV case study.