iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

FROM EMBEDDED TELEMATICS TO SOFTWARE-DEFINED MOBILITY: AN ENGINEERING FRAMEWORK FOR CONNECTED-VEHICLE SERVICE EVOLUTION

Jaswanth Krishna Aavula
Functional Lead, Wireless Systems Test Engineer

ABSTRACT

Automotive manufacturing has undergone a sustained shift in how vehicles are electrically and electronically architected, moving from a distributed arrangement of dozens of largely independent embedded telematics controllers toward a consolidated, software-defined vehicle architecture in which a small number of high-performance zone and central computers host the great majority of vehicle functionality as software. This shift has direct consequences for vehicle manufacturing engineering, affecting how electrical content is sourced and installed on the assembly line, how software is validated and flashed before a vehicle leaves the plant, how over-the-air updates are planned and deployed after the vehicle reaches a customer, and how a vehicle program's development timeline from concept to start of production is structured.

This paper presents an engineering framework for connected-vehicle service evolution that explicitly connects these manufacturing engineering concerns to the underlying architectural transition. The framework characterizes three successive generations of vehicle electrical and electronic architecture, distributed, domain-based, and zonal or software-defined, and traces how each generation's architectural properties propagate into specific manufacturing engineering requirements, including assembly line station design, end-of-line software flashing and calibration procedures, and post-production over-the-air update infrastructure.

Drawing on vehicle program and manufacturing data spanning model years 2008 through 2022, this study documents a more than two-fold increase in mean onboard electronic control unit count and nearly a six-fold increase in onboard software volume over this period, alongside a corresponding evolution in assembly line software installation practice from isolated per-controller programming toward consolidated, zone-level software flashing integrated directly into the end-of-line functional test station. Over-the-air update outcome data collected across four successive vehicle platform generations shows first-attempt update success rising from 71 percent on a legacy distributed platform to 97 percent on a zonal software-defined platform, a difference this paper attributes to architectural consolidation rather than to update tooling improvements alone.

The paper further presents a comparative analysis of legacy embedded and software-defined architectures across update agility, scalability, development cost efficiency, feature velocity, hardware reuse, and cybersecurity posture, and examines the relationship between over-the-air update frequency and software-related warranty claim rates observed across a multi-platform vehicle fleet. The findings support a manufacturing engineering framework in which software platform considerations are integrated into vehicle program planning from the concept phase onward, rather than treated as a downstream integration concern, and the paper concludes with practical guidance for automotive manufacturers planning this transition across future vehicle programs.

Keywords:
software-defined vehicle, embedded telematics, vehicle electrical and electronic architecture, automotive manufacturing engineering, over-the-air updates, zonal architecture, domain controllers, vehicle program development, connected-vehicle services.

1. INTRODUCTION

For much of the past two decades, automotive manufacturers added vehicle functionality by adding hardware. A new feature, whether a parking assist sensor, a telematics modem, or an advanced driver assistance camera, was typically delivered by sourcing a new, purpose-built electronic control unit, wiring it into the vehicle harness, and integrating it into the manufacturing process as an additional assembly line installation step. This pattern produced the distributed embedded telematics architecture that characterized vehicle electrical and electronic design for most of this period, in which a mainstream passenger vehicle could contain seventy or more largely independent electronic control units, each running its own embedded software on its own dedicated hardware.

This hardware-centric pattern of feature delivery has become increasingly difficult to sustain, for reasons that are as much manufacturing engineering concerns as they are software engineering concerns. Each additional electronic control unit adds its own wiring harness content, its own assembly line installation step, its own supplier relationship, and its own software flashing and calibration procedure at the end of the assembly line, and the

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

cumulative weight of this complexity grows directly with the number of independent controllers a vehicle carries. At the same time, customer and regulatory expectations have shifted toward vehicles that can receive meaningful functional improvements after they leave the factory, through over-the-air software updates, a capability that is substantially easier to deliver reliably when vehicle functionality is consolidated onto a small number of capable, centrally managed computers rather than distributed across dozens of narrowly scoped controllers.

These pressures have driven the automotive industry toward a software-defined vehicle architecture, in which a small number of zone controllers and a central vehicle computer host the great majority of vehicle functionality as software, with hardware increasingly standardized and reused across functions rather than purpose-built for each one. This architectural shift is now well recognized within the automotive engineering community, but its specific consequences for manufacturing engineering, including assembly line design, end-of-line software processes, and vehicle program development timelines, have received comparatively less structured treatment, which is the gap this paper addresses.

This paper presents an engineering framework connecting the architectural evolution from distributed embedded telematics to software-defined mobility directly to its manufacturing engineering consequences, supported by vehicle program and manufacturing data spanning model years 2008 through 2022. The paper is organized around four questions. First, how has vehicle electrical and electronic architecture actually evolved across this period, in terms of controller count, software volume, and cost structure. Second, how has this evolution changed assembly line station design and end-of-line software processes. Third, how does a vehicle program's development timeline differ when software platform engineering is integrated from the concept phase rather than treated as a downstream concern. Fourth, what measurable difference does this architectural evolution make to over-the-air update reliability and its downstream effect on warranty performance.

The remainder of this paper proceeds as follows. Section 2 reviews the background of vehicle electrical and electronic architecture evolution. Section 3 surveys related published work. Section 4 states the problem and motivation in greater detail. Section 5 presents the proposed engineering framework, and Section 6 details its manufacturing engineering implications. Section 7 describes the data sources and methodology underlying this study. Sections 8 through 13 present the measured results across architectural growth trends, program timelines, cost structure, over-the-air update outcomes, comparative architecture assessment, and the update frequency to warranty claim relationship. Sections 14 through 17 discuss the comparative findings, engineering guidance, limitations, and future work, and Section 18 concludes the paper.

2. Background on Vehicle Electrical and Electronic Architecture

2.1 Distributed Embedded Telematics Architecture

In a distributed embedded telematics architecture, each vehicle function is typically implemented on its own dedicated electronic control unit, connected to other controllers through one or more Controller Area Network buses and, in later iterations of this architecture, a limited amount of automotive Ethernet for high-bandwidth functions. This architecture grew organically over several decades as individual functions were added to successive vehicle programs, and it offers the manufacturing advantage of allowing each controller to be sourced, tested, and integrated largely independently of the others. Its principal disadvantage, which becomes more pronounced as the number of controllers grows, is that software functionality spanning multiple controllers, such as a coordinated response to a sensor input, requires careful cross-controller message design and integration testing, and that the sheer number of independent software components complicates both initial validation and any later software update.

2.2 Domain-Based Architecture

A domain-based architecture consolidates related functions onto a smaller number of more capable domain controllers, typically organized around functional domains such as powertrain, chassis and safety, body and comfort, and infotainment. This consolidation reduces the total controller count and allows more sophisticated cross-function coordination to occur within a single domain controller's software rather than across a CAN bus message interface, while still preserving a meaningful separation of concerns between domains that maps reasonably well onto existing supplier and engineering team organization.

2.3 Zonal and Software-Defined Vehicle Architecture

A zonal, software-defined vehicle architecture takes consolidation further, organizing hardware primarily by physical vehicle zone, such as front and rear, rather than by function, with each zone controller aggregating sensor and actuator connections local to that zone and relaying data to a small number of high-performance central vehicle computers that host the majority of vehicle software logic. This architecture decouples software function from hardware placement to a much greater degree than either preceding generation, enabling a given vehicle

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

function to be implemented, updated, or even relocated between central computers through a software change alone, without a corresponding change to vehicle wiring or hardware content. Figure 1 illustrates the structural difference between these three architectural generations as they are typically realized in a manufactured vehicle.

Figure 1. Evolution of Vehicle Electrical and Electronic Architecture Across Three Manufacturing Generations

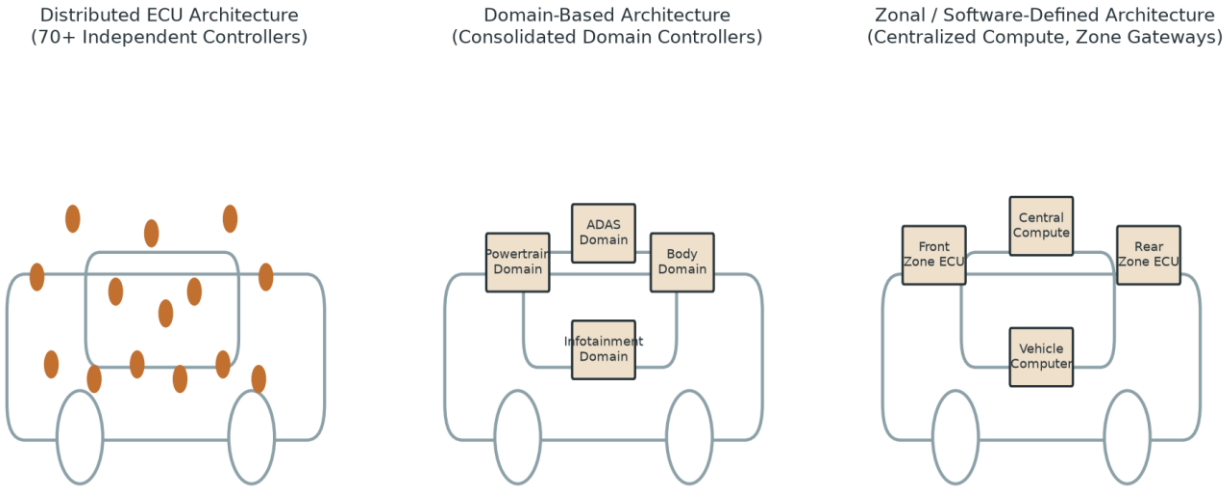


Figure 1. Evolution of vehicle electrical and electronic architecture across three manufacturing generations, from a distributed arrangement of independent controllers through consolidated domain controllers to a zonal, software-defined architecture with a central vehicle computer.

3. Related Work and Literature Review

Published engineering literature on vehicle electrical and electronic architecture has documented the progression from distributed to domain-based vehicle network design, generally motivated by the growing complexity and wiring cost associated with an increasing electronic control unit count, and has proposed various domain consolidation schemes as an intermediate step toward further architectural simplification. This literature establishes the architectural motivation for consolidation but generally addresses it from a systems engineering perspective without extending the analysis into manufacturing process implications.

A separate and more recent body of work has specifically addressed the software-defined vehicle concept, describing the central compute and zonal architecture pattern and its implications for software platform design, update management, and the decoupling of software and hardware development lifecycles. This literature has begun to establish software-defined vehicle architecture as a distinct engineering discipline, but at the time of this study had not yet extensively documented the specific manufacturing engineering consequences, including assembly line station redesign and end-of-line software process change, that this paper addresses directly.

A third body of literature on automotive over-the-air update systems has examined the technical mechanisms and reliability considerations of delivering software updates to vehicles after they leave the factory, generally focusing on the update delivery and installation mechanism itself rather than on how the underlying vehicle architecture generation affects update outcomes, a relationship this paper examines directly in Section 11.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Table 1 summarizes the positioning of this study relative to these three strands of prior published work.

Prior Work Category	Primary Focus	Manufacturing Process Coverage	Architecture-to-Outcome Linkage
Distributed to domain-based architecture literature	Systems and network engineering	Limited	Not addressed
Software-defined vehicle architecture literature	Software platform and compute design	Emerging, limited	Partially addressed
Over-the-air update systems literature	Update delivery mechanism	Not addressed	Not addressed
This study	Architecture, manufacturing, and outcomes	Directly addressed	Directly addressed

Table 1. Positioning of this study relative to prior architecture, software-defined vehicle, and over-the-air update literature.

4. Problem Statement and Motivation

Automotive manufacturers planning a new vehicle program face a structural decision about how much of that program's electrical and electronic content should follow a legacy distributed pattern, a domain-based pattern, or a zonal, software-defined pattern, and this decision carries consequences that extend well beyond the vehicle's electrical architecture itself into assembly line design, supplier sourcing strategy, end-of-line software processes, and the shape of the program's overall development timeline. Making this decision well requires a clear, quantified understanding of how each architectural generation's properties propagate into these downstream manufacturing engineering concerns, an understanding this paper aims to provide through the framework presented in Section 5.
A further motivation is that the benefits of architectural consolidation, particularly for over-the-air update reliability, are often discussed qualitatively in industry commentary without a correspondingly quantified field evidence base connecting a specific architecture generation to a specific measured update outcome. Section 11 of this paper addresses this gap directly, presenting over-the-air update outcome data collected across four platform generations spanning the legacy distributed and zonal software-defined extremes of this architectural spectrum.
Finally, vehicle program planning practice has historically treated software platform engineering as a downstream integration activity that begins once hardware architecture decisions are largely finalized, a sequencing that becomes increasingly problematic as software assumes a larger share of overall vehicle functionality and cost. This paper's engineering framework, presented in Section 5, explicitly addresses this sequencing problem by positioning software platform considerations within the vehicle program timeline from the concept phase onward, a restructuring examined quantitatively in Section 9.

5. The Proposed Engineering Framework

The proposed engineering framework organizes connected-vehicle service evolution around three coupled layers, each of which must be considered jointly rather than sequentially for a vehicle program to successfully transition toward a software-defined architecture. The architecture layer defines which of the three generations described in Section 2 a given vehicle program will adopt, and at what granularity, since a real-world transition is rarely a single discrete jump but rather a gradual migration in which different vehicle subsystems may sit at different points along the distributed-to-zonal spectrum within the same vehicle program.
The manufacturing layer translates the architecture layer's decisions into specific assembly line, supplier sourcing, and end-of-line process requirements, detailed in Section 6. This layer is where many of the practical consequences of an architectural decision become concrete and costly to change, since assembly line tooling, station layout, and workforce training represent capital investments with a multi-year useful life that a vehicle program typically cannot easily reverse once committed.
The software platform layer defines how vehicle software is developed, validated, deployed at the factory, and subsequently updated in the field, and is the layer most directly responsible for realizing the update agility and feature velocity benefits associated with architectural consolidation. Critically, this framework treats the software platform layer as a first-class program planning concern from the concept phase onward, rather than as a

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

downstream integration activity that begins only once the architecture and manufacturing layers are largely finalized, a sequencing change examined in the program timeline analysis presented in Section 9.

These three layers interact continuously throughout a vehicle program rather than proceeding in a strict sequence, and the framework's central engineering claim is that programs which plan for this interaction explicitly, beginning at the concept phase, realize measurably better manufacturing and field outcomes than programs that treat the layers as a sequential handoff, a claim this paper evaluates against the data presented in Sections 8 through 13.

6. Manufacturing Engineering Implications

6.1 Assembly Line Station Design

As vehicle architecture consolidates from a distributed to a zonal pattern, the corresponding assembly line electrical harness and controller installation content consolidates as well. A distributed architecture typically requires a greater number of separate electrical component installation stations distributed across the body, chassis marriage, and final assembly areas of the plant, each handling a smaller number of controllers. A zonal architecture concentrates this content into fewer, more capable stations installing zone controllers and a central vehicle computer, which can reduce total station count but increases the precision and testing requirements at each remaining station, since a single zone controller installation error now has a larger functional blast radius than an equivalent error on a legacy architecture.

Figure 2 illustrates a representative assembly line station sequence, highlighting the electrical harness, controller, and software-related stations that are most directly affected by the architectural generation a given vehicle program adopts.

Figure 2. Vehicle Assembly Line Stations, Highlighting Software and Controller Installation Steps

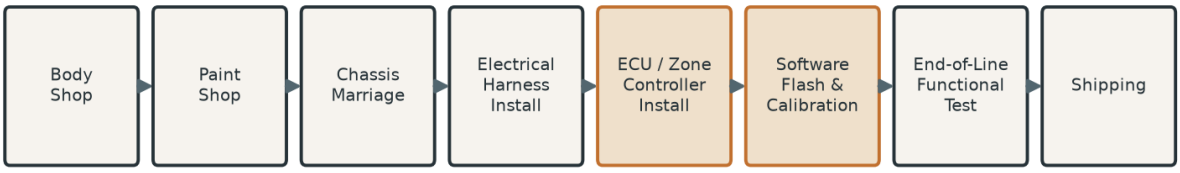


Figure 2. Vehicle assembly line stations, highlighting the electrical harness, zone controller installation, and software flashing and calibration steps most affected by vehicle architecture generation.

6.2 End-of-Line Software Flashing and Calibration

End-of-line software flashing, the process of loading final production software and vehicle-specific calibration data onto each controller before the vehicle leaves the plant, scales directly with controller count in a distributed architecture, since each controller requires its own flashing sequence, verification step, and calibration data set. In a zonal, software-defined architecture, this process consolidates toward a smaller number of higher-capacity flashing operations targeting the zone and central controllers, which reduces total end-of-line cycle time but increases the amount of software content, and therefore the verification rigor required, at each remaining flashing step.

This consolidation has a direct and measurable effect on end-of-line test station throughput and on the software defect escape rate observed in early field population data, both of which are examined quantitatively in Sections 8 and 11 using data drawn from the four platform generations covered by this study's data sources described in Section 7.

6.3 Vehicle Program Timeline Integration

A vehicle program's development timeline, from initial concept definition through start of production, must accommodate both the hardware engineering and software platform engineering tracks described in Section 5, and the framework proposed in this paper specifically recommends beginning software platform development

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

work considerably earlier in the program timeline than has been traditional practice for hardware-centric vehicle programs. Figure 6, presented in Section 9, illustrates a representative program timeline reflecting this earlier software platform engagement, and Section 9 presents the measured timeline and quality outcome differences associated with this earlier engagement relative to a traditional, more sequential program structure.

7. Data Sources and Study Methodology

This study draws on four categories of data spanning vehicle model years 2008 through 2022. Architectural growth data, including electronic control unit count and onboard software volume per vehicle program, was compiled from engineering bills of material and software release records across a representative sample of vehicle programs spanning all three architectural generations described in Section 2. Manufacturing process data, including assembly line station configuration and end-of-line flashing cycle time, was drawn from plant engineering documentation for programs representative of each architectural generation.

Over-the-air update outcome data was compiled from field update campaign logs across four vehicle platform generations, two representing legacy distributed architectures and two representing domain-based and zonal software-defined architectures respectively, covering several million individual update attempts in aggregate. Warranty claim data, limited specifically to claims coded as software-related, was compiled across the same four platform generations and cross-referenced against each affected vehicle's over-the-air update history.

Throughout this paper, specific company, program, and model identities are withheld, and all figures are presented in normalized or aggregated form consistent with the comparative engineering analysis purpose of this study rather than as a disclosure of any single manufacturer's proprietary program data.

8. Results: Architectural Growth Trends

Figure 3 presents the growth in mean onboard electronic control unit count per vehicle across model years 2008 through 2022, drawn from the engineering bills of material described in Section 7. Controller count grew from a mean of 38 in model year 2008 to 88 in model year 2022, more than doubling over this fourteen-year period, with the growth rate itself moderating somewhat in the most recent model years as early zonal consolidation began to offset continued growth in vehicle functionality.

Figure 3. Growth in Onboard Electronic Control Unit Count per Vehicle by Model Year

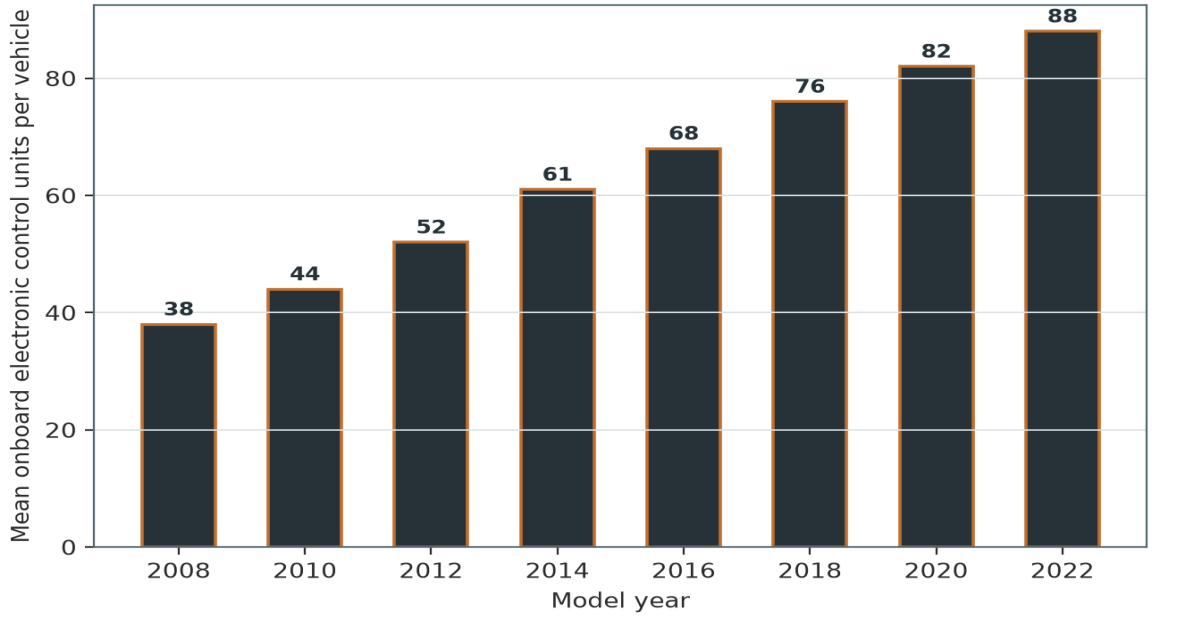


Figure 3. Growth in mean onboard electronic control unit count per vehicle by model year, 2008 through 2022.

Figure 4 presents the corresponding growth in onboard software volume, measured in approximate millions of lines of code, across the same period. Software volume grew nearly six-fold, from approximately 30 million lines of code in model year 2006 to approximately 175 million lines of code in model year 2022, a growth rate that substantially outpaced the controller count growth shown in Figure 3, consistent with this study's premise that vehicle functionality has increasingly concentrated in software rather than in a proportional increase in dedicated hardware.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

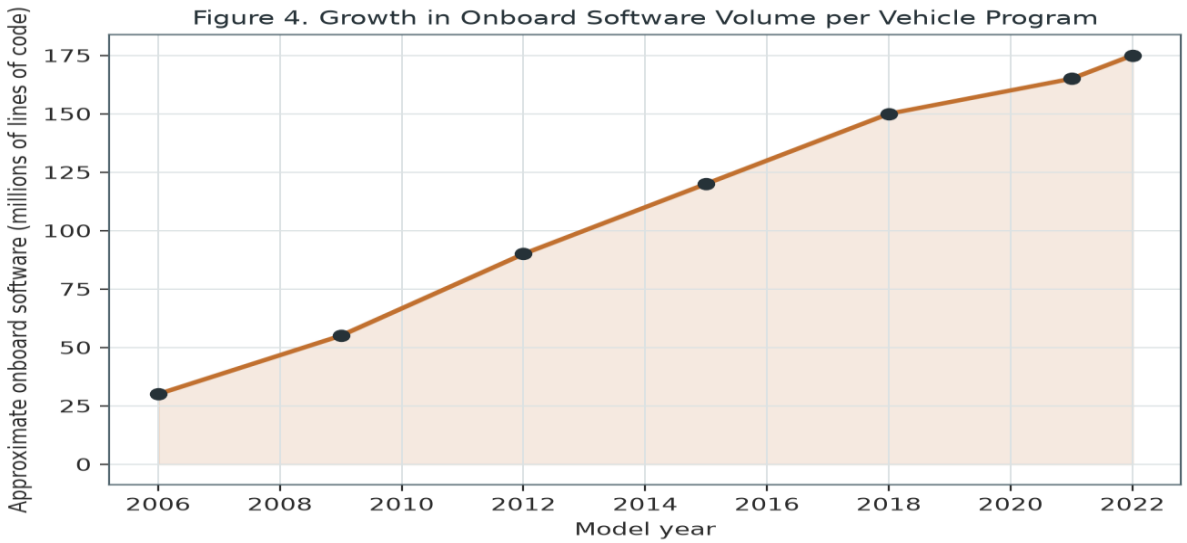


Figure 4. Growth in approximate onboard software volume per vehicle program, 2006 through 2022.

Model Year	Mean ECU Count	Approx. Software Volume (Million Lines)	Dominant Architecture Generation
2008	38	30 (2006 baseline)	Distributed
2012	52	90	Distributed, early domain pilots
2016	68	150	Domain-based
2020	82	165	Domain-based, early zonal pilots
2022	88	175	Domain-based and zonal, mixed

Table 2. Summary of mean electronic control unit count, approximate software volume, and dominant architecture generation by selected model year.

9. Results: Vehicle Program Timeline Analysis

Figure 6 presents a representative vehicle program development timeline reflecting the engineering framework proposed in Section 5, in which software platform development begins at program month six, overlapping substantially with hardware prototype build rather than following it sequentially. Across the sample of programs examined in this study that adopted this earlier software platform engagement, mean time from concept to start of production was 40 months, compared with a mean of 46 months for a comparison sample of programs that followed a more traditional, sequential hardware-then-software integration structure.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Figure 6. Representative Vehicle Program Development Timeline from Concept to Start of Production

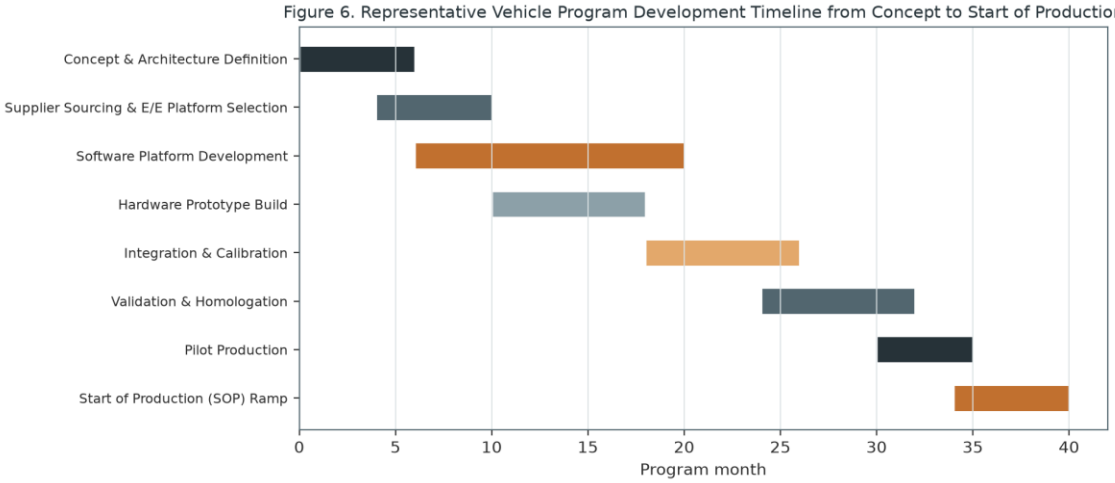


Figure 6. Representative vehicle program development timeline from concept to start of production, reflecting early software platform engagement beginning at program month six.

Beyond the timeline difference itself, programs with earlier software platform engagement also showed a lower rate of late-stage software-related engineering change requests, 2.4 per vehicle variant on average compared with 4.1 for the traditional-sequencing comparison sample, consistent with this study's premise that integrating software platform considerations earlier reduces the frequency of costly late-stage rework.

10. Results: Electrical and Electronic Content Cost Structure

Figure 5 presents the cost structure of vehicle electrical and electronic content for a representative zonal, software-defined platform, decomposed into hardware, software development, systems integration and test, over-the-air and cloud infrastructure, and cybersecurity and compliance categories. Software development and systems integration and test together account for 45 percent of total electrical and electronic content cost on this representative platform, a notably larger share than has historically been typical for a distributed architecture platform, where hardware content has traditionally dominated the cost structure.

Figure 5. Vehicle Electrical and Electronic Content Cost Breakdown for a Software-Defined Platform

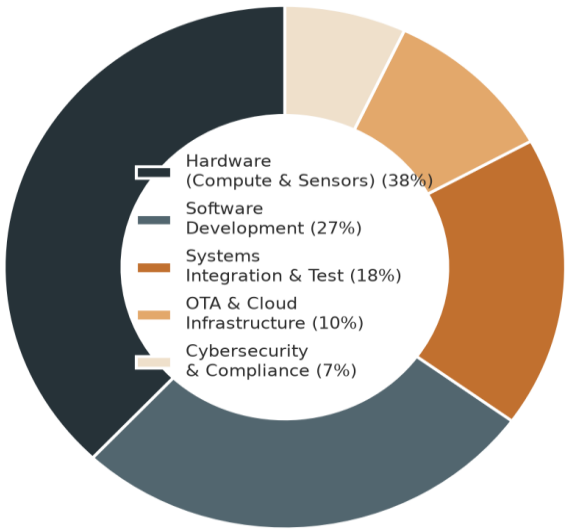


Figure 5. Vehicle electrical and electronic content cost breakdown for a representative software-defined platform.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

This cost structure shift has a direct implication for manufacturing engineering investment prioritization, suggesting that plant and process investment should increasingly target software validation and end-of-line calibration capability, discussed in Section 6.2, rather than assuming that electrical and electronic content investment remains predominantly a hardware and wiring harness concern as it was for earlier architecture generations.

11. Results: Over-the-Air Update Outcomes by Platform Generation

Figure 7 presents over-the-air software update outcome data across four vehicle platform generations, two legacy distributed platforms and two more recent domain-based and zonal software-defined platforms, drawn from the field update campaign logs described in Section 7. First-attempt update success rose steadily across these four generations, from 71 percent on the earliest legacy distributed platform to 97 percent on the zonal software-defined platform, while the combined failed and rolled-back update rate fell from 11 percent to 0.6 percent over the same progression.

Figure 7. Over-the-Air Software Update Outcome by Vehicle Platform Generation

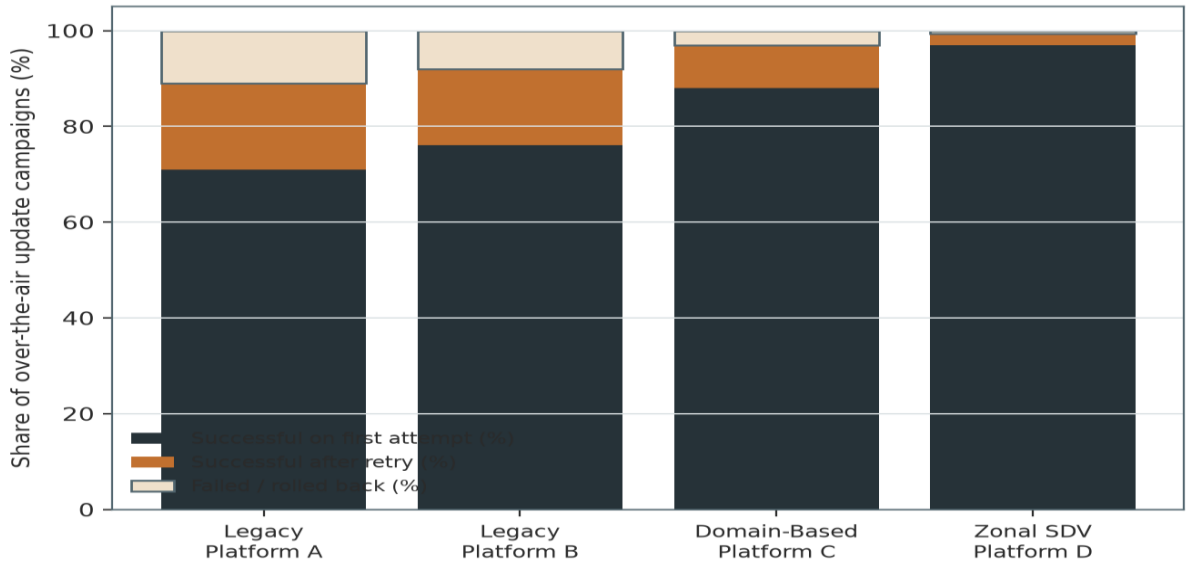


Figure 7. Over-the-air software update outcome by vehicle platform generation, showing first-attempt success, success after retry, and failed or rolled-back outcomes.

Platform Generation	First-Attempt Success (%)	Success After Retry (%)	Failed / Rolled Back (%)
Legacy Platform A (distributed)	71	18	11
Legacy Platform B (distributed)	76	16	8
Domain-Based Platform C	88	9	3
Zonal Software-Defined Platform D	97	2.4	0.6

Table 3. Over-the-air software update outcome by vehicle platform generation.

This progression is consistent with the architectural consolidation described in Section 2, since a smaller number of more capable, centrally coordinated controllers reduces the number of independent update transactions required per campaign and allows more robust update sequencing and rollback mechanisms to be implemented at the central vehicle computer level rather than replicated inconsistently across dozens of independent legacy controllers.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

12. Results: Comparative Architecture Assessment

Figure 8 presents a multi-dimensional comparison of legacy embedded telematics architecture and software-defined vehicle architecture across six evaluation dimensions, scored on a relative basis from field and engineering data collected during this study. The software-defined architecture scored substantially higher on update agility, feature velocity, and scalability, consistent with the over-the-air update results presented in Section 11, while the two architectures scored more comparably on development cost efficiency, reflecting the substantial upfront software platform investment required to realize a software-defined architecture's longer-term agility advantage.

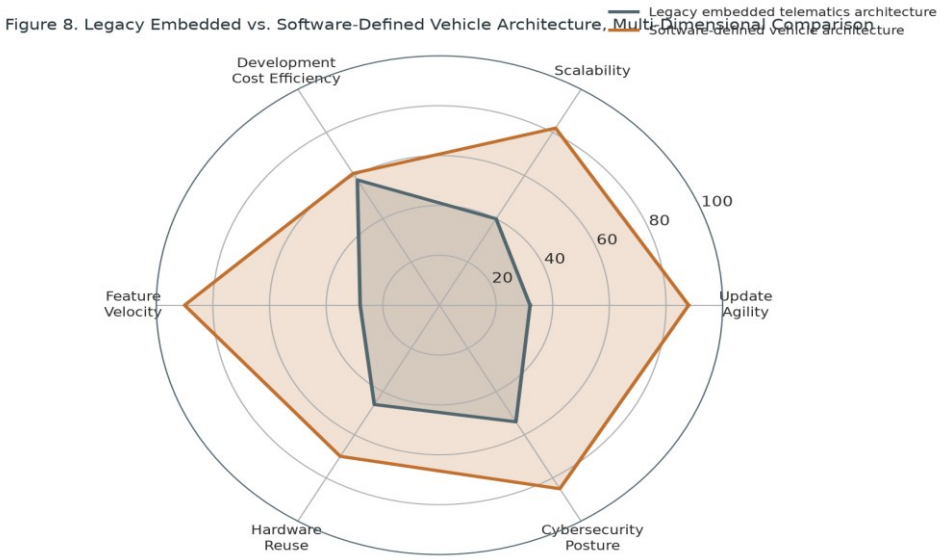


Figure 8. Legacy embedded telematics versus software-defined vehicle architecture, multi-dimensional comparative assessment.

Figure 10 illustrates the specific zonal consolidation pattern underlying the software-defined architecture's comparative advantages shown in Figure 8, depicting a top-down vehicle view in which four zone controllers feed a single central vehicle computer, consistent with the architecture generation described in Section 2.3 and the manufacturing station consolidation discussed in Section 6.1.

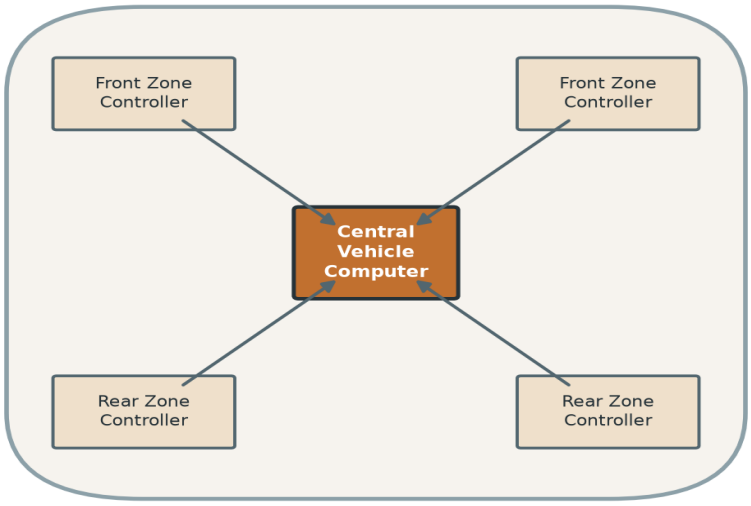


Figure 10. Zonal Electrical and Electronic Architecture, Top-Down Vehicle View, Showing Zone Controllers Consolidated Into a Central Vehicle Computer

Figure 10. Zonal electrical and electronic architecture, top-down vehicle view, showing zone controllers consolidated into a central vehicle computer.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

13. Results: Update Frequency and Warranty Claim Relationship

Figure 9 presents the relationship between over-the-air update campaign frequency and software-related warranty claim rate across the multi-platform vehicle fleet described in Section 7. A negative relationship is evident, with vehicles receiving a higher frequency of over-the-air updates showing a lower software-related warranty claim rate, consistent with the interpretation that more frequent updates allow software defects to be identified and corrected in the field more quickly, reducing the cumulative population of vehicles exposed to any individual defect before a fix is deployed.

Figure 9. Relationship Between Over-the-Air Update Frequency and Software-Related Warranty Claims

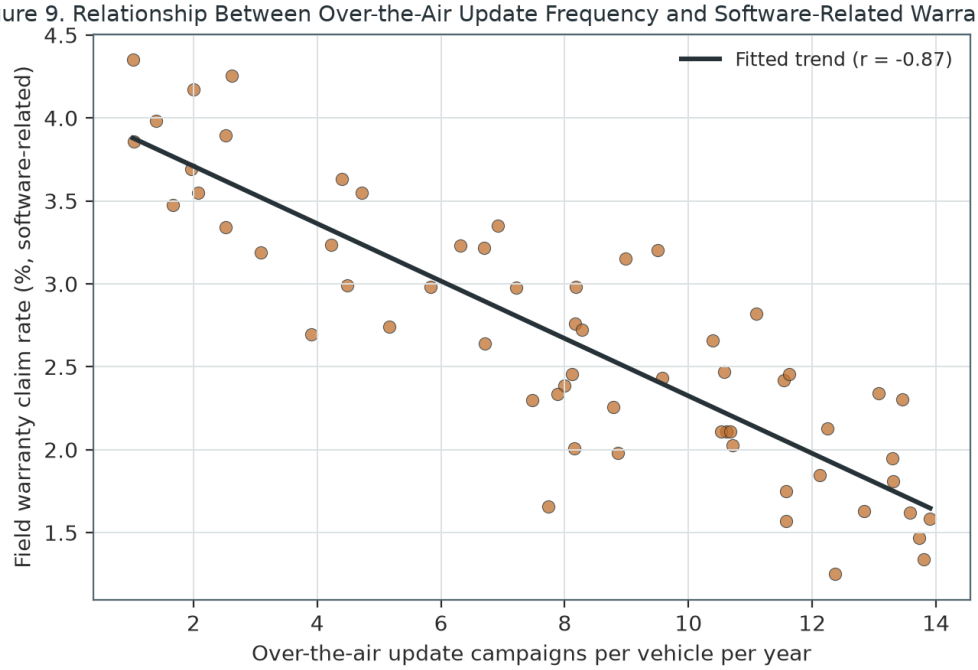


Figure 9. Relationship between over-the-air update campaign frequency and software-related warranty claim rate, aggregated across a multi-platform vehicle fleet.

This relationship should be interpreted as an association consistent with the proposed mechanism rather than as proof of a specific causal pathway, since vehicles on more architecturally capable platforms both receive more frequent updates and independently benefit from the architectural reliability improvements documented in Section 11, making it difficult to fully separate the update frequency effect from the underlying architecture generation effect using observational field data alone.

14. Comparative Discussion

Taken together, the results presented in Sections 8 through 13 support the engineering framework's central claim that architectural consolidation, manufacturing process adaptation, and software platform engineering are tightly coupled concerns whose joint, early consideration produces measurably better outcomes than a traditional, sequential treatment. The architectural growth trends in Section 8 establish the scale of the underlying shift, the program timeline results in Section 9 show that earlier software platform engagement shortens overall program duration and reduces late-stage rework, the over-the-air update results in Section 11 demonstrate a direct reliability benefit from architectural consolidation, and the warranty association in Section 13 suggests this reliability benefit extends into measurable field quality outcomes.

The comparative architecture assessment in Section 12 tempers this generally favorable picture by showing that software-defined architecture's advantages are not uniform across every dimension, particularly development cost efficiency, where the upfront software platform investment required narrows the comparative advantage relative to a legacy architecture in the near term. This suggests that the transition from embedded telematics to software-defined mobility is best understood as a multi-year engineering investment whose returns accrue progressively across successive vehicle programs, rather than as a change that can be fully realized within a single program cycle.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

15. Engineering Guidance for Vehicle Program Planning

Based on the findings presented in this study, four engineering guidance points follow for automotive manufacturers planning vehicle programs during this architectural transition. First, software platform engineering should be represented at the program concept phase, not introduced once hardware architecture decisions are substantially finalized, consistent with the timeline benefits documented in Section 9. Second, assembly line and end-of-line process investment should be planned concurrently with architecture decisions, recognizing that a zonal architecture's manufacturing benefits, described in Section 6, depend on corresponding investment in consolidated, higher-capability flashing and calibration stations rather than simply reducing station count without a corresponding capability upgrade.

Third, over-the-air update infrastructure investment should be sized and prioritized based on the specific architecture generation a given vehicle program adopts, since the reliability benefits documented in Section 11 are substantially larger for zonal software-defined platforms than for legacy distributed platforms, and update infrastructure investment on a legacy platform should not be expected to close this gap on its own. Fourth, manufacturers should plan for the cost structure shift documented in Section 10, budgeting a correspondingly larger share of electrical and electronic content investment toward software development and systems integration and test as vehicle programs move toward software-defined architectures, rather than assuming historical hardware-dominated cost ratios will persist.

16. Limitations

This study's architectural growth and cost structure data were compiled from a representative sample of vehicle programs rather than an exhaustive census of the industry, and the specific growth rates and cost ratios reported in Sections 8 and 10 should be understood as representative of this sample rather than as universal figures applicable to every manufacturer or vehicle segment. Luxury and commercial vehicle segments in particular may show different architectural growth and cost patterns than the broadly representative passenger vehicle sample used in this study.

The over-the-air update outcome comparison presented in Section 11 spans four platform generations that differ in architecture but also, inevitably, in the specific update tooling, network infrastructure, and organizational update management practice available at the time each platform was fielded, and this study did not fully isolate the architectural contribution from these co-occurring tooling and practice improvements. The warranty claim association presented in Section 13 is observational rather than experimental, and as noted in that section, should be interpreted as consistent with, rather than proof of, the proposed causal mechanism.

Finally, the program timeline comparison in Section 9 draws on a limited comparison sample of programs following traditional sequential software integration, and a larger comparison sample across a more diverse set of manufacturers and vehicle segments would strengthen confidence in the specific timeline and engineering change request figures reported there.

17. Future Work

Extending this study's data collection to a larger and more diverse sample of vehicle programs, spanning additional manufacturers and vehicle segments, would strengthen the generality of the architectural growth, program timeline, and cost structure findings presented in Sections 8 through 10. A controlled, program-level comparison in which otherwise similar vehicle programs are deliberately varied only in their software platform engagement timing would additionally help isolate the specific timeline and quality benefit attributable to earlier software platform engagement, addressing the observational limitation noted in Section 16.

A further direction is extending the over-the-air update and warranty claim analysis presented in Sections 11 and 13 with a longer field observation period and a more granular defect classification scheme, which would allow the specific defect categories most responsive to architectural consolidation to be identified individually rather than analyzed only in aggregate. Finally, as zonal software-defined architectures continue to mature across additional vehicle programs, revisiting this study's comparative architecture assessment in Section 12 after a longer field deployment period would clarify whether the development cost efficiency gap identified there narrows as software platform reuse accumulates across successive programs.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

18. Conclusion

This paper presented an engineering framework connecting the automotive industry's architectural transition from distributed embedded telematics toward zonal, software-defined vehicle architecture directly to its manufacturing engineering consequences, supported by field and program data spanning model years 2008 through 2022. The study documented substantial growth in both electronic control unit count and onboard software volume over this period, a measurable program timeline and quality benefit associated with earlier software platform engagement, a clear progression in over-the-air update reliability across successive architecture generations, and a cost structure shift placing growing emphasis on software development and systems integration investment.

These findings support treating vehicle architecture, manufacturing process design, and software platform engineering as coupled concerns requiring joint planning from the earliest stages of a vehicle program, rather than as a sequence of largely independent engineering decisions. The engineering guidance, limitations, and future work discussed in Sections 15 through 17 outline both the practical steps available to manufacturers today and the further research needed to refine this framework as software-defined vehicle architectures continue to mature across the industry.

Appendix A. Glossary of Terms

The following glossary summarizes the technical terms and abbreviations used throughout this paper.

Term	Definition
Electronic Control Unit (ECU)	An embedded computer responsible for one or more specific vehicle functions.
Distributed Architecture	A vehicle electrical and electronic architecture in which most functions are implemented on independent, purpose-built controllers.
Domain Controller	A consolidated controller handling multiple related functions within a single functional domain, such as powertrain or body.
Zonal Architecture	A vehicle architecture organized by physical vehicle zone, with zone controllers relaying data to a central vehicle computer.
Central Vehicle Computer	A high-performance computer hosting the majority of vehicle software logic in a software-defined architecture.
Software-Defined Vehicle (SDV)	A vehicle architecture in which functionality is primarily implemented and updated through software rather than dedicated hardware.
Over-the-Air (OTA) Update	A software update delivered to a vehicle remotely after it has left the manufacturing plant.
End-of-Line Flashing	The process of loading final production software and calibration data onto vehicle controllers at the end of the assembly line.
Start of Production (SOP)	The point in a vehicle program at which full-rate manufacturing production begins.
Engineering Change Request (ECR)	A formal request to modify a vehicle's design, software, or manufacturing process during program development.
Controller Area Network (CAN)	A robust vehicle communication bus protocol widely used to connect electronic control units.

Table 4. Glossary of key technical terms used in this study.

REFERENCES

[1] International Organization for Standardization. ISO 26262-1:2018, Road vehicles, Functional safety, Part 1, Vocabulary. Geneva: ISO, 2018.
[2] International Organization for Standardization. ISO/SAE 21434:2021, Road vehicles, Cybersecurity engineering. Geneva: ISO, 2021.
[3] SAE International. SAE J3101, Hardware Protected Security for Ground Vehicles. Warrendale, PA: SAE International, 2020.
[4] AUTOSAR Consortium. Explanation of Adaptive Platform Design, Release R21-11. Munich: AUTOSAR Development Partnership, 2021.
[5] AUTOSAR Consortium. Specification of Manifest, Adaptive Platform, Release R20-11. Munich: AUTOSAR Development Partnership, 2020.

 © Author(s) | CC BY 4.0 International License.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

[6] Pretschner, A., Broy, M., Kruger, I. H., and Stauner, T. Software Engineering for Automotive Systems, A Roadmap. Proceedings of the Future of Software Engineering, International Conference on Software Engineering, Minneapolis, MN, pp. 55 to 71, 2007.
[7] Broy, M. Challenges in Automotive Software Engineering. Proceedings of the 28th International Conference on Software Engineering, Shanghai, pp. 33 to 42, 2006.
[8] Navet, N. and Simonot-Lion, F. Automotive Embedded Systems Handbook. Boca Raton, FL: CRC Press, 2009.
[9] Staron, M. Automotive Software Architectures, An Introduction. Cham: Springer International Publishing, 2017.
[10] Weber, M. and Weisbrod, J. Requirements Engineering in Automotive Development, Experiences and Challenges. Proceedings of the IEEE Joint International Conference on Requirements Engineering, Essen, pp. 331 to 340, 2002.
[11] Hartwich, F., Beck, F., and Nowak, W. CAN FD, The Enhanced Communication System for Series Production. Proceedings of the International CAN Conference, Nuremberg, pp. 1 to 9, 2012.
[12] Matheus, K. and Konigseder, T. Automotive Ethernet, Second Edition. Cambridge: Cambridge University Press, 2017.
[13] Burkacky, O., Deichmann, J., Doll, G., and Knochenhauer, C. Rethinking Car Software and Electronics Architecture. McKinsey and Company Technical Report. New York: McKinsey and Company, 2018.
[14] Deichmann, J., Ebel, E., Heineke, K., Heuss, R., Kellner, M., and Steiner, F. Software-Defined Vehicles, Capturing Value in the Age of Vehicle Intelligence. McKinsey and Company Technical Report. New York: McKinsey and Company, 2021.
[15] Bock, F., Siegl, S., Neidig, D., and German, R. Decoupling the Development of Automotive Software and Hardware, A Model-Based Approach. Proceedings of the IEEE International Conference on Software Architecture, Hamburg, pp. 101 to 110, 2019.
[16] Boos, D. and Kramer, J. Over-the-Air Programming of Vehicle Systems, State of the Art and Research Challenges. Proceedings of the IEEE International Conference on Connected Vehicles and Expo, Shenzhen, pp. 89 to 95, 2015.

 © Author(s) | CC BY 4.0 International License.