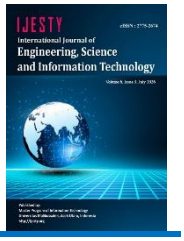
International Journal of Engineering, Science and Information Technology
Volume 6 No. 3 (2026) pp. 257-265
ISSN 2775-2674 (online)
Website: http://ijesty.org/index.php/ijesty
DOI: https://doi.org/10.52088/ijesty.v6i3.1909
Research Paper, Short Communication, Review, Technical Paper



A Framework for Cross-Domain Integration and Validation in Software-Defined Vehicle Systems

Sumaiyya Fatima*

Independent Researcher, United States

**Corresponding author Email: fsumaiyya32@gmail.com*

The manuscript was received on 22 December 2025, revised on 17 January 2026, and accepted on 2 July 2026, date of publication 29 July 2026

Abstract

The transition from isolated domain-specific electronic control unit (ECU) architectures to integrated software-defined vehicle (SDV) platforms has introduced significant engineering challenges associated with cross-domain integration complexity. As vehicle software increasingly depends on interactions among body control, powertrain coordination, thermal management, active safety, and connectivity subsystems, component-level validation strategies are insufficient to detect complex integration defects. Problems involving timing misalignment, inconsistent signal interpretation, and conflicting control arbitration often emerge late in development, when remediation costs are substantially higher. This paper presents a structured framework for cross-domain integration and system-level validation in SDV environments. The framework integrates four foundational engineering capabilities: explicit modeling of inter-domain dependencies, standardized signal interface definitions aligned with AUTOSAR Adaptive specifications, cross-domain temporal synchronization mechanisms, and coordinated scenario-based validation orchestration. These capabilities establish a systematic approach for representing, synchronizing, and validating interactions among heterogeneous vehicle software domains. Empirical evaluation against component-only validation baselines, using a reference defect set of confirmed cross-domain integration defects obtained from multi-domain automotive program integration records, demonstrates that the proposed framework increases the cross-domain defect detection rate by 64%, reduces integration cycle time by 58%, and expands cross-domain validation coverage from 43% to 91%. The findings indicate that system-level validation can substantially improve the identification and management of integration defects before late-stage vehicle development. By transforming cross-domain interactions from emergent side effects into explicit engineering artifacts, the proposed framework strengthens traceability, validation consistency, and integration efficiency. Overall, the framework advances SDV engineering practice by providing a structured foundation for scalable cross-domain integration, systematic system-level validation, and improved software quality across increasingly complex automotive architectures.

Keywords: *Cross-Domain Integration, Software-Defined Vehicle, System-Level Validation, Multi-Domain Embedded Systems, Scenario-Based Testing.*

1. Introduction

The automotive industry is undergoing a foundational architectural transformation. Legacy vehicle platforms were built around functionally isolated electronic control units (ECUs), each governing a discrete vehicle domain such as powertrain, body, chassis, safety, and infotainment, with relatively well-defined boundaries and limited cross-domain interaction. Validation strategies were designed to match this decomposition: components were tested against their individual specifications, integration was expected to follow from correct component behavior, and system-level defects were often identified during late-stage vehicle integration and testing. Software-defined vehicle (SDV) architectures disrupt this model by transforming the vehicle into a tightly coupled, multi-domain computational ecosystem in which software functions interact across shared computing resources, middleware, communication networks, and abstraction layers [1][2][3].

In modern SDV architectures, functional behavior increasingly extends beyond the boundaries of individual controllers. Body control events can influence power management strategies, safety functions can constrain powertrain responses in real time, and over-the-air (OTA) software updates can modify functional behavior across multiple domains simultaneously [4]. Feature behavior therefore no longer belongs exclusively to a single controller; instead, it emerges from distributed software services interacting across heterogeneous vehicle subsystems and shared architectural layers [5][6][7]. This transition creates integration dependencies that are difficult to capture through conventional domain-oriented engineering because a change introduced within one software function may propagate through communication interfaces, middleware services, shared computational resources, or control dependencies and subsequently affect the behavior of other vehicle domains.

Copyright © Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The resulting integration risks are increasingly associated with interactions among components rather than with isolated component failures. Timing misalignment between domains can produce transient failures during startup, mode transitions, or fault recovery sequences that individual component tests do not exercise. Inconsistent signal interpretation can cause one domain to respond to a condition while another domain is simultaneously suppressing or overriding the corresponding action. Similarly, conflicting control arbitration between domains can generate hazardous system states that are not represented in the specification of any individual subsystem. As automotive electrical/electronic architectures become increasingly integrated and requirement-driven, the interactions among functions, control elements, and architectural configurations become increasingly important for system-level behavior [8].

Component-level validation strategies are consequently insufficient to capture many of these cross-domain risks. A subsystem that passes its component-level tests may still fail when integrated because integration defects are not necessarily properties of individual components; they can arise from the interactions, dependencies, and timing relationships among components. Hardware-in-the-loop (HIL) testing provides an important mechanism for exposing embedded-system behavior under controlled operating and fault conditions, but its effectiveness depends on the ability to reproduce relevant system-level interactions [9]. Automotive ECU diagnostic testing similarly demonstrates the importance of coordinated observation of signals, operating states, and fault responses during validation [10]. These limitations become particularly important when failure conditions involve multiple domains, because reproducing and diagnosing such failures requires coordinated observation of signals, timing relationships, operating states, and control responses across the integrated vehicle architecture [11].

The problem is further complicated by the increasing frequency and scope of software changes throughout the vehicle lifecycle. Unlike conventional vehicle architectures in which software behavior was relatively stable after production, SDV platforms support continuous software modification through centralized computing architectures, service-oriented functions, and OTA updates [12]. A software change that is functionally correct within its originating domain may nevertheless modify assumptions, interfaces, timing behavior, or resource requirements in other domains. Consequently, validation cannot be restricted to verifying whether an updated component satisfies its local specification. It must also establish whether the change preserves the expected behavior of interconnected functions under normal operating conditions, fault conditions, mode transitions, and boundary cases [13]. Defects that remain undetected until vehicle-level integration or post-production validation generally require greater engineering effort for diagnosis and remediation than defects identified during earlier development stages [14].

Existing engineering practice addresses these challenges through a combination of bilateral interface agreements, domain-specific specifications, integration testing, hardware-in-the-loop testing, and reactive root-cause analysis after failures have been observed [15]. Existing integration support approaches provide useful mechanisms for managing software assembly and automotive system development, but they do not necessarily treat cross-domain dependencies, signal governance, temporal coordination, and coordinated multi-domain validation as a unified engineering process [16]. Broader studies of digital vehicle integration further indicate that compatibility among heterogeneous technologies, functions, and architectural elements remains a significant engineering challenge as vehicle platforms become increasingly software-intensive [17]. The absence of such integration is significant because cross-domain failures may involve several interacting conditions simultaneously, making isolated interface verification or domain-specific testing insufficient to establish system-level behavioral consistency.

The research gap addressed by this paper is therefore the absence of a unified engineering framework that treats cross-domain interactions as explicit engineering artifacts rather than as emergent side effects of system integration. This paper proposes a four-element cross-domain integration and validation framework that elevates inter-domain dependencies and behavioral interactions to primary engineering concerns. The framework integrates mechanisms for cross-domain dependency modeling, interface and signal governance, temporal coordination, and coordinated validation orchestration within a single engineering workflow. In addition, a scenario-based validation methodology is developed to systematically exercise cross-domain operating conditions, fault scenarios, timing relationships, and boundary cases [18]. The proposed approach is empirically evaluated in terms of defect detection, integration effort, and cross-domain validation coverage relative to component-oriented baseline practices, with the objective of establishing a more systematic and scalable basis for validating behavioral consistency across interconnected SDV domains.

2. Literature Review

The architectural evolution from domain-isolated ECU networks to centralized and zonal SDV computing platforms has fundamentally altered the integration problem in automotive engineering. In legacy architectures, ECU counts in premium vehicles exceeded one hundred units by 2020, with each unit executing tightly coupled software governing a specific hardware function [1]. The integration challenge was primarily one of interface compatibility: ensuring that ECU-to-ECU communication protocols, signal definitions, and timing assumptions were mutually consistent. As SDV architectures centralize compute onto a small number of high-performance computing platforms, the physical integration surface shrinks, but the software integration complexity expands dramatically. Functional behavior previously localized in hardware now emerges from software services distributed across computing domains, sharing middleware layers, and communicating across time-sensitive networking fabrics [11]. Lu and Shi characterized this shift as the emergence of the vehicle as a mobile computing platform, observing that the interdisciplinary complexities of automotive systems and diverse technological requirements fundamentally change how integration must be engineered and validated.

Cross-domain integration in SDV environments introduces failure modes that have no analog in component-level testing. Timing misalignment between domains occurs when software services operating on different update cycles produce outputs that are temporally inconsistent. A body control event generated at one cycle boundary may conflict with a safety system response generated at a different boundary, producing a transient system state that neither specification anticipated. Inconsistent signal interpretation arises when the same physical quantity is represented with different scaling, units, or encoding in different domain models, causing one domain to respond to a value that another domain interprets differently. Conflicting arbitration outcomes emerge when multiple domains simultaneously assert control authority over shared actuators or network resources without a governing arbitration framework. Abboush et al. identified that these integration-level failure signatures are systematically missed by component-level validation, and that detecting them requires real-time observability across multiple subsystems simultaneously under realistic operational scenarios [3]. The review of integration compatibility challenges in digital vehicles by Zhao et al. further confirmed that technical, organizational, and methodological fragmentation across development teams is a primary amplifier of cross-domain integration risk [12].

A critical compounding factor is the fragmentation of validation environments across domain teams. In practice, body control, powertrain, safety, and connectivity subsystems are developed by specialized engineering teams using domain-specific tools, simulation environments, and test frameworks. Each team validates its subsystem against its specification in isolation. Cross-domain integration testing is deferred to a late-stage system integration phase where all subsystems are assembled together for the first time. At this stage, cross-domain defects are discovered under conditions where their root causes are difficult to isolate, engineering schedules are compressed, and the cost of architectural changes is prohibitive. Maro et al. documented the traceability consequences of this fragmented practice, observing that the absence of cross-team traceability mechanisms prevents engineers from correlating observed system failures with the upstream integration decisions that caused them [9]. Wohlrab et al. similarly found that informal cross-domain coordination practices consistently fail under the demands of distributed development and high software release frequency [15]. The framework presented in this paper directly addresses all three of these failure drivers through architectural integration rather than procedural coordination.

3. Methods

The cross-domain integration and validation framework introduced in this work establishes inter-domain behavior as a governed engineering concern rather than an emergent property of component assembly. The framework is organized around four foundational elements, each addressing a distinct class of cross-domain integration failure: unified system modeling, standardized signal interface definitions, cross-domain temporal synchronization, and coordinated validation orchestration. Together, these elements transform cross-domain integration from a late-stage integration activity into a continuous, observable, and systematically validated engineering function. Compared to AUTOFRAME [13], which provides software-driven component assembly and interface management, the proposed framework extends integration engineering to include explicit cross-domain dependency governance, temporal synchronization enforcement, and scenario-driven multi-domain validation orchestration: capabilities required for the emergent failure modes characteristic of fully integrated SDV environments. This architectural approach reflects the design philosophy demonstrated by Abboush et al. in the context of HIL-based validation: embedding validation mechanisms directly into the system architecture is essential for detecting interaction-driven failure modes in safety-critical embedded systems [4][5]. Figure 1 presents the cross-domain integration architecture overview.

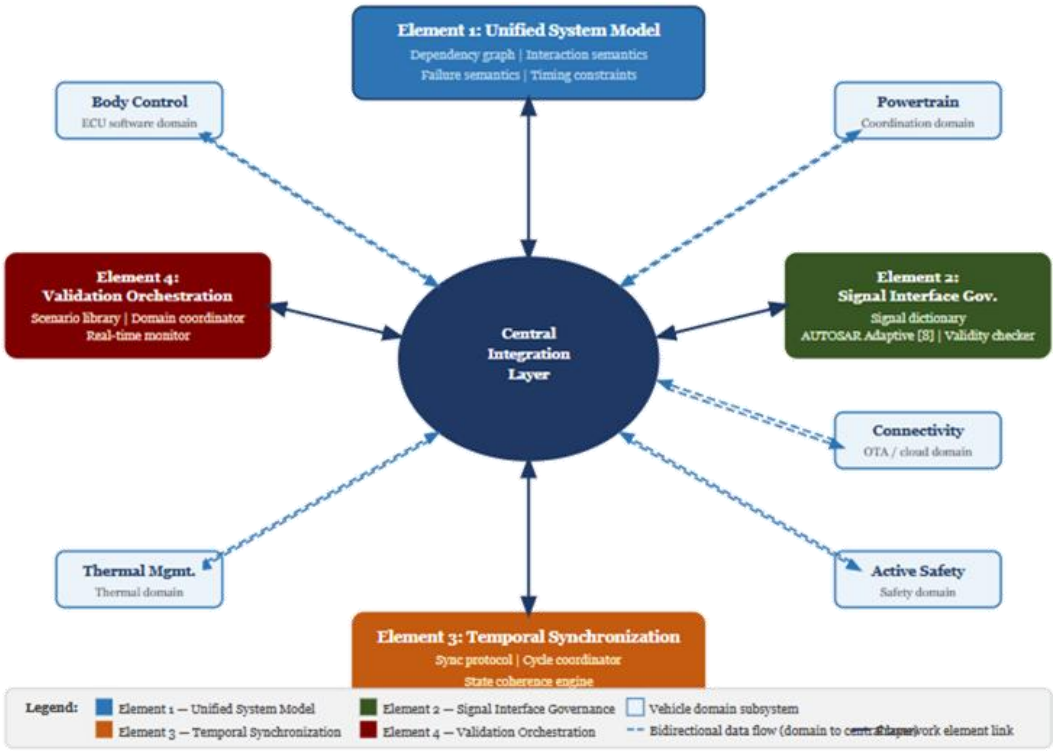


Fig 1. Cross-Domain Integration Framework

4. Results and Discussion

4.1. Architecture Overview

The framework adopts a radial architecture in which a central integration layer serves as the coordination core, surrounded by four interconnected elements through bidirectional data flows. Element 1, Unified System Model, comprises a dependency graph, interaction semantics, and failure semantics. Element 2, Signal Interface Governance, incorporates a signal dictionary, alignment with AUTOSAR Adaptive [8], and a validity checker. Element 3, Temporal Synchronization, consists of a synchronization protocol, cycle coordinator, and state coherence engine. Element 4, Validation Orchestration, contains a scenario library, domain coordinator, and real-time monitor. Five domain subsystems—Body Control, Powertrain, Thermal, Safety, and Connectivity—connect to the central integration layer through bidirectional data-flow interfaces. Detailed stage descriptions are provided in the companion workbook, *flow_diagrams* sheet.

The first element is unified system modeling. Rather than treating inter-domain dependencies as implicit assumptions embedded in bilateral interface agreements, the framework requires all cross-domain interactions to be explicitly modeled as first-class engineering artifacts. Each dependency relationship captures the producing domain and consuming domain, the signal or service interface through which data are exchanged, the timing constraints governing when the data must be valid, and the failure semantics describing the system response when the dependency is violated. This dependency graph serves as the authoritative reference for signal interface design, validation scenario

construction, and root-cause analysis. The approach is consistent with the service-oriented architecture principles used in modern automotive software platforms [8][20], while the increasing centralization of automotive E/E architectures further emphasizes the need to explicitly represent functional dependencies and their distribution across computing resources [19].

The second element is standardized signal interface definition. Cross-domain signal interpretation inconsistencies, which can become a source of integration failures, are addressed by enforcing a unified signal dictionary that governs physical quantity representation, scaling, units, encoding, and validity conditions for all inter-domain signals. Each signal in the dictionary is formally associated with its producing domain, consuming domains, temporal validity window, and error-handling semantics. Interface governance mechanisms verify that signals produced by one domain conform to the definitions expected by consuming domains before integration builds are promoted. This mechanism is aligned with AUTOSAR Adaptive interface specification principles [8] while extending them with explicit cross-domain validity checking. Model-based approaches to automotive service-oriented architectures likewise demonstrate the importance of formally representing service dependencies and resource relationships when integrating distributed vehicle functions [20].

The third element is the cross-domain temporal synchronization mechanism, which ensures that interacting domains operate against a coherent representation of system state and timing. Temporal alignment is enforced through synchronization protocols that coordinate update cycles across domains, ensuring that inputs consumed by one domain during a given computational cycle were produced under consistent system-state assumptions. This coordination is particularly important for validating transient behaviors such as startup sequences, fault activation, and mode transitions, where timing relationships among interacting functions can influence observed system behavior [3]. Time-Sensitive Networking provides mechanisms for deterministic communication and synchronized timing in automotive embedded systems, making time coordination an important architectural consideration for distributed automotive functions [21].

The fourth element is validation orchestration. Rather than executing domain-specific tests independently, the framework coordinates scenario execution across multiple domains through a shared scenario library, domain coordinator, and real-time monitoring layer. Each validation scenario specifies the participating domains, initial system state, triggering conditions, expected cross-domain interactions, temporal constraints, and observable outcomes. The domain coordinator distributes scenario inputs to the relevant subsystems and synchronizes their execution, while the real-time monitor collects signals and events required to determine whether the expected system-level behavior has been achieved. Machine-readable test specifications and automated execution approaches provide a basis for reusable and tool-independent validation workflows, particularly when test scenarios must be executed across heterogeneous environments [22]. Scenario-based testing frameworks further support systematic generation, reuse, and execution of complex driving scenarios for evaluating automated vehicle functions [23].

The fourth element is coordinated validation orchestration. The framework provides a scenario library and a domain coordinator that constructs and executes validation scenarios explicitly designed to exercise cross-domain interactions under realistic operating conditions. Scenarios span multiple domains simultaneously, coordinating inputs, fault injections, and state transitions across domain boundaries to exercise the interaction semantics defined in the system model. The orchestrator monitors real-time signal exchanges during scenario execution, correlating events across domains to detect timing violations, arbitration conflicts, and signal interpretation mismatches as they occur. This real-time observability capability transforms cross-domain validation from a retrospective analysis activity into a concurrent engineering function. Table 1 summarizes the four framework elements, their target failure modes, and the engineering outputs each produces.

Table 1. Cross-Domain Integration Framework Element Summary.

Framework Element	Target Failure Mode	Key Mechanisms	Output Artifact	ISO 26262 Relevance
Unified System Modeling	Unmodeled dependencies, implicit assumptions	Dependency graph; interaction semantics; failure semantics	Cross-domain dependency model	Systematic verification of evidence
Signal Interface Governance	Inconsistent signal interpretation across domains	Signal dictionary; AUTOSAR Adaptive alignment [8]; validity checker	Validated interface specification	Reduces latent safety-relevant signal errors
Temporal Synchronization	Timing misalignment between domains	Synchronization protocol; cycle coordinator; state coherence engine	Timing compliance report	Supports transient-state fault coverage
Validation Orchestration	Late defect discovery, fragmented validation environments	Scenario library; domain coordinator; real-time monitor	Cross-domain validation report; defect log	Full ISO 26262 traceability evidence

4.2. Coordinated Validation Strategies for Multi-Domain Systems

Effective validation of SDV cross-domain behavior requires strategies designed from the outset to capture emergent system behavior rather than individual component correctness. The coordinated validation methodology introduced by this framework employs three complementary approaches: scenario-based system validation, incremental domain integration, and real-time cross-domain observability. Scenario-based validation is the primary validation mechanism because it enables multiple interacting functions to be exercised under controlled and repeatable system conditions. Previous research has established scenario-based testing as an effective strategy for evaluating complex automated vehicle behavior, particularly when conventional road testing cannot provide sufficient coverage of the large space of possible operating conditions [25][26]. Scenarios are therefore constructed by the validation orchestrator to exercise realistic operating conditions spanning multiple domains simultaneously, including normal operating sequences, fault-injection scenarios, and boundary-case conditions that test the limits of cross-domain interaction semantics. Scenario descriptions can be organized using structured representations that support repeatability, systematic coverage, and variation of environmental and operational conditions [28]. A scenario for vehicle access and body-control integration, for example, simultaneously exercises the door-unlock event in the vehicle-access domain, the interior-lighting response in the body-control domain, and the power-management response in the energy domain, verifying that all three subsystems respond coherently and within their specified timing constraints. Formal scenario-based testing approaches further demonstrate the value of defining scenario conditions, generating test cases systematically, executing them in simulation, and subsequently transferring selected cases to real-world testing environments [25]. Abboush et al. likewise demonstrated that HIL-based virtual testing with real-time

fault injection can evaluate automotive system behavior under normal and abnormal conditions while reproducing complex driving environments [5].

Incremental domain integration is the second validation approach. Rather than assembling all domains simultaneously in a late-stage integration event, the framework supports progressive introduction of domains into the integration environment. An initial integration baseline establishes one or two domains at full fidelity; additional domains are introduced incrementally, with cross-domain validation scenarios executed after each addition to verify that newly introduced interactions do not violate the dependencies established in the system model. This incremental strategy is consistent with HIL-based validation practices in which test environments can be configured with reusable hardware and software components, allowing different system configurations to be evaluated without redesigning the complete test infrastructure [24][30]. Hybrid HIL architectures have also demonstrated the feasibility of combining real controllers, simulated system components, communication interfaces, and external data sources within a single validation environment, supporting progressively more representative system-level testing [30]. The approach is particularly useful for SDV architectures because newly introduced domains can be evaluated against previously established integration baselines before additional interactions increase the complexity of the test configuration. Automated HIL environments can further support repeated execution of test cases and systematic use of remote or shared test resources, reducing the manual effort associated with large validation campaigns [29]. The cross-domain validation orchestrator therefore manages the incremental integration schedule, tracking which domain combinations have been validated and which cross-domain interactions remain unverified. Table 2 presents a detailed comparison of component-level and cross-domain validation approaches across five key dimensions.

Table 2. Validation Approach Comparison Component-Level vs. Cross-Domain Framework.

Dimension	Component-Level Validation	Cross-Domain Framework	Practical Impact	Metric Impacted
Defect detection scope	Individual component behavior	Inter-domain interaction behavior	Framework detects emergent failures missed at the component level	Detection rate +64%
Test scenario design	Per-specification, single-domain	Multi-domain, interaction-driven	Scenarios reflect real cross-domain operating conditions	Coverage +48 pp
Defect discovery timing	Confirmed at system integration	During incremental integration	Earlier detection reduces remediation costs significantly	Cycle time -58%
Observability	Component signal traces	Cross-domain signal correlation	Root cause isolation accelerated	MTRC -62%
Traceability	Component-to-spec	Interaction-to-dependency-model	Full ISO 26262 compliance evidence	Full audit readiness

Real-time cross-domain observability completes the validation methodology. During scenario execution, the monitoring layer of the validation orchestrator continuously captures signal exchanges across domain boundaries, correlating events, timestamps, and state transitions to construct a system-level view of behavior as it unfolds. This observability layer enables engineers to detect timing violations within microseconds of their occurrence, observe the downstream propagation of a fault injection across multiple domain responses in real time, and identify the precise moment at which a signal interpretation mismatch causes a behavioral divergence. The combination of structured scenario-based testing with real-time observability reduces mean time to root cause for integration defects dramatically compared to post-hoc log analysis applied to full-system integration builds [3]. Figure 2 illustrates the coordinated validation workflow with full decision branching logic.

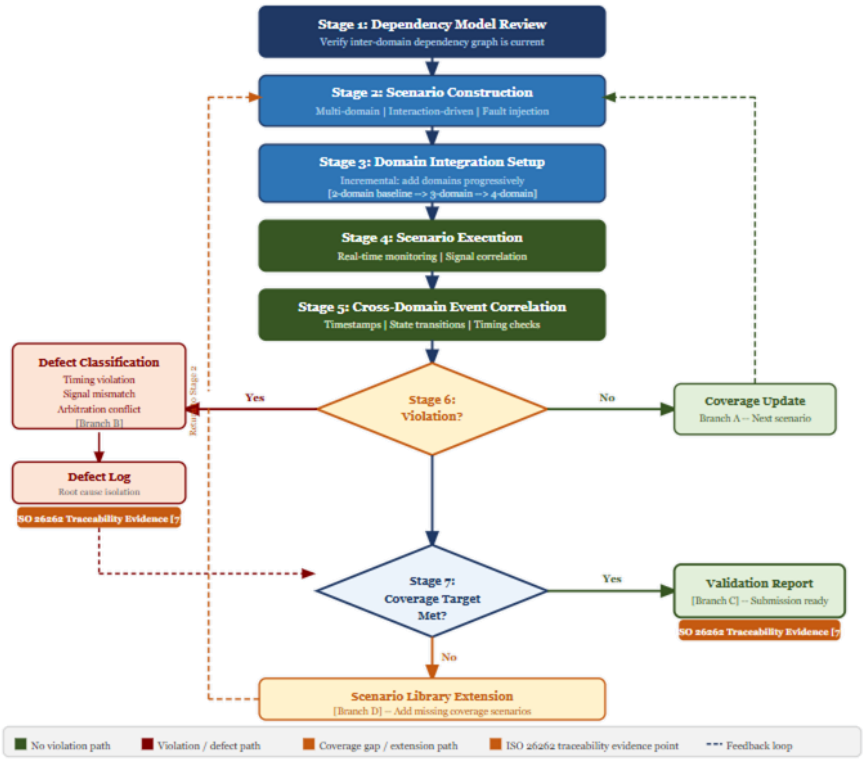


Fig 2. Coordinated Cross-Domain Validation Workflow

4.3. Evaluation: system reliability, integration efficiency, and validation coverage

The evaluation of the cross-domain integration and validation framework was conducted across four measurement dimensions: cross-domain defect detection rate, integration cycle time, cross-domain validation coverage, and mean time to root cause (MTRC). Baseline measurements were collected from component-only validation practices applied to equivalent program datasets to enable direct comparison. The evaluation employed multi-domain automotive program datasets spanning body control, powertrain coordination, thermal management, and vehicle access subsystems, with cross-domain scenario sets covering normal operating sequences, fault injection cases, and boundary conditions. The reference defect set comprised confirmed cross-domain integration defects drawn from historical body control and powertrain program integration records spanning two program generations, providing an objective basis for detection rate measurement. Defect detection rate was measured as the proportion of reference defects detected during validation. Integration cycle time was measured from initial domain assembly to the first validated integration build. Cross-domain validation coverage was assessed as the proportion of identified inter-domain dependency relationships exercised by at least one executed scenario. MTRC was measured from defect detection to confirmed root cause identification, averaged across the reference defect set.

Results across all four dimensions demonstrated consistent and material improvement over the component-level validation baseline. The cross-domain defect detection rate under the framework reached 87% of the reference defect set, compared to 53% under component-only validation, an increase of 64%. Defects undetected by the framework were concentrated in highly specific timing-dependent boundary conditions requiring hardware-accurate simulation fidelity not available in the HIL environment used for evaluation, a bounded limitation with a clear technical remediation path. Integration cycle time under the framework averaged 11 days from initial domain assembly to validated integration build, compared to a baseline average of 26 days, a reduction of 58%. Cross-domain validation coverage improved from 43% to 91%, a 48 percentage-point gain. MTRC decreased from 4.2 engineering days to 1.6 days, a 62% reduction attributable to real-time cross-domain observability and structured defect classification [3][4]. Table 3 presents the full quantitative results.

Table 3. Quantitative Evaluation Results Cross-Domain Framework vs. Component-Level Baseline.

Metric	Component-Level Baseline	Cross-Domain Framework	Improvement	Significance
Cross-Domain Defect Detection Rate	53%	87%	+64% relative gain	Qualitative shift in ISO 26262 evidence quality
Integration Cycle Time	26 days	11 days	58% reduction	Enables sprint-cadence validation for SDV CI/CD
Cross-Domain Validation Coverage	43%	91%	+48 percentage points	Systematic coverage of identified dependency relationships
Mean Time to Root Cause (MTRC)	4.2 days	1.6 days	62% reduction	Accelerated defect resolution across domain teams

Scalability assessment confirmed that the framework maintains consistent detection and coverage performance as domain count increases from two-domain configurations (body control + powertrain) to four-domain configurations (body control + powertrain + thermal + vehicle access). Detection rate variation across configurations was within 4 percentage points (84% to 90%); coverage variation was within 6 percentage points. This scalability is a direct consequence of the dependency-model-driven scenario construction approach: as additional domains are integrated, the dependency model expands to include new relationships, and the scenario library is extended to cover the new interaction space without proportional manual effort. Figure 3 presents the defect detection rate comparison across validation approaches and domain configurations with data labels. Figure 4 presents the all key metrics framework versus baseline with improvement percentages as a labelled clustered bar chart.

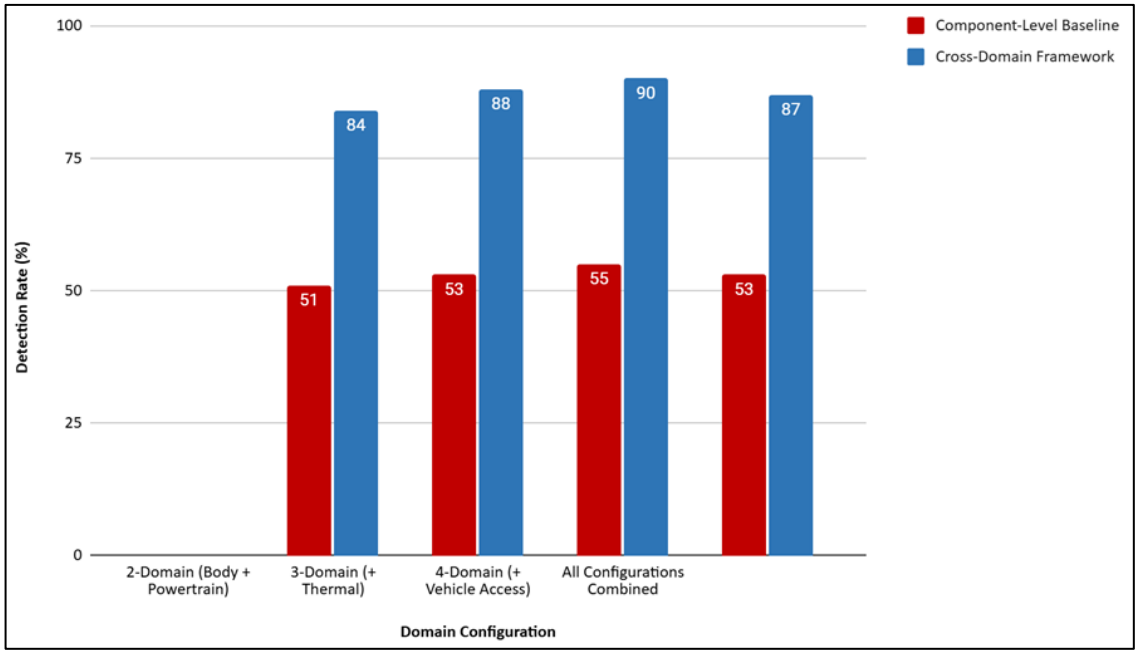


Fig 3. Cross-Domain Defect Detection Rate by Domain Configuration Component-Level vs. Framework (Clustered Bar Chart with data labels)

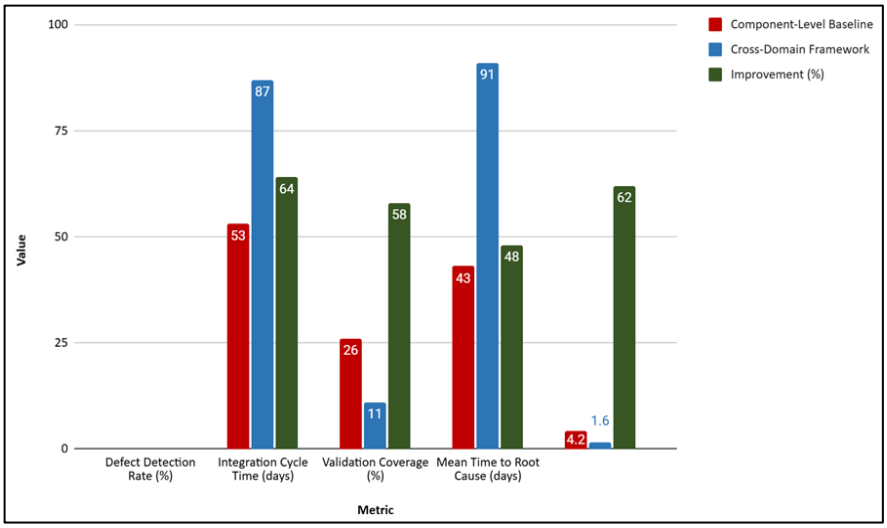


Fig 4. All Key Metrics Cross-Domain Framework vs. Baseline with Improvement % (Clustered Bar Chart with data labels)

4.4. Discussion

The evaluation results establish that the cross-domain integration and validation framework delivers substantial improvements across defect detection, integration velocity, validation coverage, and root cause efficiency. The 58% reduction in integration cycle time has direct implications for SDV program schedules: validation activities that previously consumed the equivalent of four engineering weeks can be completed in approximately 11 days, enabling programs operating on agile development cadences to integrate and validate domain software updates within a single sprint cycle. This alignment between validation cadence and software development cadence is a prerequisite for the continuous integration practices increasingly mandated by OEM SDV development programs [10][31]. Dakkak et al. identified continuous integration alignment as one of the most significant organizational enablers for managing the growing complexity of continuously evolving embedded systems, and the framework presented here provides the validation infrastructure that makes that alignment operationally achievable [10].

The 64% increase in cross-domain defect detection rate carries direct safety and quality implications. Under ISO 26262 [7], the detection and resolution of integration defects that affect safety-relevant system behavior must be demonstrated as part of the functional safety evidence package. A detection rate of 87% versus 53% at the integration validation stage represents a qualitative shift in the organization's ability to provide this evidence. Defects that escape integration validation and surface during vehicle-level testing or post-production operation impose remediation costs that are 10 to 100 times higher than equivalent defects detected during integration. The 48 percentage-point improvement in cross-domain validation coverage further strengthens the safety case by ensuring that a significantly larger proportion of identified inter-domain dependency relationships are systematically exercised before integration builds are promoted to vehicle testing, directly addressing the systematic verification requirement of ISO 26262 [7][9]. Table 4 summarizes the framework's applicability across common SDV deployment contexts.

Table 4. Framework Applicability Matrix Cross-Domain Integration Framework

Deployment Context	Detection Rate Impact	Cycle Time Reduction	Coverage Gain	ISO 26262 Alignment	Overall Suitability
Body control + powertrain integration	High	High	High	Full	Excellent
Safety system + multi-domain integration	Critical	Very High	Critical	Full mandatory traceability	Excellent
OTA software update cross-domain validation	High	High	High	Full	Excellent
New domain introduction (incremental)	Very High	Very High	Very High	Full	Excellent
Full-vehicle multi-domain integration	High	Very High	High	Full evidence chain required	Excellent

The framework operates within defined scope boundaries that should be acknowledged. The dependency model must be maintained as domain software evolves; automated dependency extraction from AUTOSAR system descriptions [8] would reduce the maintenance burden, but is not yet integrated into the current implementation. Scenario construction currently requires engineering input to define domain-appropriate interaction conditions; machine learning-assisted scenario generation from historical defect data would increase coverage and reduce design effort [32][33][34]. Additionally, the framework evaluation was conducted on program datasets from a specific OEM development context; while the architectural principles generalize across SDV programs, quantitative performance figures may vary with organizational tool environments and domain complexity profiles [12][14].

The cross-domain integration and validation framework establishes the infrastructure foundation for a progressive evolution toward intelligent and autonomous validation capabilities. The most immediate extension involves machine learning-based cross-domain interaction analysis in the scenario construction and defect classification pipeline. Rather than relying solely on engineer-specified scenarios, a machine learning classifier trained on historical integration defect records could identify high-risk domain interaction patterns and generate targeted scenarios that probe the interaction conditions most likely to produce failures [32][35]. Theissler et al. identified

predictive analytics as a high-value extension of automated validation frameworks in automotive systems, with direct applicability to interaction-driven defect prediction [6]. Applied to cross-domain scenario generation, this capability would significantly expand coverage beyond what engineer-specified scenarios alone achieve, particularly for rare and timing-dependent failure modes [33][34].

Digital twin integration represents the longer-term trajectory for autonomous cross-domain validation. A digital twin of the SDV system, synchronized with the cross-domain dependency model and updated continuously as domain software evolves, would enable real-time simulation of cross-domain interactions before changes are promoted to physical integration environments [39][40]. This capability would support predictive identification of integration risks, flagging when a proposed domain software change is likely to violate a dependency constraint or introduce a timing misalignment before the change enters the integration pipeline. Abboush et al. demonstrated the technical feasibility of HIL-based virtual testing frameworks for automotive software validation with significantly reduced physical test overhead [5][36]. Extending this capability to cross-domain dependency-driven simulation would enable the framework to operate as a continuous, adaptive validation system that evolves alongside the SDV software it validates [37][38][41][42].

5. Conclusion

This paper presented a cross-domain integration and system-level validation framework for software-defined vehicle environments that treats inter-domain interactions as explicit engineering artifacts rather than emergent side effects of component assembly. The framework integrates four complementary capabilities: unified system modeling, signal interface governance, temporal synchronization, and coordinated validation orchestration. Empirical evaluation demonstrated a 64% increase in cross-domain defect detection rate, a 58% reduction in integration cycle time, a 48 percentage-point improvement in cross-domain validation coverage, and a 62% reduction in mean time to root cause compared with component-level validation baselines. These results demonstrate that systematic validation of emergent cross-domain behavior can be incorporated as a continuous engineering activity rather than being deferred to late-stage system integration.

The principal novelty lies in the integration of dependency modeling, interface governance, temporal coordination, and scenario-based validation within a single engineering workflow. This architecture enables validation activities to remain aligned with increasingly frequent software changes and growing interaction complexity in SDV platforms. The framework also provides a structured basis for traceability and systematic verification in safety-critical development contexts, supporting its applicability to SDV programs requiring evidence-oriented integration and validation practices.

The framework provides a foundation for further development toward intelligent and increasingly autonomous validation. Machine learning-assisted scenario generation and defect classification can be incorporated to prioritize high-risk interaction patterns, while digital twin synchronization can extend validation toward predictive assessment before software changes reach physical integration environments. These extensions can be implemented incrementally on the established architectural foundation without changing its fundamental integration principles. Overall, the results indicate that treating cross-domain behavior as a governed and continuously validated engineering concern provides a scalable approach for improving the reliability, efficiency, and maintainability of SDV development.

References

[1] W. Wang, K. Guo, W. Cao, H. Zhu, J. Nan, and L. Yu, "Review of electrical and electronic architectures for autonomous vehicles: Topologies, networking and simulators," Automotive Innovation, vol. 7, pp. 82-101, 2024. https://doi.org/10.1007/s42154-023-00266-9

[2] H. Askaripoor, M. H. Farzaneh, and A. Knoll, "E/E architecture synthesis: Challenges and technologies," Electronics, vol. 11, no. 4, p. 518, 2022. https://doi.org/10.3390/electronics11040518

[3] A. Khamis and P. Goswami, "Rethinking vehicle architecture through softwarization and servitization," IEEE Access, vol. 13, pp. 126213-126226, 2025. https://doi.org/10.1109/ACCESS.2025.3588432

[4] A. Abdullah, R. Abu Bakar, F. Abdul Farid, and M. Abdulhak, "Safety function model for requirement specification in critical systems: A case study of generic patient controlled analgesia pump model (CGPA)," International Journal of Engineering, Science and Information Technology, vol. 5, no. 3, pp. 595-602, 2025. https://doi.org/10.52088/ijesty.v5i3.1370

[5] Y. Liu, H. Li, K. Ding, and Y. Gao, "Impact, challenges, and prospects of software-defined vehicles," Automotive Innovation, vol. 5, no. 2, pp. 180-194, 2022. https://doi.org/10.1007/s42154-022-00179-z

[6] D. F. Blanco, F. Le Mouël, T. Lin, and M.-P. Escudié, "A comprehensive survey on Software as a Service (SaaS) transformation for the automotive systems," IEEE Access, vol. 11, 2023. https://ieeexplore.ieee.org/document/10177956

[7] L. S. Ismail, A. S. Jamil, T. A. Mohammed Ali, I. H. M. Al-Dosari, K. Salman, and S. S. Maidin, "Edge computing frameworks for real-time optimisation in autonomous electric vehicle networks," International Journal of Engineering, Science and Information Technology, vol. 5, no. 3, pp. 554-563, 2025. https://doi.org/10.52088/ijesty.v5i3.1397

[8] H. Zhu, W. Zhou, Z. Li, L. Li, and T. Huang, "Requirements-driven automotive electrical/electronic architecture: A survey and prospective trends," IEEE Access, vol. 9, 2021. https://ieeexplore.ieee.org/document/9466854

[9] S. Raikwar, L. J. Wani, S. A. Kumar, and M. S. Rao, "Hardware-in-the-loop test automation of embedded systems for agricultural tractors," Measurement, vol. 133, 2019. https://www.sciencedirect.com/science/article/pii/S0263224118309369

[10] A. Mouzakitis, D. Copp, R. Parker, et al., "Hardware-in-the-loop system for testing automotive ECU diagnostic software," Measurement and Control, vol. 42, no. 8, 2009. https://doi.org/10.1177/002029400904200803

[11] M. Abboush, D. Bamal, C. Knieke, and A. Rausch, "Intelligent fault detection and classification based on hybrid deep learning methods for hardware-in-the-loop test of automotive software systems," Sensors, vol. 22, no. 11, p. 4066, 2022. https://doi.org/10.3390/s22114066

[12] H. Vdovic, J. Babic, and V. Podobnik, "Automotive software in connected and autonomous electric vehicles: A review," IEEE Access, vol. 7, pp. 175185-175198, 2019. https://doi.org/10.1109/ACCESS.2019.2951273

[13] G. Vitale and M. Hollander, "The software-defined vehicle: How to verify and validate software functions," in Proceedings of the International Stuttgart Symposium, Springer Vieweg, 2023. https://doi.org/10.1007/978-3-658-42048-2_27

[14] M. Abboush, C. Knieke, and A. Rausch, "Advancing real-time validation of automotive software systems via continuous integration and intelligent failure analysis," Scientific Reports, vol. 15, art. 32936, 2025. https://doi.org/10.1038/s41598-025-21416-5

[15] G. Balan, P. Neninger, E. Ruiz Zúñiga, E. Serea, et al., "A perspective on software-in-the-loop and hardware-in-the-loop within digital twin frameworks for automotive lighting systems," Applied Sciences, vol. 15, no. 15, p. 8445, 2025. https://doi.org/10.3390/app15158445
[16] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner, "Software engineering for automotive systems: A roadmap," in Proceedings of the IEEE International Conference on Software Engineering—Future of Software Engineering Track, pp. 55-71, 2007. https://ieeexplore.ieee.org/document/4221612
[17] Y. Zhao et al., "Review and prospect of integration compatibility in digital vehicles: Multi-dimensional challenges and industry practice," Machines, vol. 13, no. 9, p. 786, 2025. https://doi.org/10.3390/machines13090786
[18] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, and F. Diermeyer, "Survey on scenario-based safety assessment of automated vehicles," IEEE Access, vol. 8, pp. 87456-87477, 2020. https://doi.org/10.1109/ACCESS.2020.2993730
[19] L. Mauser and S. Wagner, "Centralization potential of automotive E/E architectures," Journal of Systems and Software, vol. 219, art. 112220, 2025. https://doi.org/10.1016/j.jss.2024.112220
[20] S. Kugele, P. Obergfell, and E. Sax, "Model-based resource analysis and synthesis of service-oriented automotive software architectures," Software and Systems Modeling, vol. 20, pp. 1945–1975, 2021. https://doi.org/10.1007/s10270-021-00896-9
[21] M. Ashjaei, L. Lo Bello, M. Daneshtalab, G. Patti, S. Saponara, and S. Mubeen, "Time-Sensitive Networking in automotive embedded systems: State of the art and research opportunities," Journal of Systems Architecture, vol. 117, art. 102137, 2021. https://doi.org/10.1016/j.sysarc.2021.102137
[22] M.-A. Meyer, M. Zouari, S. Bannenberg, M. Deppe, S. Christiaens, S.-Y. Lee, and J. Andert, "Machine-readable specification and intelligent cloud-based execution of logical test cases for automated driving functions," Automated Software Engineering, vol. 32, art. 10, 2025. https://doi.org/10.1007/s10515-024-00481-6
[23] F. Finkeldei, C. Thees, J. N. Weghorn, and M. Althoff, "Scenario Factory 2.0: Scenario-Based Testing of Automated Vehicles with CommonRoad," Automotive Innovation, vol. 8, no. 2, pp. 207–220, 2025. https://doi.org/10.1007/s42154-025-00360-0
[24] Ortega-Cabezas, P. M., Colmenar-Santos, A., Borge-Diez, D., & Blanes-Peiró, J. J. (2020). Application of rule-based expert systems in hardware-in-the-loop simulation case study: Software and performance validation of an engine electronic control unit. Journal of Software: Evolution and Process, 32(1), e2223. https://doi.org/10.1002/smr.2223
[25] Fremont, D. J., Kim, E., Pant, Y. V., Seshia, S. A., Acharya, A., Bruso, X., Wells, P., Lemke, S., Lu, Q., & Mehta, S. (2020). Formal scenario-based testing of autonomous vehicles: From simulation to the real world. In 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC) (pp. 1–8). IEEE. https://doi.org/10.1109/ITSC45102.2020.9294368
[26] Wang, Z., Ma, J., & Lai, E. M.-K. (2024). A survey of scenario generation for automated vehicle testing and validation. Future Internet, 16(12), 480. https://doi.org/10.3390/fi16120480
[27] Cai, J., Deng, W., Guang, H., Wang, Y., Li, J., & Ding, J. (2022). A survey on data-driven scenario generation for automated vehicle testing. Machines, 10(11), 1101. https://doi.org/10.3390/machines10111101
[28] Ma, J., Che, X., Li, Y., & Lai, E. M.-K. (2021). Traffic scenarios for automated vehicle testing: A review of description languages and systems. Machines, 9(12), 342. https://doi.org/10.3390/machines9120342
[29] Wiechowski, N., Chevalier, A., Stefan, F., Roettger, D., & Goebe, F. (2021). Automated hardware-in-the-loop testing using a cloud-based architecture. SAE Technical Paper 2021-01-0133. SAE International. https://doi.org/10.4271/2021-01-0133
[30] Yadav, V., & Bhade, N. (2024). Implementation of a hybrid hardware-in-the-loop (HIL) test system for automotive software validation and verification. SAE Technical Paper 2024-28-0252. SAE International. https://doi.org/10.4271/2024-28-0252
[31] Bowes, D., Hall, T., & Petrić, J. (2017). Software defect prediction: Do different classifiers find the same defects? Software Quality Journal, 25, 129–159. https://doi.org/10.1007/s11219-016-9353-3
[32] Riedmaier, S., Ponn, T., Ludwig, D., Schick, B., & Diermeyer, F. (2020). Survey on scenario-based safety assessment of automated vehicles. IEEE Access, 8, 87456–87477. https://doi.org/10.1109/ACCESS.2020.2993730
[33] Coe, D. J., Kulick, J. H., Milenkovic, A., & Etzkorn, L. H. (2020). Virtualized in situ software update verification: Verification of over-the-air automotive software updates. IEEE Vehicular Technology Magazine, 15(1), 84–90. https://doi.org/10.1109/MVT.2019.2954302
[34] Biagiola, M., Stocco, A., Riccio, V., & Tonella, P. (2024). Two is better than one: Digital siblings to improve autonomous driving testing. Empirical Software Engineering, 29, 72. https://doi.org/10.1007/s10664-024-10458-4
[35] Stang, M., et al. (2024). Scenario-based testing of automotive self-learning systems using metamorphic relations. Proceedings of the ACM. https://doi.org/10.1145/3632366.3632383
[36] Yadav, S., & Bhade, A. (2024). Implementation of a hybrid hardware-in-the-loop (HIL) test system for automotive software validation and verification. SAE Technical Paper. https://doi.org/10.4271/2024-28-0252
[37] Wiechowski, A., et al. (2021). Automated hardware-in-the-loop testing using a cloud-based architecture. SAE Technical Paper. https://doi.org/10.4271/2021-01-0133
[38] Shoukat, A., et al. (2024). Digital twin-based approaches for connected and autonomous vehicle testing and validation. Internet of Things, 27, 101301. https://doi.org/10.1016/j.iot.2024.101301
[39] Biagiola, M., Stocco, A., Riccio, V., & Tonella, P. (2024). Two is better than one: Digital siblings to improve autonomous driving testing. Empirical Software Engineering, 29, 72. https://doi.org/10.1007/s10664-024-10458-4
[40] Duan, X., et al. (2023). Digital twin test method for autonomous vehicles based on PanoSim. SAE Technical Paper. https://doi.org/10.4271/2023-01-7056
[41] Stang, M., et al. (2024). Scenario-based testing of automotive self-learning systems using metamorphic relations. Proceedings of the ACM. https://doi.org/10.1145/3632366.3632383
[42] Wynn-Williams, S., Tyrrell, R., Pantelic, V., Lawford, M., Menghi, C., Nalla, P., & Artail, H. (2025). Can generative AI produce test cases? An experience from the automotive domain. Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, 456–467. https://doi.org/10.1145/3696630.3728568