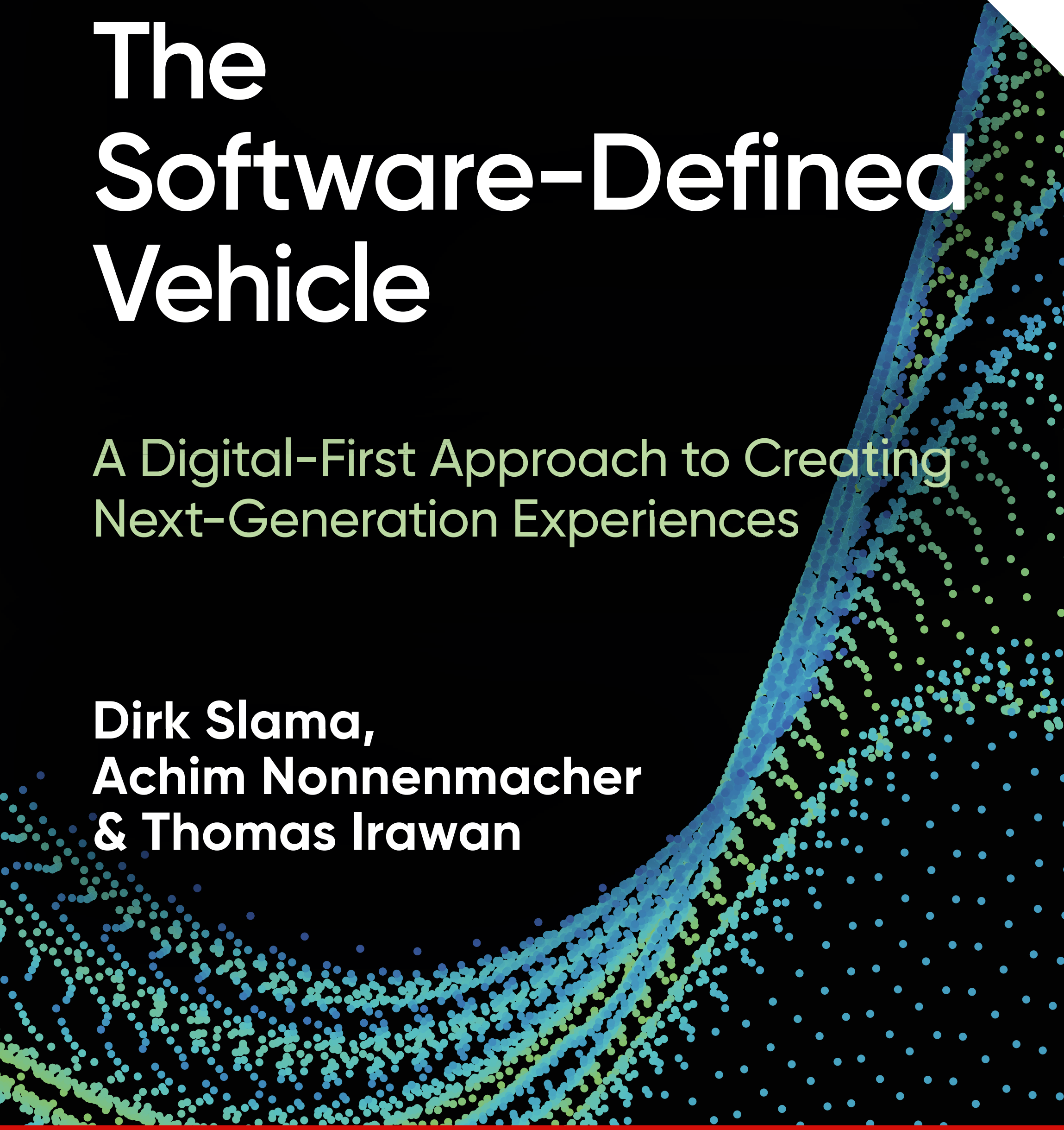
O'REILLY®

Compliments of digital.auto

The Software-Defined Vehicle

A Digital-First Approach to Creating Next-Generation Experiences

Dirk Slama, Achim Nonnenmacher & Thomas Irawan



REPORT

digital.auto

Working together to create the software-defined vehicle experience.

www.digital.auto

Try out our open and web-based prototyping environment to quickly validate your SDV ideas: playground.digital.auto

Sponsor of this report

BOSCH ETAS

Initiators: BOSCH DASSAULT SYSTEMES LeanIX

Standards & Open Source: ECLIPSE FOUNDATION COVESA

Academic Partners: Ferdinand-Steinbeis-Institut

The Software-Defined Vehicle

A Digital-First Approach to Creating Next-Generation Experiences

by Dirk Slama, Achim Nonnenmacher, and Thomas Irawan

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

The Software-Defined Vehicle: A Digital-First Approach to Creating Next-Generation Experiences
by Dirk Slama, Achim Nonnenmacher, and Thomas Irawan

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisitions Editor: Suzanne McQuade
Development Editor: Gary O'Brien
Production Editor: Aleeya Rahman
Copyeditor: Paula L. Fleming
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

September 2023: First Edition

Revision History for the First Edition
2023-09-13: First Release

978-1-098-15778-4

[LSI]

Table of Contents

Preface and Acknowledgments. vii

1. Introduction. 1
Goals of the SDV 2
Impediments: Why Is Automotive Software Development Different? 6

2. What the Automotive Industry Can Learn from Other Industries. . . 15
Learning from the Smartphone Folks: Standardization, Hardware Abstraction, and App Stores 15
Learning from the Internet Folks: Open Source and Cloud-Native Development 18

3. Vehicle OS and Enabling Technologies. 21
Foundation: E/E Architecture 22
Vehicle APIs 22
Vehicle SOA: Encapsulating the E/E Architecture with Vehicle APIs 24
Layers of the Vehicle SOA 25
Ensuring Functional Safety in the Vehicle SOA 26
SDV Tech Stack for the Vehicle SOA 28
OTA: Over-the-Air Updates 30
The Vehicle App Store 31
SDV and AI 32

4. **Value Stream Management for the SDV. 35**
Working at Different Speeds 35
Divide and Conquer 37
Enterprise Perspective 38

5. **#digitalfirst: A New Way of Working. 41**
Shift North 42
Shift Left 43
Virtualization 43
Summary 44

6. **Next Steps. 47**

Preface and Acknowledgments

This publication is the result of the intensive work we are doing for the digital.auto initiative, which brings together OEMs and suppliers, industry consortia members, and open source enthusiasts to help our industry make the software-defined vehicle (SDV) a reality. The digital.auto initiative is open to everyone, and it is vendor and technology agnostic. We seek to advance the adoption of SDVs by focusing on use cases and how to realize them using state-of-the-art technologies and methodologies. Please refer to Chapter 6, "Next Steps" if this endeavor is of interest to you.

This publication would not have been possible without support and feedback from several people. We would like to extend a special thank-you to the contributors of the original SDV 101 course:

- Achim Henkel, Director, Robert Bosch Group
- Alex Oyler, Director, SBD Automotive
- Alexander Djordjevic, Director of Solution Management at RideCare, Robert Bosch GmbH
- Andre Marquis, Senior Fellow/Lecturer, UC Berkeley, Haas School of Business, Entrepreneurship Program
- Ansgar Lindwedel, Director of SDV Ecosystem Development, ETAS GmbH
- Daniel Riexinger, Senior Manager AI & Data-Driven Business Models, Mercedes-Benz AG

- Dominik Rose, VP Product Management & Platform Strategy, LeanIX GmbH
- Frédéric Merceron, Transportation & Mobility Solutions Director, Dassault Systèmes
- Georg Hansbauer, CEO, Testbirds GmbH
- Jacek Marczyk, CEO, Ontonix
- Jann Kirchhoff, Product Success Manager, BMW
- Pei Shen, General Manager of Strategy for Tencent Intelligent Mobility, Tencent Inc.
- Sasha Milinkovic, Manager, mm1 Consulting GmbH
- Sebastian Werner, Head of Automotive Software, Kearney & BinaryCore
- Sven Kappel, VP Software-Defined Vehicles, ETAS GmbH
- Tom Acland, CEO, Dassault Systèmes 3DEXCITE

In addition, we would like to thank Bosch senior management for their support of digital.auto, in particular Tanja Rückert (CDO, Bosch group), Andreas Dempf (CSO/CMO, Bosch Mobility) and Mathias Pillin (CTO, Bosch Mobility), as well as the leadership of the Ferdinand-Steinbeis-Institute: Prof. Heiner Lasi and Michael Köhnlein. Chris Cheng and the team from the Ferdinand-Steinbeis-Institute AIoT Lab have earned our thanks for their relentless support of digital.auto and a variety of use cases (this research was funded by Dieter Schwarz Stiftung); Christian Seiler from Mercedes-Benz for intense discussions about software-defined vehicles and variant management; Damian Dyrbusch from Bosch for strategic vision; Can Yasin Gümüş from Bosch for his input on SDV and ADAS; Hoàng Phan Thanh, Nhân Lương Nguyen, Phong Pham Tuan, and Tam Thai Hoang Minh from Bosch Vietnam for their work on the digital.auto playground; Oliver Kust for his expert guidance on automotive safety; Marco Wagner for discussions around SDV and vehicle abstraction; Imran Abdul Rahiman from ETAS (a Bosch company) for intense discussions about AUTomotive Open System ARchitecture (AUTOSAR); Louis Jackman for the initial version of the digital.auto playground; Mohan B V and Babu Niranjan from Bosch Global Software Technologies for their guidance on the overall structure of this book; Grace Lee from Landing.AI for showing us how AI can be democratized; Boris Scholl from Microsoft for his inputs on Cloud Native; Francesco Salamida and

Stefano Marzani from AWS for discussions around innovations in the SDV space; Wieland Holfelder from Google for his input on the digital.auto playground; and, finally, to our families for having our backs while we were putting in the extra hours to work on this project—thank you!

— Dr. Dirk Slama
Vice President at Bosch
Professor at Ferdinand-Steinbeis-Institute

— Dr. Achim Nonnenmacher
Senior Manager, SDV Innovations at
ETAS (a Bosch company)

— Dr. Thomas Irawan
CEO at ETAS (a Bosch company)

Berlin/Stuttgart, August 2023

Here are some key terms that are found throughout this booklet:

- API: Application Programming Interface
- AUTOSAR: AUTomotive Open System ARchitecture
- CAN: Controller Area Network
- COVESA: Connected Vehicle Systems Alliance
- CI/CD: Continuous integration and delivery
- E/E: Electrical/electronic vehicle architecture
- ECU: Electronic Control Unit
- OEM: Original Equipment Manufacturer
- SOA: Service-oriented architecture
- SOP: Start of Production
- QoS: Quality of Service
- vECU: Virtual Electronic Control Unit
- VSS: Vehicle Signal Specification (from COVESA)

CHAPTER 1

Introduction

In the past, the automotive industry stood as a testament to the power of combustion engines and the prestige of owning a car with the "most exhaust pipes." Today, this old-school paradigm is undergoing a seismic transformation. Four major innovations—electrification, automation, shared mobility, and connected mobility—are happening all at once, leading to dramatic changes in the automobile landscape. Further amplifying this complexity are industry newcomers who are not just building top-quality cars but are specifically focusing on the digital-savvy consumer with their "smartphones on wheels"—vehicles equipped with large interactive screens, seamless connectivity, and frequent updates to software features.

At the same time, customers are expressing dissatisfaction because their vehicles lack the features and user-friendliness that are commonplace in smartphones. Many are left wondering: why can't my $50,000 car perform the same tasks as my $300 smartphone?

From this frustration emerged the idea of a software-defined vehicle (SDV), a car that's fully programmable. New features can be developed and deployed within a matter of months, not years, and there's extra computational capacity for future updates that can be delivered wirelessly. This is a car that keeps getting better, thanks to the continuous delivery of valuable new software features and updates throughout its lifetime.

However, the automotive industry has hurdles to overcome if it wants to achieve this goal quickly. The complexity of software is spiraling out of control, with the volume of code reaching an estimated 300 million lines with level 5 autonomous driving, all while developers cling to the software development methodologies of yesteryear. Further, hardware and software are still tightly integrated and released simultaneously, meaning significant changes occur only once every seven years or so.

Consider the mobile phone industry, where former leading brands like Nokia and BlackBerry were swiftly outpaced by smartphones. A similar fate for the automotive industry's incumbents seems possible. At first, many veterans of the cell phone industry insisted that their business was fundamentally different from the computing industry, but the smartphone revolution quickly shattered that illusion.

This shift raises several critical questions: How can the incumbents in the automotive industry succeed in their ongoing digital transformation to avoid a similar fate? What makes change so challenging within this industry, and how can we significantly accelerate automotive innovation? What will the future of revenue generation look like? Will revenue still come predominantly from car sales, or will digital services layered on top become the primary revenue driver? And crucially, how do we pinpoint the "killer apps"—those standout applications or digital features that prove so essential or attractive that they drive the success of future vehicle generations?

While the SDV is a key technical enabler, the path to answering these questions starts with adopting a "digital first" strategy. This means starting with the digital customer experience and working backward to the solution designs, engaging in early exploration, and testing to ensure that the vehicles on the road align with the ever-evolving needs of the modern consumer.

Goals of the SDV

Let us start our discussion by looking at why we want to build an SDV in the first place. It isn't about the bits and bytes or the number of sensors and cameras—it's about the customer experiences we build. And these customer experiences cannot aim simply to rebuild the smartphone experience. This is about creating a "habitat on wheels" powered by cross-domain applications and data fusion and delivered and constantly improved at 10 times the speed we see today.

~~Smartphone~~ Habitat on Wheels

During the past decade, many car manufacturers have sought to replicate successful smartphone applications in their cars. In many cases, however, these in-vehicle applications could not match the quality of the smartphone apps. In addition, consumers usually don't want redundant experiences, inconsistencies between their digital ecosystems, or the irritation of cumbersome data synchronization. Today, car makers have to walk a fine line between application and data ownership on the one hand and customer experience and tighter integration with the dominant smartphone ecosystems on the other.

This is why it is important that the SDV surpass the concept of a "smartphone on wheels." Instead, it has to enable a "habitat on wheels," utilizing the specifics of the car to provide multisensory experiences that a smartphone could never match. With multiple displays and a network of hundreds of sensors and actuators, the SDV brings together domains like infotainment, autonomous driving, intelligent body, cabin and comfort, energy, and connected car services, crafting a unique journey.

Passengers feel recognized as they enter a vehicle personalized to their needs, one that is a clear departure from the impersonal confines of traditional cars. Passengers experience a reassuring sense of security while journeying under the vigilant care of cutting-edge safety systems. Passengers step into the vehicle with a green conscience. Passengers reclaim their time, transforming travel into an opportunity for meaningful engagement, whether through work, rest, or play. Passengers with disabilities can enjoy a newfound freedom and mobility thanks to autonomous driving. This technology allows them to move with confidence, their independence undiminished. It's not just about the journey; it's about the freedom to explore, to engage, and to live life unrestricted.

SDVs have revolutionized our perception of mobility. It's no longer merely about getting from point A to point B but about making the journey itself enriching. Thanks to advanced driver-assistance systems and autonomous driving, we're embracing the transformative, multisensory power of the SDV.

In the SDV, the digital and physical worlds merge, creating experiences that are both immersive and intuitive. This fusion allows the car to interpret and react to physical cues with precision and sophistication, enhancing the overall experience. For instance, the vehicle's sensors could identify the state of health of children in the back seat and estimate potential motion sickness based on road curvature and speed. Responding to these subtle cues, the system could adapt the automated driving style, adjust window positions, and regulate the air conditioning to enhance comfort and minimize motion sickness.

Moreover, value-added digital features transform the vehicle into a hub of productivity and entertainment. Imagine video conferences displayed on the front car window as holograms, engaging in interactive games with other drivers during traffic jams, or learning a new language based on your surroundings during your commute—all within the safety and comfort of your personal, mobile space. This seamless blend of the digital and physical worlds, facilitated by the SDV, ensures that every journey is not just a ride but an enriching experience that adds value far beyond arrival at your destination.

Cross-Domain Applications and Data Fusion

Today, vehicle experiences often occur in isolated domains. But the future with SDVs promises to blur the boundary between the vehicle and the outside world. Experiences will be cross-domain, where various vehicle functions and systems intercommunicate and interact harmoniously to enrich the overall journey. As Figure 1-1 shows, SDVs represent a shift from disparate functions to integrated experiences, where the vehicle works as a unified entity rather than a collection of separate parts.

Consider the example of a digital "dog mode" that some cars already feature. The vehicle monitors your dog in the car while you are out shopping. Because it is hard for dogs to cool down, a hot car interior on a summer's day is often enough to cause serious injuries or even death. This is a perfect illustration of customer-centric and cross-domain functionality. It involves multiple systems: the car's air conditioning to maintain a comfortable temperature, the infotainment screen to display a message letting passers-by know not to worry as the dog is safe and comfy, and the battery management system to ensure the car has sufficient energy. All these domains are coordinated in order to ensure the dog's safety and comfort.

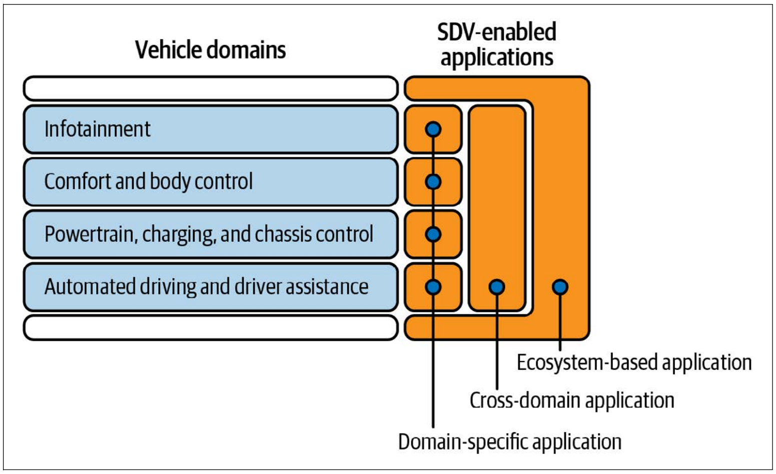

Vehicle domains
SDV-enabled applications
Infotainment
Comfort and body control
Powertrain, charging, and chassis control
Automated driving and driver assistance
Ecosystem-based application
Cross-domain application
Domain-specific application

Figure 1-1. Cross-domain application services

In this connected ecosystem, your car could even become a creative extension of your social media presence. With your permission, it could capture a stunning sunset through its high-quality on-board cameras during a scenic drive and propose a pre-edited post for your approval. Your vehicle becomes an active participant in your digital life, enabling you to share unique moments without detracting from the driving experience.

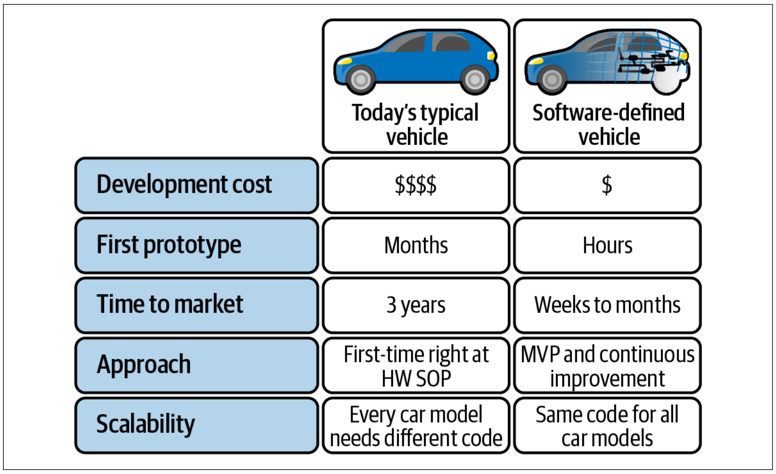
Envision your electric vehicle communicating with a smart grid, scheduling its charging during off-peak electricity usage. This interaction optimizes energy use and maximizes cost efficiency. Your car is no longer just a consumer of energy but also an intelligent participant in the wider energy ecosystem.

Cross-domain experiences also extend to personal wellness. Imagine that your fitness wearable signals that you've had an intense workout. In response, your car sets the cabin temperature to a cooler setting, selects soothing illumination for the ambient lighting, and plays your favorite cool-down playlist. By seamlessly integrating with your digital devices, your car enhances your post-workout recovery and comfort.

This fusion of digital and physical experiences makes the car an extension of your digital life, connecting you seamlessly to the world around you.

Accelerate Innovation by 10x

To achieve the customer-centric development and continuous improvement required to support the usability goals outlined above, development of SDVs must significantly speed up, shortening the time to market for new features. Figure 1-2 shows the transformation that must occur.



	Today's typical vehicle	Software-defined vehicle
Development cost	$$$$	$
First prototype	Months	Hours
Time to market	3 years	Weeks to months
Approach	First-time right at HW SOP	MVP and continuous improvement
Scalability	Every car model needs different code	Same code for all car models

Figure 1-2. "Need for speed" in automotive software development

The goal here should be a tenfold improvement over the old ways of delivering new features. Development costs must be significantly reduced. Initial prototypes must be available in hours, not months. Time to market must be reduced from years to a few weeks.

And all of this does not only apply to new features: we must be able to constantly monitor how customers are using existing features, learn how to improve them, and then optimize them—again, all done 10 times faster than we do it today.

Impediments: Why Is Automotive Software Development Different?

To achieve our goals for the SDV, we have to understand the impediments we need to overcome. Why is automotive software development different from development in other domains (e.g., smartphones)? Sure, "smartphone on wheels" is an accessible metaphor that illustrates the evolving nature of SDVs. With the rise of touchscreens, robust connectivity, and a host of app-driven functions in our cars, drawing a parallel with our handheld smart devices seems apt. However, the simplistic charm of this metaphor can be misleading, as it does not account for the intricacies and unique challenges of the automotive domain.

The following discussion provides a deep dive into the key impediments to the broad adoption of SDV in the automotive industry, including complexity and heterogeneity, functional safety, the "clash of two worlds," and the need for organizational transformation.

Complexity and Heterogeneity

A key difference between the automotive industry and the smartphone industry is their levels of complexity. Over the past decade, the smartphone industry has managed to address technical complexity through standardization and abstraction. Today's smartphones are highly integrated with multiple layers of abstractions and interfaces. The automotive industry, on the other hand, is still plagued by high levels of complexity and heterogeneity.

Furthermore, high levels of abstractions have enabled the smartphone industry to encapsulate most of the hardware complexity, allowing it to deal with complexity on the software level. On the other hand, today's vehicles are still complex systems of systems. Each subsystem in a car, from the brakes to the powertrain, is a complex entity, supplied by a different vendor and integrated with a unique software architecture. The level of complexity and the need for seamless interoperability among systems far surpasses what we see in today's smartphones.

Another key difference is that the automotive industry is dealing with many more product variants. Many cars are configured according to individual customer preferences expressed during the sales process. Regional differences in customer preferences and regulatory requirements lead to large numbers of product variants. And different functional requirements—think family van versus convertible versus truck—lead to many different vehicle types, models, editions, and variants. Thus, there are usually much lower production numbers per vehicle model than per smartphone model.

Functional Safety

The “smartphone on wheels” comparison also falls short when we consider automotive functional safety requirements. Unlike smartphones, vehicles are subject to stringent safety standards, such as ISO 26262. This standard deals with the functional safety of electrical and electronic systems within vehicles and is fundamental to the concept of an SDV.

The ISO 26262 standard doesn’t just offer a set of rules but embodies a risk-based philosophy that assesses potential hazards and requires appropriate design techniques to mitigate them. Risks are categorized according to three factors: severity (the potential harm that can occur), exposure (the probable frequency of the risk situation), and controllability (the ability of the driver to prevent the hazard). By evaluating these factors, the standard effectively balances innovation and safety in automotive design.

To quantify the risk, the standard employs a framework known as Automotive Safety Integrity Levels (ASIL), illustrated in Figure 1-3, which classifies hazardous events that could result from a malfunction based on their level of severity, exposure, and controllability. Levels of risk range from ASIL A, the lowest level, to ASIL D, the highest level. ISO 26262 defines the requirements and safety measures to be applied at each ASIL.

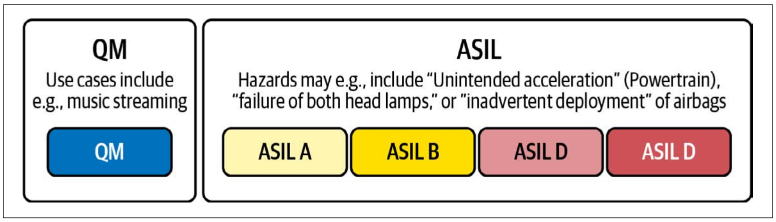

QM
Use cases include e.g., music streaming
QM
ASIL
Hazards may e.g., include “Unintended acceleration” (Powertrain), “failure of both head lamps,” or "inadvertent deployment” of airbags
ASIL A
ASIL B
ASIL D
ASIL D

Figure 1-3. Automotive safety levels

Additionally, a class called QM (Quality Managed) is assigned to systems that, after thorough analysis, require only standard quality management processes due to their lower potential impact on safety. For instance, a safety-relevant function like a braking system would be assigned a high ASIL rating, and an in-car entertainment system would be assigned a QM rating. The software controlling the brakes would need adhere to more rigorous safety standards than the entertainment system. The overarching objective in vehicle design is risk mitigation, a process that seeks to reduce potential hazards.

For example, the risk of a false speed signal due to a defective sensor can be mitigated by introducing a redundant sensor. This additional sensor can cross-validate signals, thus minimizing the chances of error and enhancing the overall safety of the vehicle.

Therefore, while the "smartphone on wheels" analogy succinctly portrays the emerging role of software in vehicles, it does not wholly capture the stringent safety standards and rigorous risk mitigation strategies employed in the automotive industry. The SDV is more than a simple mobile device; it's a sophisticated ensemble of systems that prioritizes safety as much as functionality and convenience.

Clash of Two Worlds

Automotive engineering has traditionally focused on physical functionality, from early electronic features (such as airbags, vehicle stabilization, and braking systems) to modern driver assistance systems or even automated driving. The new, digital experience is driven by software-enabled features, and the focus is on improving the experience of using the vehicle or even providing value-added digital features like usage-based insurance or automated payment of parking fees. Figure 1-4 shows some of these differences.

The traditional world of engineering physical vehicle features is characterized by a complex system of systems—conforming to functional safety, security, and real-time requirements—and homologation, a process of obtaining government-approved certification for product market readiness to ensure that safety and environmental standards are met. In contrast, the new digital vehicle experience is built on best practices from the technology sector, including Agile methods and cloud-style development.

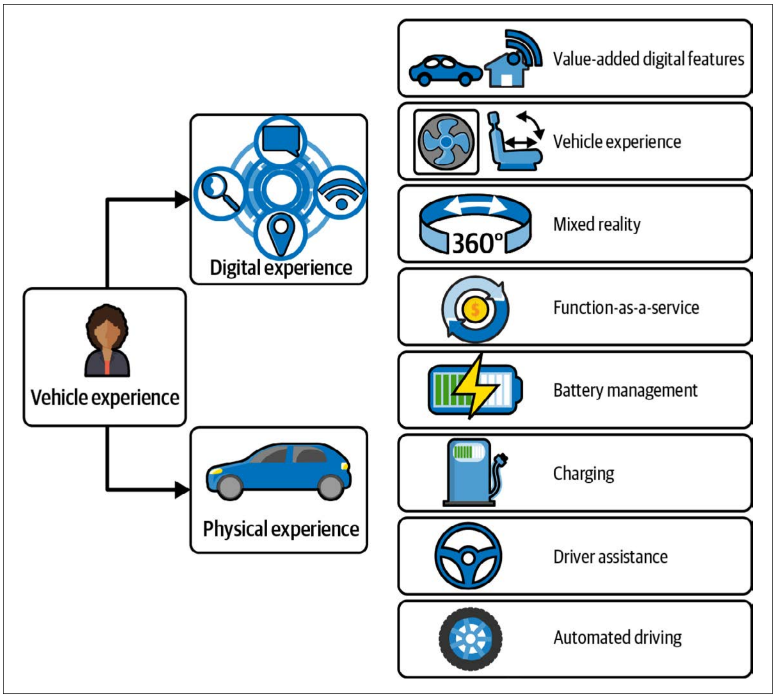


Figure 1-4. Digital versus physical experience

Modern vehicles are a fusion of these worlds; they must merge the reliability of traditional engineering with the agility of modern software development (see Figure 1-5). This presents the automotive industry with a significant challenge: OEMs that are best able to address this “clash of two worlds” are most likely to succeed.

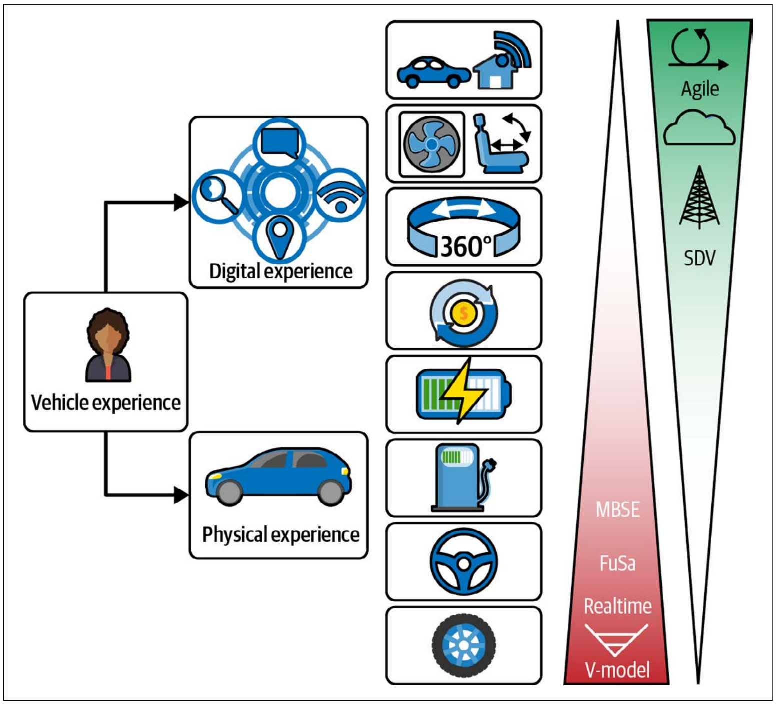


Figure 1-5. Clash of two worlds

Organizational Transformation

We can observe that over the last decade, the incumbent OEMs have undergone complex and large-scale organizational transformation to deliver both physical and digital car features. It is vital that OEMs enable their organizations to combine the worlds of traditional engineering and modern software development and have them work together in relative harmony. But this means that they must create organizations with suborganizations that can move at different speeds and work with different cultures, methods, and tools. It is not trivial to create vehicle platforms and architectures that are so modular that components with different requirements (e.g., functional safety) can be assigned to the organizational units that are the best fit.

This doesn't stop with the company's internal organization. Traditionally, many OEMs purchased different subsystems from different vendors, usually combining hardware and software for each subsystem. With the SDVs, however, the decoupling of hardware and software must be considered with respect to sourcing. Established supply chains are undergoing significant transformations. The roles of existing vendors are changing, and new vendors are entering the playing field.

Rethinking the Vehicle Lifecycle: Digital First

What can we learn from the discussion of impediments that can be applied to the vehicle lifecycle? Historically, the lifecycle of a vehicle was defined by the simultaneous production and deployment of tightly coupled hardware and software. Once the vehicle was in the consumer's hands, its features remained essentially unaltered until its end of life. However, an SDV paradigm allows for the decoupling of hardware and software release dates—a prerequisite for a digital first approach, which puts design and virtual validation of the digital vehicle experience at the start of the lifecycle.

Digital first means that new ideas for the vehicle experience are initially explored in virtual environments, getting early user feedback long before any custom hardware must be developed or even a physical test vehicle is provided. Digital first means that the principles of design thinking and Lean startup, which originated in internet culture, can now be applied in the physical world of automotive development.

In the SDV model, new ideas are generated with approaches like design thinking, where features are explored virtually and tested with early customer feedback, facilitating a faster and more cost-effective way to find a product–market fit. *Product–market fit* refers to a state in which a product meets the specific needs and desires of its target market—in this case for new vehicle applications and services. The ability to rapidly explore, innovate, and adapt is a cornerstone of the new automotive era.

A lot of this relates to our smartphone discussion. Just as the launch of the iPhone opened doors to applications beyond our wildest dreams—identifying plant species, optimizing sleep quality, studying for a driver's license—the advent of the SDV propels us into a future of unimagined possibilities. The applications and experiences made possible by these intelligent habitats on wheels remain largely untapped, inviting creative minds to explore them.

We can't precisely predict what these revolutionary applications might be. It's likely that the SDV's "killer app" won't just be an equivalent of *Angry Birds* for the car but something that leverages the unique potential of this new mobile environment.

But we can confidently make the following statement: to discover these groundbreaking applications and experiences, experimentation and innovation at scale need to happen. This requires much faster development, as we discussed earlier. However, it also requires a new approach to validating new ideas, especially from the point of view of desirability and usability. Building physical test vehicles is very expensive and time-consuming. Because of this, the automotive industry is looking at ways to virtualize usability and acceptance testing. For example, the digital.auto playground provides an environment that allows us to try out new ideas for digital vehicle features in a pure cloud environment and evaluate them against real vehicle APIs. The emergence of spatial computing and virtual reality will accelerate virtualized UX testing. It is important that these tests not be limited to the physical vehicle design. The ability to test the vehicle experience as it is enabled by SDV in virtual environments will help significantly left-shift user testing. This in turn will help make sure that investments are prioritized according to market demand, and the resulting vehicle experience creates popular products and high levels of customer loyalty. Figure 1-6 summarizes the digital first vehicle lifecycle.

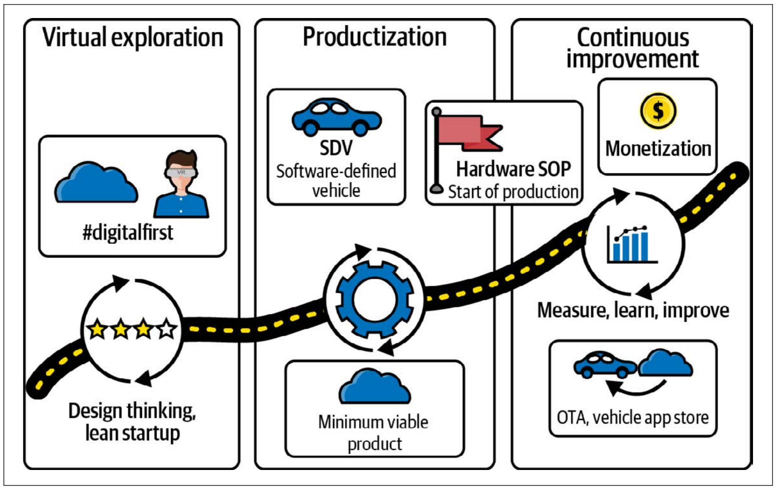


Figure 1-6. Digital first vehicle lifecycle

Unlike in the past when car manufacturers had limited insight into the actual use of certain features, SDVs can, provided the driver gives consent, measure and assess actual usage of vehicle apps and their features. This data-driven approach fosters a fast learning cycle, facilitating innovations that are tailored to customer use and preferences. Chapter 5, “#digitalfirst: A New Way of Working” provides more details on how all of this can be applied in practice.

CHAPTER 2

What the Automotive Industry Can Learn from Other Industries

In Chapter 1, we explored the vision for the modern automotive software realm, diving deep into the concept of a "smartphone habitat on wheels" and addressing the unique challenges and opportunities it introduces. We've demonstrated the imperative to innovate at a rapid pace and the complexities of ensuring functional safety in this hybrid landscape.

Now we shift our focus to the foundational concepts and technologies that underpin this vision. We will take a look at what made the smartphone and internet revolution successful and how we can apply these lessons to the SDV.

Learning from the Smartphone Folks: Standardization, Hardware Abstraction, and App Stores

Nokia's journey into the mobile world offers a significant lesson for the SDV. By 2009, Nokia had created a maze with 57 different and incompatible versions of its operating system. The consequences were devastating. For developers, it became a daunting task to create apps because of the vast fragmentation. For users, the resultant limited application ecosystem made Nokia's platform less attractive.

This tale mirrors a current challenge in the automotive world. Today, almost every car model, even those from a single manufacturer, employs custom hardware and software components sourced from various suppliers. The result: extreme fragmentation combined with monolithic programming frameworks, where creating a "vehicle app" that can run across multiple models of the same manufacturer seems nearly impossible.

However, the smartphone industry offers a blueprint for overcoming this fragmentation. Its solution was multipronged:

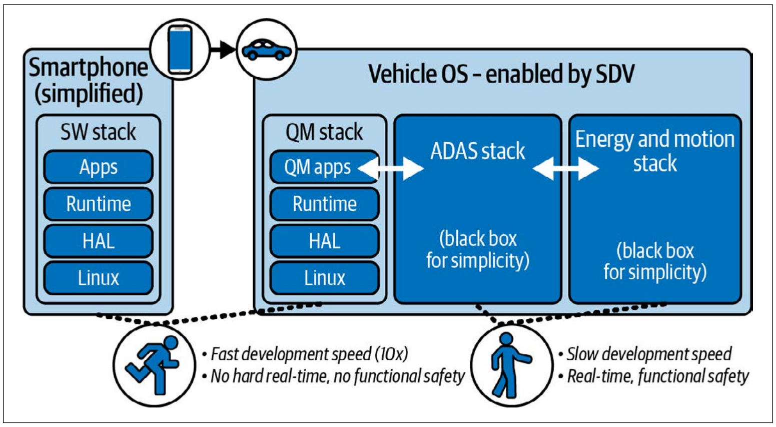
- *Standardized Application Programming Interfaces (APIs)* provide a consistent interface for developers, regardless of underlying system differences.
- An easy-to-use *programming language* and *app development environment* ensures that developers experience a flat learning curve and encapsulated environment for their applications.
- An *app store* acts as a centralized distribution hub, allowing third-party developers to get their applications seamlessly into the hands of end users.

So, what should the SDV industry learn from this? There are at least three lessons here:

1. *Unified APIs.* A set of standardized vehicle APIs would greatly simplify the process of creating software for vehicles. By ensuring that these APIs have minimal fragmentation, developers could write software once and have it work across multiple vehicle models. Based on industry trends, we believe that, in the long run, there will be a natural gravitation toward a select few dominant API sets, minimizing fragmentation across the industry. Within each OEM, we anticipate a more unified API approach, leading to consistency across their individual product lines.
2. *Hardware abstraction layer (HAL).* This acts as a bridge between the software applications and the multitude of vehicle hardware variations. It ensures that software can run irrespective of the underlying hardware differences, adding a layer of consistency and predictability.

3. *Supportive software stack (vehicle OS).* A robust software stack that's in harmony with the standardized APIs and HAL ensures that software can interact seamlessly with a vehicle's components, making software-driven innovations easier to introduce and adopt.

Figure 2-1 provides an overview of how best practices in the smartphone industry can be transferred to the automotive industry. The overview takes into consideration the fact that the automotive domain includes areas with requirements for real-time processing and functional safety.



Smartphone (simplified)
SW stack
Apps
Runtime
HAL
Linux
Vehicle OS - enabled by SDV
QM stack
QM apps
Runtime
HAL
Linux
ADAS stack
(black box for simplicity)
Energy and motion stack
(black box for simplicity)
• Fast development speed (10x)
• No hard real-time, no functional safety
• Slow development speed
• Real-time, functional safety

Figure 2-1. Applying smartphone best practices to automotive development

One key obstacle is the need for OEMs to differentiate themselves. Large OEMs in particular will want to keep large parts of their development proprietary while smaller OEMs might have a stronger interest in standardization and industry-wide reuse. The solution here can be to standardize frameworks and allow differentiation on the content side. For example, an OEM might want to standardize and open up the 70% to 80% of APIs that cover commodity functions. Using the same technical framework, this OEM could still manage the 20% to 30% of APIs that it wants to differentiate in a proprietary way.

Mohan B.V., Technology Head of Strategy, Mobility Next at Bosch BGSW, says:

> We are convinced that in the future, a lot of obligatory features will be cross-OEM. Does it make sense to build OEM-specific APIs? Seamless interoperability for data between vehicle, service provider, and driver is key, and harmonized API will enable this—this is what we have learned from the smartphone industry.

Can Yasin Gümüş, Senior Project Manager of Innovation Management at Bosch XC, adds:

> Defining standardized API opens the market for a community strategy. As an example, solving automated driving and driver assistance are huge challenges. One reason is the regional, complex problems the systems face. Opening up access to the solution space, developers around the globe can contribute to solving these kinds of problems. SDV opens a marketplace, allowing democratization of the development of automated driving and driver assistance.

Learning from the Internet Folks: Open Source and Cloud-Native Development

Even before the emergence of smartphones, the internet transformed the world by revolutionizing communication, access to information, commerce, and social interaction. Key enablers of this change include cloud-native technologies, the democratization of knowledge, and open source development. *Cloud native* refers to an approach to software architecture and development that leverages cloud-computing principles and services to enable fast adaptation to market demands, internet-level scalability, and resilience. Today, many of the software elements required for cloud native are supported by a large, global open source community.

As shown in Figure 2-2, key elements of cloud-native include microservices and APIs, containerization, continuous integration and delivery (CI/CD), and development and operations of IT systems (DevOps).

Microservices and APIs

Microservices are software components that encapsulate their data and business logic and make these available through well-defined APIs. Because microservice architectures are loosely coupled, they are ideally suited to support cross-organizational teams working on multiple microservices, evolving at different speeds.

Containerization

Containers provide the cloud-native runtime for microservices. Containers provide virtualization on the application level, which is more lightweight than the virtualization of an entire operating system. Containers are usually deployed on multiple network nodes in the cloud, providing scalability and resilience for the microservices running on them. They also provide additional levels of isolation, security, and systems management.

Continuous delivery and DevOps

Continuous delivery and DevOps are modern software development practices that ensure that the complexities, dynamics, and uncertainties of today's markets can be supported by frequent and reliable incremental code changes by cross-functional DevOps teams that collaborate throughout the product lifecycle and jointly take ownership of the deliverables.

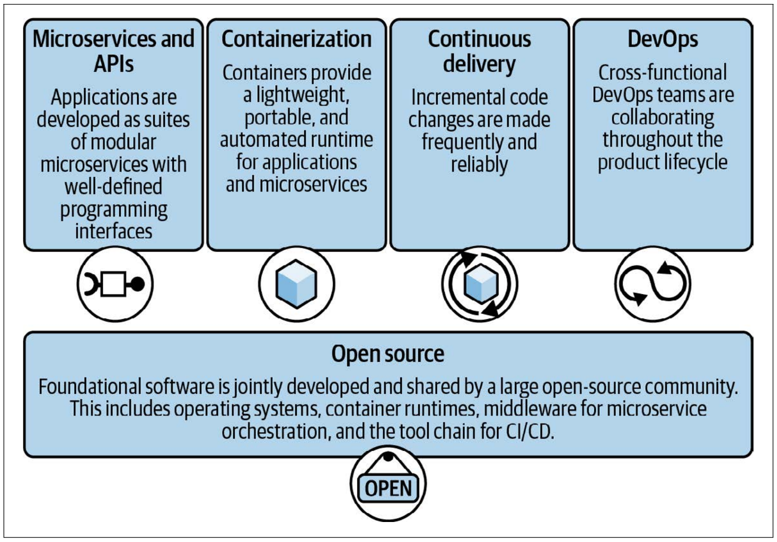

Microservices and APIs
Applications are developed as suites of modular microservices with well-defined programming interfaces
Containerization
Containers provide a lightweight, portable, and automated runtime for applications and microservices
Continuous delivery
Incremental code changes are made frequently and reliably
DevOps
Cross-functional DevOps teams are collaborating throughout the product lifecycle
Open source
Foundational software is jointly developed and shared by a large open-source community. This includes operating systems, container runtimes, middleware for microservice orchestration, and the tool chain for CI/CD.
OPEN

Figure 2-2. Key elements of cloud native and open source development

The success of the internet would not have been possible without *open source*, which has evolved from grassroots community projects to a mainstream movement. Today's thriving open source ecosystem has delivered a secure and scalable infrastructure, which is the backbone of the internet and most modern enterprise application landscapes. The open source community has delivered operating systems (e.g., Linux), container infrastructure (e.g., Kubernetes and Docker), middleware for microservices (e.g., Swagger), and the toolchain for CI/CD. Leading open source organizations in this space include the Linux Foundation, the Eclipse Foundation, and the Cloud Native Computing Foundation. The IT industry learned that by collaborating on non-differentiating parts like a Linux Kernel (shared by everything from smart TVs to cloud servers), more resources can be spent on the differentiating, customer-facing parts of their products.

Today, open source has evolved into a multi-billion-dollar market, including commercial enterprise-grade support, consulting, customizing, training, hosting services, dual licensing, and building commercial products on top of open source foundations.

In the SDV space, the Eclipse SDV working group is emerging as a highly visible open source initiative with a clear focus on SDV. Eclipse SDV aims to provide an open technology platform for the software-defined vehicle of the future and accelerating innovation of automotive software stacks through a vibrant open source community.

CHAPTER 3

Vehicle OS and Enabling Technologies

Almost all OEMs—and many of their suppliers—have been working on creating a modern vehicle operating system (OS) for some time now. There is still no comprehensive and widely accepted definition of what comprises a vehicle OS. However, most definitions contain the following key components: specialized software runtimes for different functional domains, hardware and software decoupling via standardized APIs, and over-the-air (OTA) software updates.

When following this definition, an SDV is a key part of the vehicle OS—or maybe even synonymous with it. SDVs are enabled by a set of technologies that are efficient and continuous, even after the start of production (SOP) delivery of new, digital vehicle features, combining on-board and off-board components into an integrated end-to-end software architecture. We will start by looking at emerging electrical and electronic (E/E) architecture and how it can work with a service-oriented architecture (SOA). Key elements of E/E and SOA are hardware abstraction, vehicle APIs, and the SDV tech stack. Modern vehicles use OTA updates to support post-SOP updates, which will eventually be the foundation for vehicle app stores. Finally, we will look at how artificial intelligence (AI) can augment the software-centric vehicle.

Foundation: E/E Architecture

Today, so-called E/E architecture describes the overall design and layout of electrical and electronic systems in a vehicle. This architecture encompasses the distribution of power, data, and control signals throughout the vehicle as well as the integration and interconnection of various E/E components and systems. Many vehicles still implement a domain-centric E/E architecture where different vehicle domains, such as the powertrain, chassis, passenger compartment, and body, are logically grouped together and connected by dedicated bus systems, such as the Controller Area Network (CAN) bus. The CAN bus is a signal-based protocol designed to allow electronic control units (ECUs) and other compute nodes in a vehicle to communicate with each other in a reliable, priority-driven way.

Since the domain E/E architecture results in very complex and heavy vehicle-wiring harnesses, OEMs are using so-called zonal architectures, which aim to group different vehicle sensors and actuators according to their physical location in the vehicle. In a zonal E/E architecture, wiring harnesses become less redundant, allowing for simplified connections within individual zones, reducing complexity and weight, and enabling easier integration of new features and technologies. Zonal architectures usually combine dedicated zone controllers with high-performance vehicle computers. The zone controllers are locally connected to various sensors and actuators, often using different legacy/heritage bus systems, such as CAN, Local Interconnect Network (LIN), and FlexRay. The zone controllers are then connected with each other and with the high-performance vehicle computers via new on-board, high-speed networks based on Ethernet (the foundation of today's internet).

Vehicle APIs

The first step toward a service-oriented architecture for digital on- and off-board services is to provide a hardware abstraction via vehicle APIs. Today, developing new on-board features usually involves a complex and lengthy alignment process among many departments of the OEM. This is because all signals within a given on-board domain are communicated via a shared bus system (e.g., the CAN bus). For each vehicle type, a CAN matrix defines which ECUs send which message under which conditions and with which cycle time, which ECUs receive which messages, and how the messages are structured and prioritized. This results in a tightly coupled architecture that requires very close alignment on technical and organizational levels. To get from here to a loosely coupled, service-oriented architecture, a new level of hardware abstraction is required.

From the perspective of the service consumer (i.e., user of digital applications), the vehicle should provide a set of well-defined APIs that provide an abstraction of the vehicle's functions. On the lowest level, these are the sensors and actuators of the vehicle. A good example of this open industry standard is the Vehicle Signal Specification (VSS) defined by the Connected Vehicle Systems Alliance (COVESA VSS). COVESA VSS defines a tree-like API structure for accessing vehicle sensors and actuators as signals (see Figure 3-1).

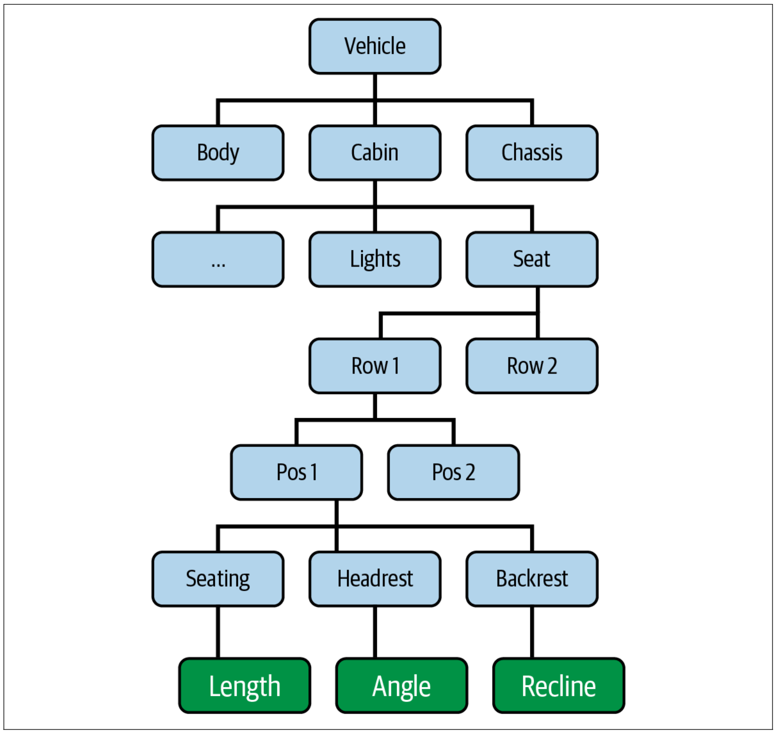


Figure 3-1. Vehicle signal API tree (based on COVESA VS)

For example, `Vehicle.Cabin.Seat.Row1.Pos1.Headrest.Angle` would give an application developer access to the actuator that controls the angle of the headrest of the front left seat. This is a level of abstraction suitable for developers used to cloud-native or smartphone development. Of course, these relatively low-level signal-to-service APIs have to be augmented with higher-level orchestration services over time.

However, there are some issues with this approach. First, as discussed before, it can be challenging to map such a high-level software API to the underlying, complex E/E architecture. Second, there is the question of how to ensure the functional safety of APIs that may impact vehicle physics. And third, it is necessary to solve for Quality of Service (QoS) aspects of such an API (e.g., real-time requirements). We will look at all of these aspects in the following sections.

Vehicle SOA: Encapsulating the E/E Architecture with Vehicle APIs

The issue with the E/E architecture perspective is that it is modeled after the physical design of the vehicle, including actuators, sensors, on-board networks, and compute nodes. A key prerequisite for SDVs will be to encapsulate the hardware-focused E/E architectures with a vehicle service-oriented architecture (SOA), as depicted in Figure 3-2.

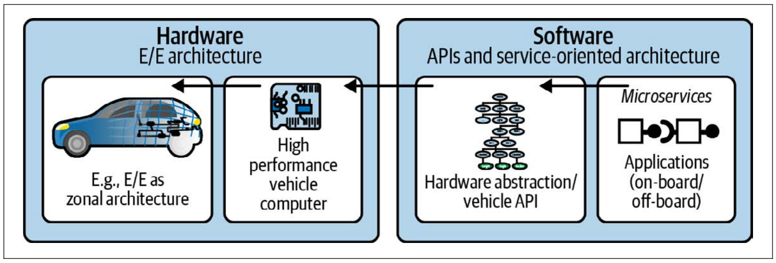

Hardware
E/E architecture
E.g., E/E as zonal architecture
High performance vehicle computer
Software
APIs and service-oriented architecture
Hardware abstraction/ vehicle API
Microservices
Applications (on-board/ off-board)

Figure 3-2. SOA encapsulating the E/E architecture

On the E/E hardware side, high-performance computers run the SDV functions. These computers are physically connected with the other on-board components, including smaller compute nodes, sensors, and actuators. The translation from the hardware side to the software side can happen on these high-performance computers; hardware functions are exposed via APIs. These APIs are the foundation for the communication between application-level microservices and the underlying hardware. A *microservice* is an encapsulated piece of software that implements a specific function. Interaction between microservices always happens via APIs. These can either be APIs abstracting a specific hardware function or application-level APIs. An *application* is a collection of microservices that are orchestrated to provide a specific feature or digital service.

Layers of the Vehicle SOA

SOAs introduce a level of architectural layering that helps deal with the individual characteristics of the different microservices involved. In the world of the internet, for example, the frontend layer of an application would include microservices that change frequently due to continuous optimization of the user interface while the basic services layer would include more stable, data-centric services that change much less frequently. This type of layering is essential for the efficient evolution of complex systems.

In the automotive world and SDV, this layering is different and has multiple dimensions. The first dimension we need to look at is on-board versus off-board. The second dimension is defined by the functional safety and real-time requirements of the contained microservices and the vehicle event chains encapsulated by those microservices.

Putting these dimensions together with what we discussed earlier about E/E architectures and vehicle APIs, we get a service-oriented architecture for the SDV as described in Figure 3-3. At the top, the green layer indicates the environment for QM-only microservices hosted in the cloud. A set of vehicle-to-cloud APIs connects on-board services with off-board services. In the middle part, the SDV layer includes both QM-only and ASIL A/B microservices, which are hosted in different environments. Further south in this architecture, the physical vehicle functions can be found, made accessible through a signal-to-service API (e.g., based on COVESA VSS). The signal-to-service API must encapsulate the mapping of the APIs to the E/E architecture of the vehicle. For example, the API must know which zone controller would actually host the functions required to support `Vehicle.Cabin.Seat.Row1.Pos1.Headrest.Angle` (see

Figure 3-1). The zone controller (in this case the "Zone FL" of the front left) would, in turn, need to provide a software proxy that can translate the API onto the matching CAN signal so that the angle of the front left headrest is changed.

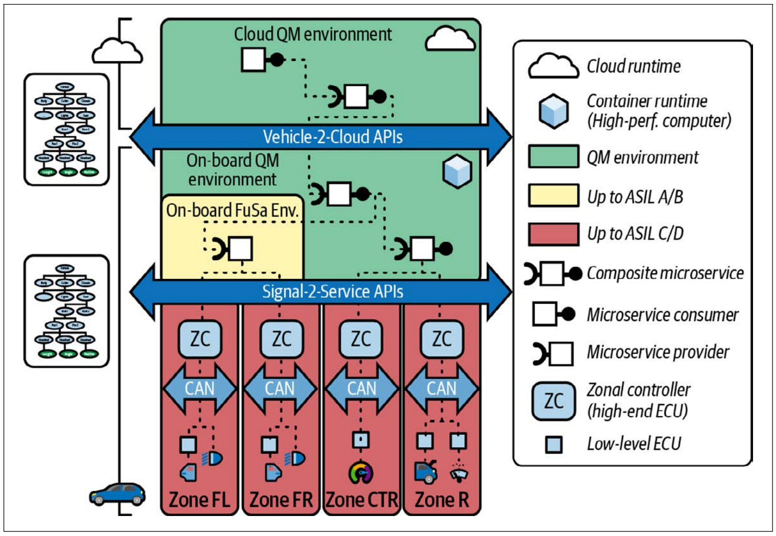


Figure 3-3. Service-oriented architecture for the software-defined vehicle

Ensuring Functional Safety in the Vehicle SOA

A key benefit of the multilayered vehicle service architecture is that we now have several options for functional safety. Let's look at a concrete example in Figure 3-4, a function in a smartphone app that remotely opens the trunk of a vehicle.

The problem with the "open trunk" function is that it can only be safely executed if the vehicle is in a safe state (i.e., not moving). So, the question is: who will perform this check, and how? The caller itself cannot perform the check—this would violate the loose coupling principles of the service-oriented architecture, as the service implementation cannot assume clients always properly follow a certain protocol. This is certainly true for clients calling from the cloud, but also for clients calling from a QM environment on board the vehicle. Indeed, this is the whole point of the QM environment; it provides none of the ASIL QoS properties. This means that the on-board "open trunk" service has to perform the required safety checks itself. Assuming that the service interface is exposed to QM clients, there are now at least two options.

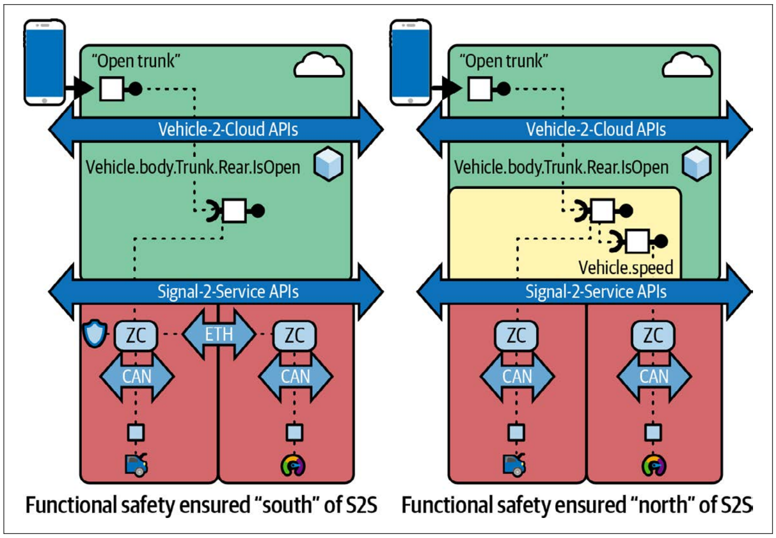


Figure 3-4. Vehicle SOA and Functional Safety–two examples

In the first option (Figure 3-4, left), the on-board "open trunk" service (`Vehicle.Body.Trunk.Rear.IsOpen`) runs in a QM environment and is therefore not considered safe. It might provide some additional services, but eventually the safety check must be done from within the ASIL safety environment. In our case, this would be the zone controller for the trunk. Here, the system must check the current vehicle speed before opening the trunk. However, in this example, a different zone controller manages the speed signal (i.e., the zone controller for the powertrain). So the zone controller that manages the trunk must access the zone controller that manages the speed signal before communicating with the low-level ECU controlling the trunk. And all of this must happen in real time (which is why this check is performed on the zone controller, not in the QM layer above). This does not necessarily mean it has to happen in a fraction of a millisecond, but it has to happen within a guaranteed time interval. For this to happen, the two zone controllers must be in more direct communication, such as a direct link supporting so-called Time-Sensitive Networking (TSN) via high-speed Ethernet.

The second option (Figure 3-4, right) assumes that the on-board "open trunk" service is executed in a safety environment matching the ASIL levels required for this function. This means the exposed interface can still be called from a QM client (e.g., an event sequence originating in the cloud), but the implementation behind the interface is safe. If there is another safety-compliant `Vehicle.Speed` service (and a safe way for the trunk service to call the speed service), then this can now all be done in the same environment. The benefit is that the speed service is provided as a real microservice that can be reused by QM as well as ASIL services. However, this requires a service architecture capable of orchestrating microservices with ASIL QoS levels.

Both options presented here are valid. The first option makes fewer demands on the integration levels within the environment and might be easier to implement with current technologies. The second option has more potential to enable advanced cross-domain use cases, but requires a more advanced ASIL-compliant runtime environment. The following discussion will look at the SDV tech stack required to support both options.

SDV Tech Stack for the Vehicle SOA

To understand how a vehicle SOA can support architectural layers with different QoS levels, we need to add another dimension—adding the hardware, operating system, and middleware/application environment. This is shown in Figure 3-5.

The cloud layer shown here is as to be expected. The hardware includes generic central processing units (CPUs) and graphics processing units (GPUs), which are used today for processing machine learning workloads in the cloud. On the operating systems (OS) level, we usually find a general-purpose OS like Linux. Hypervisors or virtual machines are used for running multiple OS instances on shared hardware, which is important for scalability and enabling cloud elasticity. Modern clouds also provide very rich middleware and application runtimes.

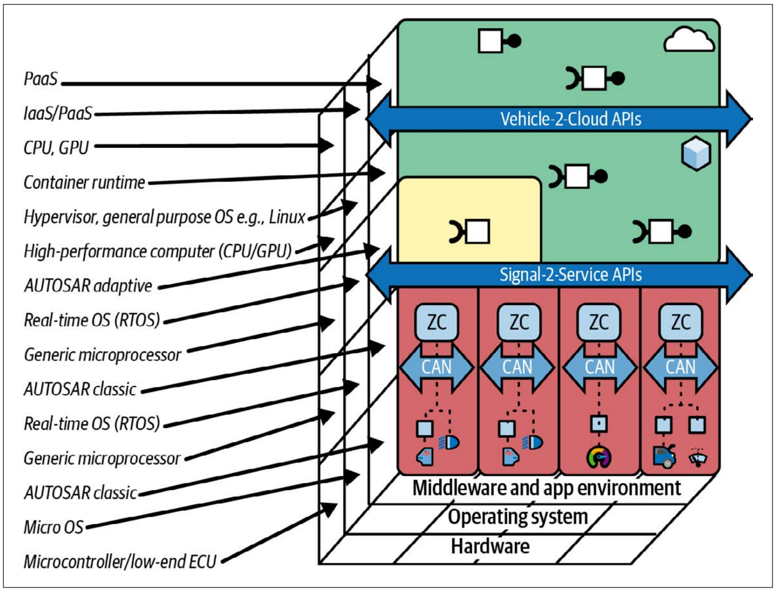


Figure 3-5. The SDV tech stack

Next we have the on-board environment for QM-only microservices and applications. This type of environment aims to replicate as much as possible the rich tech stack found today in the cloud while adding some vehicle-specific aspects. These will include, for example, faster start-up times, energy efficiency, and support for managing highly distributed instances that are not always online (e.g., large vehicle fleets, as opposed to central cloud data centers). The OS in this layer will usually be a general-purpose OS combined with hypervisors for virtualization. This will run on a generic high-performance vehicle computer, potentially with additional GPUs for AI/ML workloads.

The next layer has to support high QoS levels and consequently requires a more traditional high-availability environment, usually with real-time support. A widely used OS in this area is QNX from BlackBerry. The zone controllers will probably share a similar setup. A real-time-capable middleware will be required to link microservices from the environment north of the S2S API with those residing south of it (e.g., on zone controllers).

Finally, the zone controllers connect to lower-level ECUs. Especially for the control of specific vehicle sensors and actuators, microcontrollers are used. These are integrated circuits (ICs), usually representing a complete system on chip (SoC), including the processor core, memory, and IO, all inside one discrete package. Specialized environments for low-level but highly efficient and very targeted embedded functions are used here. Between the zone controller and the different low-level ECUs and microcontrollers, different bus systems have to be supported, including CAN, LIN, and FlexRay.

OTA: Over-the-Air Updates

In the past, the automotive industry aimed to freeze the hardware and supporting software of a new vehicle generation at the SOP. Post-SOP changes to the supporting software were usually made only if serious problems needed to be fixed.

Of the 70 to 100 ECUs and controllers usually found on a traditional vehicle, the majority would never be updated after the initial software flashing as part of the manufacturing process. If updates were needed, a dedicated update campaign would be run for each affected variant ("push"). Updates are usually manually packaged based on extensive variant research and validation.

In the future, many OEMs envision a rich application ecosystem developing for their vehicles. Software updates will no longer be done only for quality reasons but can happen on demand. Customers will select which new applications and features they want to download and activate ("pull"). These new applications can span different domains, ranging from infotainment and well-being to cabin comfort and driving performance. However, this increasing individualization will also lead to a dramatic rise in variants of hardware/software combinations. The next generation of OTA will have to address this (Figure 3-6).

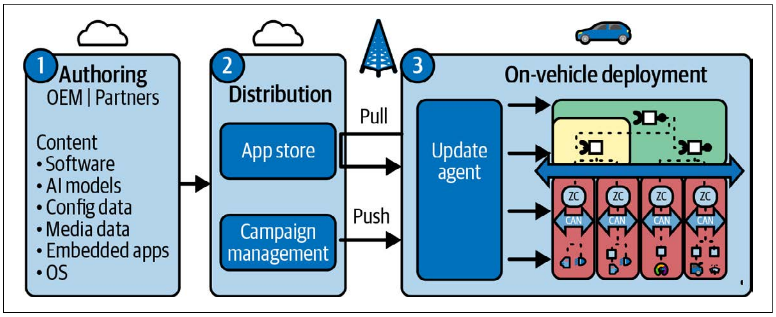


Figure 3-6. Modern over-the-air update architecture

The Vehicle App Store

The holy grail of the SDV is the vehicle app store. The smartphone industry has proven the huge potential of applications and content generated by partners and external developers, made accessible to customers on demand and post-SOP. Now OEMs are jumping on the bandwagon. Replicating the success story of smartphone app stores in the automotive industry requires several things. Figure 3-7 gives an overview of the technical prerequisites.

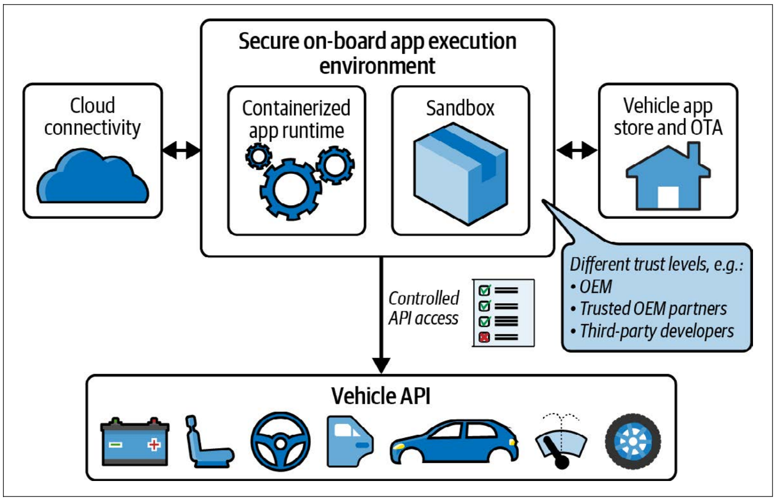

Secure on-board app execution environment
Cloud connectivity
Containerized app runtime
Sandbox
Vehicle app store and OTA
Controlled API access
Different trust levels, e.g.:
• OEM
• Trusted OEM partners
• Third-party developers
Vehicle API

Figure 3-7. Vehicle app store

First, vehicles need a secure on-board application runtime environment for the execution of applications downloaded via the vehicle app store. This runtime is often referred to as a sandbox. It is important that this sandbox be secure, not allowing code to break out and access other parts of the vehicle. A virus that takes control of a vehicle can have deadly consequences for the vehicle's occupants or bystanders.

Second, the system architecture must ensure that applications running inside the sandbox can only interact with other parts of the vehicle via a set of controlled APIs. Most likely, OEMs will have to define different trust levels for their APIs, differentiating between API access through OEM-developed applications, trusted partner apps, and potentially apps developed by completely unknown third parties. Already, some vehicles available today provide app stores for in-vehicle infotainment. In the future, vehicle app stores should also provide access to sensor APIs to unleash the creativity of the global developer community. They may even offer safe access, so selected actuators might be an option (e.g., in the body and comfort domain).

While the smartphone juggernauts were already able to position apps via Apple Car Play and Android Automotive, many OEMs are now starting to offer app stores for applications running natively in the vehicle. This journey starts in the infotainment area, introducing apps designed to run natively on the built-in infotainment system screen. Examples include apps for music and podcasting, video conferencing, weather, gaming, news, parking, and electric vehicle (EV) charging. It remains to be seen how well these vehicle infotainment applications will be able to compete with their smartphone counterparts.

To be better differentiated, the next generation of in-vehicle apps is likely to make use of vehicle-specific features. Once apps are able to interact in a meaningful and safe way with the vehicle, a new experience can be created. Being able to access data from vehicle sensors—and potentially even control some of the vehicle actuators—will create a new type of application. For example, being able to access in-vehicle sensors, such as cameras, radar, and microphones, could create a new generation of health and well-being apps. This would help OEMs come closer to the vision of a "habitat on wheels."

SDV and AI

While AI has already become a hot topic, the release of ChatGTP has moved the discussion from "software will eat the world" to "AI will eat the world." AI is disruptive on many levels, and the automotive world is no exception. So, do we need an "AI-defined vehicle" instead of (or in addition) to a software-defined one? Where in the SDV architecture does AI play a role? The answer is: potentially in all the layers of the SDV architecture (see Figure 3-8).

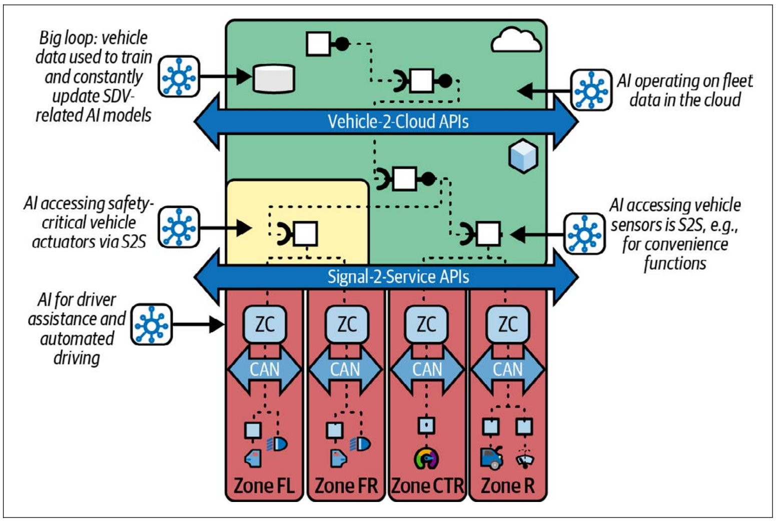

Big loop: vehicle data used to train and constantly update SDV-related AI models
AI operating on fleet data in the cloud
Vehicle-2-Cloud APIs
AI accessing safety-critical vehicle actuators via S2S
AI accessing vehicle sensors is S2S, e.g., for convenience functions
Signal-2-Service APIs
AI for driver assistance and automated driving
ZC
CAN
Zone FL
Zone FR
Zone CTR
Zone R

Figure 3-8. SDV and AI

On the smartphone, AI can be used to control access to the vehicle or specific vehicle functions (e.g., via face recognition). In the cloud, AI can operate on vehicle fleet data (e.g., to support vehicle routing, optimize the EV charging experience, perform traffic analysis, improve battery performance, etc.).

On-board, we are seeing different types of AI. For highly automated driving, AI is deeply integrated with the vehicle's E/E architecture, providing object identification (e.g., recognizing a child on a bicycle in front of the car) and controlling the vehicle's trajectory (e.g., braking for the aforementioned child). In addition, there seems to be a huge potential for AI-enabled long-tail applications. These applications require much less investment for their development and will probably have a lesser overall impact individually but still have a huge impact overall. The smartphone app stores have proven that if the boundaries of entry are sufficiently low, a plethora of new, creative applications will be created. This should also be true for AI-enabled vehicle apps.

Combining AI with data from in-vehicle sensors alone could be a game changer (e.g., for infotainment and well-being). If this was then combined with SDV-enabled functions, a whole new generation of in-vehicle applications could be created (e.g., cabin comfort applications that use sensor data to change the vehicle's ambience by accessing in-cabin light, HVAC, seat massage functions, and so forth). The use of AI will reach far beyond vehicle apps. From voice assistance to predictive maintenance, from fleet operations to automated driving to AI-assisted development toolchains, AI will be a game changer for the way we design, develop, operate, and interact with vehicles.

CHAPTER 4 Value Stream Management for the SDV

The internet is built on sophisticated, constantly evolving technologies. Consequently, many successful internet businesses are built on a technology culture. However, this reliance on technology can sometimes make it difficult to focus on building customer-centric products, increasing competitiveness and generating revenue. Therefore, many companies have adopted Value Stream Management (VSM) as a best practice for their digital businesses. VSM is a set of practices designed to ensure that new digital features are delivered fast and efficiently and that they deliver clear customer value.

So how can VSM be applied to the software-defined vehicle? Let's have a look.

Working at Different Speeds

What is maybe the most important realization in the context of the SDV is that there cannot be a single value stream. Our discussion of the "clash of two worlds" and the following technical discussions have shown that different requirements need different, specialized approaches. The digital vehicle experience must be delivered by a digital value stream, which adheres to Agile best practices, while the physical vehicle experience must be delivered by a value stream that adheres to the rigorous and "first time right" thinking required for areas with high levels of functional safety requirements (see Figure 4-1).

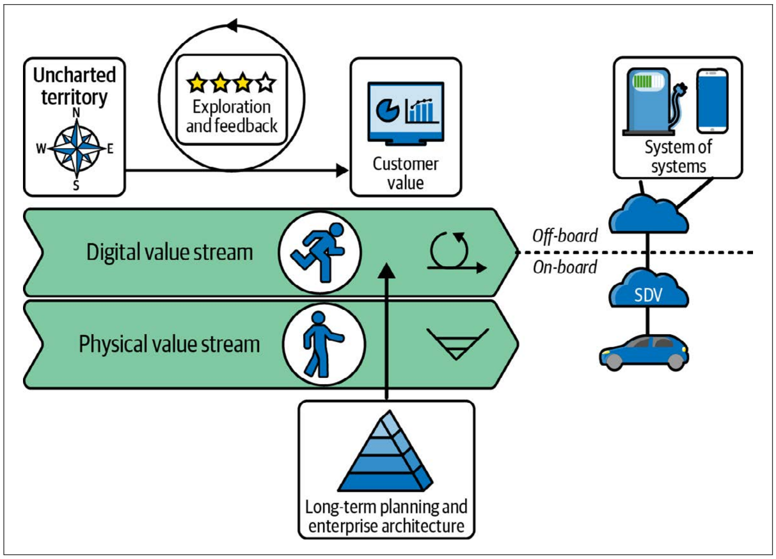


Figure 4-1. High-level value stream perspective for the SDV

The digital value stream must be able to address uncharted territory, deal with vague requirements and constantly changing ideas, and be able to quickly react to customer feedback. Building an explorative and feedback-based approach to the digital value stream is key. The technical delivery pipelines of the digital value stream will be outputting artifacts that are deployed either in the cloud or on-board. The on-board artifacts will use SDV, containerization, OTA, and vehicle APIs. Mechanisms for measuring customer success can be natively built into this technology stack, similar to the customer journey analysis tools familiar from internet applications.

The physical value stream must be aligned much more closely with long-term planning and the enterprise architecture. The methods applied here will adhere to established best practices for functional safety-related features (e.g., the well-established V-model of software development, which has formal verification and validation mechanisms built in). The technical outputs of the physical value stream will include a combination of hardware and software, such as embedded software and ECUs, as well as mechatronic system components. Implementing mechanisms for measuring customer success in the physical value stream will be more difficult.

Establishing an effective VSM strategy that enables OEMs to work at different speeds in different areas will be a key success factor in the future.

Divide and Conquer

The beauty of SOA and the API-centric way of working that we introduced earlier is that these approaches give us the capability to create integration not only on the technical level but also on the organizational level. To support value streams that are moving at different speeds, a loose coupling is required both on the technical as well as on the organizational level. And this is exactly what can be achieved with APIs and hardware abstraction.

Figure 4-2 shows an example of loose coupling. At the top, we have technical artifacts created as output of the digital value stream. First, prototypes are created and implemented against the vehicle API—e.g., using the digital.auto playground for rapid online prototyping of SDV features. These prototypes can be used to get very early feedback from key stakeholders, including customers and management. Next, the early SDV prototypes are refined and brought closer to real applications. In the early development phases, these SDV applications can utilize vehicle simulations behind the APIs to create realistic test environments. Once real vehicle hardware is available, this hardware can replace the vehicle simulation. Since the APIs are not changing (at least in an ideal world) and SDV application is implemented against the API, this shift from a vehicle simulation to real vehicle hardware should be seamless from the point of view of the SDV application. This means that the API becomes the mechanism that ensures loose coupling not only between the technical artifacts but also between the different organizations involved. The teams developing the digital value stream are doing so against the APIs representing the real hardware underneath (hence "hardware abstraction"), and by using simulators, they can move at their own speed, independent of the availability of the real hardware. This approach takes the well-established concepts of hardware-in-the-loop (HIL) and software-in-the-loop (SIL) to another level.

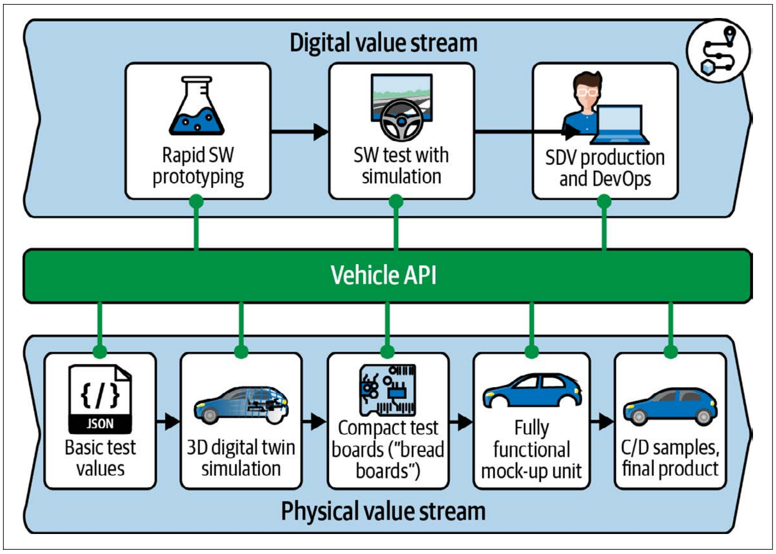

Digital value stream
Rapid SW prototyping
SW test with simulation
SDV production and DevOps
Vehicle API
JSON
Basic test values
3D digital twin simulation
Compact test boards ("bread boards")
Fully functional mock-up unit
C/D samples, final product
Physical value stream

Figure 4-2. Value streams with technical artifacts moving at different speeds

Being able to split up a complex body of work using a divide and conquer strategy is a huge benefit, reducing complexity to a manageable level. Notice that the APIs are likely to take on the role of a "master clock," helping to synchronize work between the different teams in the different value streams.

Enterprise Perspective

OEMs have traditionally addressed the complexity they face with enterprise architecture management (EAM) and Model-Based Systems Engineering (MBSE), shown in Figure 4-3. EAM helps manage the dependencies between the systems-of-systems perspective (e.g., the vehicle in the context of its environment), the system perspective (the vehicle itself), as well as the subsystems, including key components and features. Of course, all of this must be seen in the context of many different vehicle variants and vehicle types. Finally, managing the reuse of vehicle platforms is key, including hardware platforms, E/E platforms, and software platforms. MBSE plays an increasingly important role in the detailed design of many system components and their interdependencies.

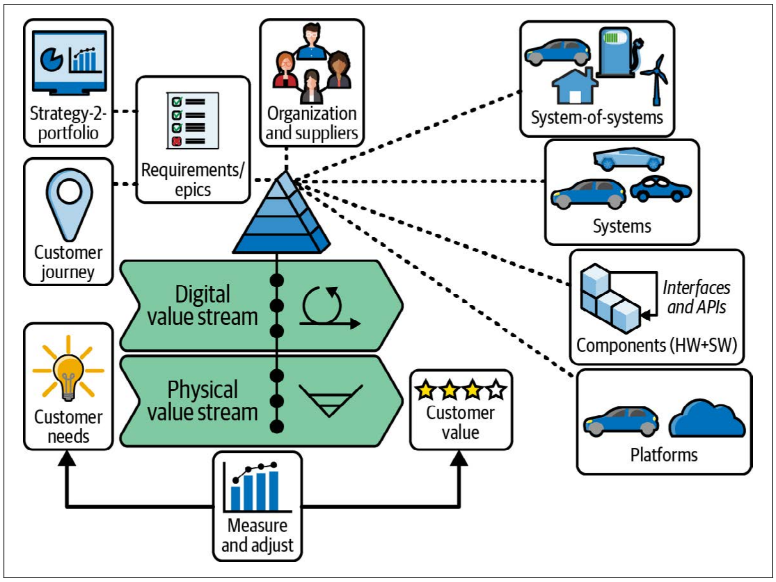


Figure 4-3. The enterprise perspective

However, coming back to the "clash of two worlds" discussion, it is important to notice that these kinds of tools and methods are often seen as very controversial in the software world, which has spent the last 20 years adopting an Agile culture. This becomes clear in Table 4-1, which compares how the Agile values defined in the famous manifesto for Agile software development map to model-centric and code-centric approaches to automotive development.

Table 4-1. Model-centric versus code-centric development

Values from the Agile Manifesto	Model-centric (EAM/MBSE)	Code-centric/SDV
Individuals and interactions over processes and tools	Tools and processes required, especially for hardware and software with ASIL requirements	Well suited to Agile processes for QM features
Working software over comprehensive documentation	Can be achieved via code generated from models (but not a trivial task)	Well supported
Customer collaboration over contract negotiation	Model as contract; early customer validation requires virtual exploration	APIs as contracts, but features developed in close alignment with customers
Responding to change over following a plan	Long-term planning required for hardware and ASIL software	Supported by Agile approach

As we have discussed before, the ability to work with different value streams that embody different approaches and methods is the key to success. On the enterprise-level, methods and mechanisms must be established to keep the different value streams in sync, supported by a loose-coupling approach on the organizational level. Again, APIs can play a key role here, for example, by providing a loose coupling between the Agile/code-centric and the model-centric/MBSE perspective.

CHAPTER 5

#digitalfirst: A New Way of Working

Taking everything into account that we have discussed so far, we propose a new way of working for OEMs, which we are simply calling #digitalfirst.

The fluid nature of today's consumer preferences means we can't definitively predict today which features will be in demand tomorrow. However, a key certainty emerges: OEMs that can't swiftly explore, test, and deliver new features at scale may find themselves incapable of crafting the engaging customer experiences today's consumers demand. In our rapidly evolving digital age, the agility to innovate quickly and effectively is not just desirable—it's essential to staying relevant in the automotive industry.

Consequently, #digitalfirst starts with customer experiences and works backward to the technology. We have co-founded the digital.auto initiative to support this. "Digital first, auto second" is not just a tagline, but the core ethos guiding this transformation. As shown in Figure 5-1, #digitalfirst assumes that an OEM has to undergo three tectonic shifts: the shift north (of the vehicle API), the shift left (toward early-stage testing), and the shift toward virtualized development. We will look at each of these shifts in more detail.

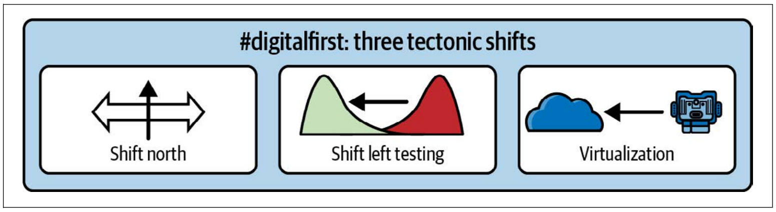


Figure 5-1. The three tectonic shifts underlying #digitalfirst

Shift North

The shift north involves moving functionalities from the safety domain to the QM domain, creating a separation between the domains via well-defined APIs. This is driven by the extra effort that will remain for every development in the safety domain—validation, homologation, and extra documentation—that is usually required to a lesser degree in the QM world.

Shifting code north into the QM world, as shown in Figure 5-2, means that modern software engineering techniques and tools can be used, speeding up development and making updates post-SOP much easier. In addition, in this domain, even software developers who don't have years of experience in the automotive sector (modern software engineers who also know ISO 26262 are rare) can work productively, thus accelerating development significantly.

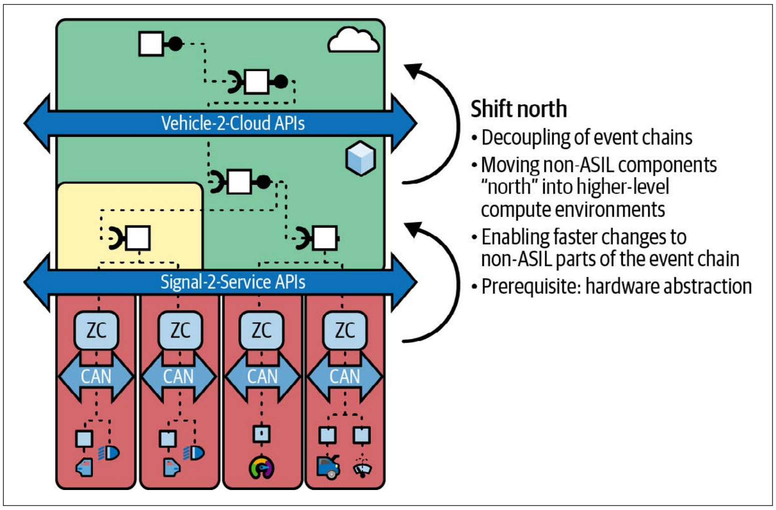
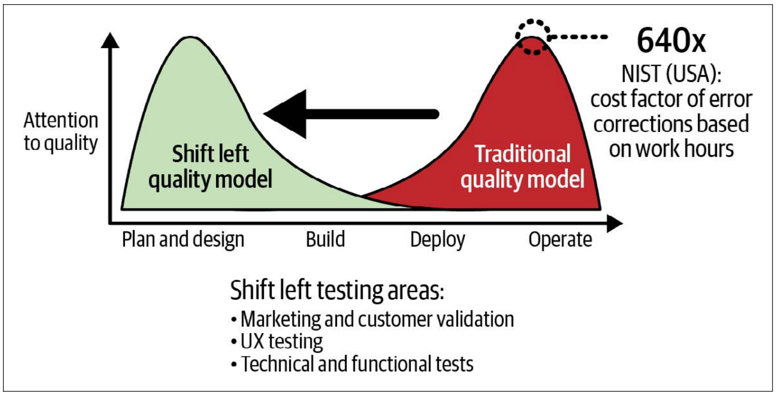

Vehicle-2-Cloud APIs
Signal-2-Service APIs
ZC
ZC
ZC
ZC
CAN
CAN
CAN
CAN
Shift north
• Decoupling of event chains
• Moving non-ASIL components "north" into higher-level compute environments
• Enabling faster changes to non-ASIL parts of the event chain
• Prerequisite: hardware abstraction

Figure 5-2. Shift north

Shift Left

The shift left refers to exploring customer features and conducting user testing as early as possible in the development process, as shown in Figure 5-3. Many of today's cars offer an abundance of buttons and features, many of which remain unknown or unused by users. Hence, the objective is to discern truly desirable features and maximize customer satisfaction through early exploration and testing.



640x
NIST (USA): cost factor of error corrections based on work hours
Attention to quality
Shift left quality model
Traditional quality model
Plan and design
Build
Deploy
Operate
Shift left testing areas:
• Marketing and customer validation
• UX testing
• Technical and functional tests

Figure 5-3. Shift-left testing

According to a report from the National Institute of Standards and Technology (NIST), an agency of the US Department of Commerce, the cost of fixing problems downstream in the development process can be up to 640 times the original development cost. And this does not factor in the cost for losing business due to unhappy customers, or customers who simply don't care about certain features that the developers were sure the customers would love but never cared to ask.

Virtualization

Virtualization emphasizes the development and testing of systems in virtual cloud environments. An interesting organization in this space is SOAFEE, which is developing virtualization concepts in the context for ARM-centric hardware architectures. The main motivation here is to overcome the traditional tight coupling between hardware and software in automotive development, where software engineers must wait for expensive, early versions of hardware for development and testing. Integration and testing of components are costly and complex due to limited prototypes and numerous vehicle variations, such as different engines, trim levels, or country-specific requirements.

The capacity that cloud environments offer for infinite scalability and cost reduction, along with the use of virtual electronic control units (vECUs) or virtualized cars, presents a solution to these challenges, as shown in Figure 5-4.

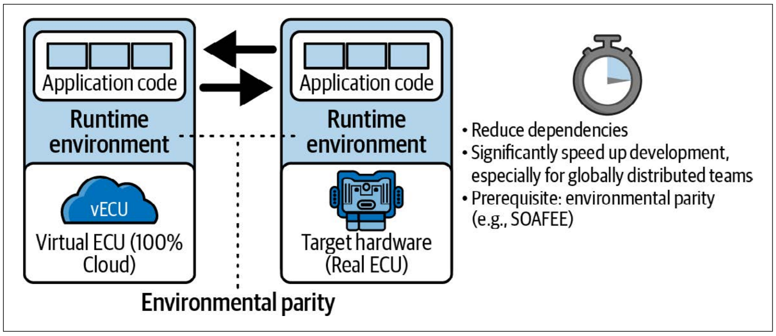

Application code
Runtime environment
vECU
Virtual ECU (100% Cloud)
Application code
Runtime environment
Target hardware (Real ECU)
Environmental parity
• Reduce dependencies
• Significantly speed up development, especially for globally distributed teams
• Prerequisite: environmental parity (e.g., SOAFEE)

Figure 5-4. Virtualization

Summary

To deliver on the vision of the "habitat on wheels," with rich cross-domain applications and data fusion delivered 10 times as fast, OEMs must overcome significant impediments presented by functional safety requirements, technical constraints, and organizational constraints ("clash of two worlds").

The vehicle OS can be a powerful platform, enabling rapid application development in the QM world via service-oriented architectures, OTA, and vehicle app stores. Combining the SDV and AI is important for data-driven applications.

To manage "development at different speeds," OEMs must embrace VSM and use hardware abstraction and vehicle APIs to create a loose coupling, not only on the technical level, but also on the organizational level between the digital and the physical value streams, as shown in Figure 5-5.

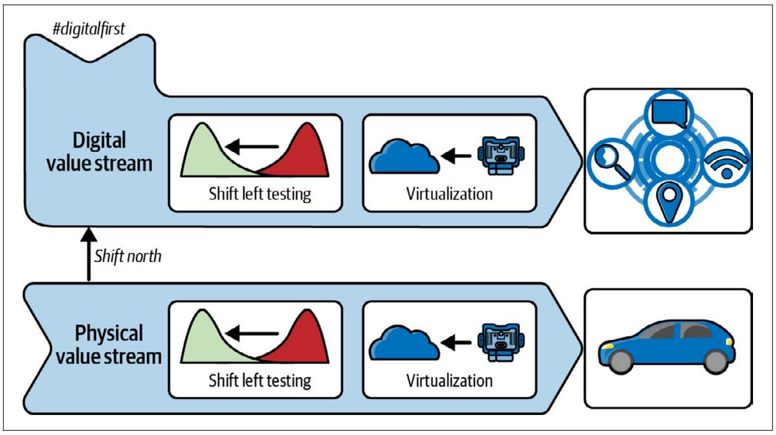


Figure 5-5. #digitalfirst and VSM for the digital.auto

#digitalfirst is a new way of working that combines three tectonic shifts:

Shift north
: Breaking up event chains and focusing on QM application development north of the vehicle API

Shift left
: Left-shifting user tests and getting early customer validation of the integrated digital/physical experience

Shift toward virtualization
: Further decoupling of hardware and software development, thus increasing agility

Together, the SDV and #digitalfirst can deliver a number of benefits for OEMs and the entire supply chain. Early and continuous user feedback helps ensure that digital investments are paying off as intended. Avoiding investments in unwanted digital features helps ensure that development capacities are used where they create the highest customer value. Significantly increasing development speed (10x) provides the agility that is required to react quickly to changing customer needs and preferences. The divide and conquer approach enabled via hardware abstraction and specialized value streams helps manage the complexity of today's automotive world. Virtualization helps reduce the cost and complexity of managing too many hardware prototypes.

CHAPTER 6

Next Steps

We hope you find the concepts outlined here helpful for your daily work. Should you be interested in finding out more about how we work and how you can get involved, please visit digital.auto.

We helped co-initiate digital.auto as an open and vendor-neutral community to enable our industry to use the SDV to deliver all the exciting use cases we have been talking about here. We do this via different collaboration and thought leadership activities, including this publication. In addition, the digital.auto community has worked together to create a number of open source activities.

Figure 6-1 maps the digital.auto focus areas against the #digitalfirst SDV lifecycle introduced in Chapter 1.

First, we focus on advancing methods and tools to support early stage virtual exploration of SDV experiences. In particular, we created the autowrx open source project in the Eclipse SDV working group. This is a cloud-based, rapid-prototyping environment for the SDV. A free-to-use instance can be accessed via playground.digital.auto. This playground can be used to rapidly try out new ideas for the SDV against real vehicle APIs, which are simulated in the backend to get realistic test data. The resulting prototypes can be used to get early customer feedback and learn more about requirements, including the APIs needed for the new application. Prototypes developed in the playground can be directly deployed onto real SDV platforms (e.g., the Eclipse Velocitas open source SDV runtime). The digital.auto playground is open source and free to use. You can try it out at playground.digital.auto. We look forward to your feedback!

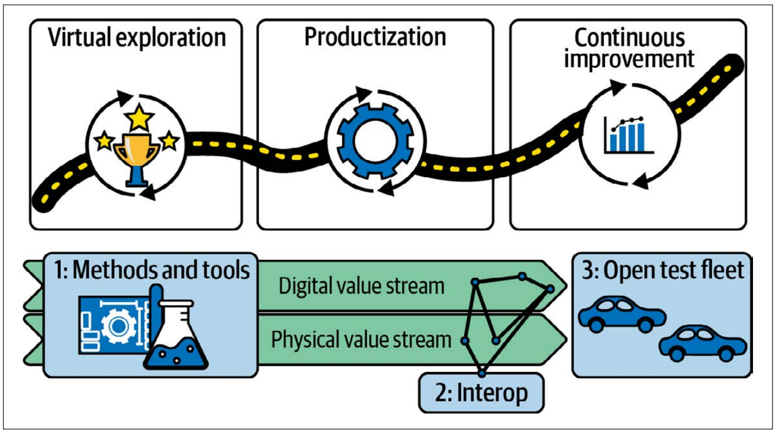


Figure 6-1. The digital.auto focus areas

Second, we are fostering interoperability along the SDV value chain. To support a #digitalfirst approach, a number of tools and technical platforms must be seamlessly integrated, from early-stage testing over virtualized development in the cloud to deployment on the target vehicle hardware. The site interop.digital.auto summarizes our current activities in this space and provides a number of interop proof points we have implemented in the digital.auto community.

Third, we believe that to unlock the full potential of the SDV, we must create a vibrant ecosystem of OEMs, start-ups, developers, and innovators, all working together to try out new, exciting use cases. This will require a test infrastructure for SDVs that is not yet available. Our vision is to work together on an open test fleet for SDVs, including an open app store for experimental new features, executed in a safe environment. If this sounds intriguing to you, visit www.digital.auto and contact us to help make this vision become a reality.

About the Authors

Dirk Slama is a vice president at Robert Bosch GmbH and chair of the digital.auto initiative. He is also the director of the AIoT Lab at the Ferdinand-Steinbeis-Institute, where he holds a full professorship. Dirk has 25 years experience in large-scale IT projects in automotive, manufacturing, finance, and telecoms. He is coauthor of four books, with more than 50,000 copies sold. His academic credentials include a PhD in information systems, an MBA, and a diploma in computer science.

Achim Nonnenmacher is driving software-defined vehicle innovations as a senior manager at ETAS GmbH (a Robert Bosch Subsidiary). As cochair of the digital.auto initiative, he helps speeding up the use case-driven adoption of new technologies in the automotive industry. Before this, he led product innovations at scale in the mobility sector by validating business, technology hypotheses and user needs for strategic projects. Achim holds a PhD from Swiss Institute of Technology (EPFL), an M.Sc. in physics, and an executive degree on innovation acceleration from UC Berkeley.

Thomas Irawan is president of ETAS and chairman of the Board of Management. Before joining ETAS, Thomas spent 16 years at Robert Bosch GmbH in a number of manufacturing, quality, development, and engineering leadership positions. He was technical plant manager of Bosch Thailand and served as SVP of quality management in the Chassis Systems Control division. Most recently, he headed the driver experience business unit for assisted, automated, and connected driving in the Cross-Domain Computing Solutions division. Thomas earned his doctorate in physics from the University of Dortmund, Germany.