White Paper

# Beyond the Edge

The Evolution and Future of
The Software-Defined Vehicle

The evolution of vehicle technologies has empowered automakers to expand the sophistication of driver and passenger features without compromising safety. The increased demand for more advanced features like autonomous driving, coupled with the value of the terabytes of vehicle data, will further drive trends like connectivity of vehicles, consolidation of functions on shared hardware, and complexity of automotive systems. And these forces will continue to challenge OEMs to redefine automotive architectures and to adopt a software-first approach to their designs.

This white paper explores how the software-defined vehicle (SDV) is propelling the future of transportation. It provides the insights behind the SDV to help leaders make strategic decisions for today and into the future. It also explores the technologies that have been developed to meet new feature demands—from those within the vehicle (in the edge) like high-performance computing technologies that enable responsive driver experiences while maintaining safety, to those beyond the edge like cloud technologies that enable the continuous maintenance and improvement of vehicles after sale.

More software can be found in the modern vehicle than in a passenger jet[1] thanks to a growing array of ADAS and self-driving technologies, connectivity-dependent features, and infotainment expansion. In fact, according to McKinsey[2], modern vehicles are estimated to contain more than 150 million lines of code. Clearly, software is now the main driver in new vehicle development, placing the automotive industry in the era of the SDV. Automakers are now shifting from a focus on hardware-centric designs to the design of software-centric computers on wheels. And this shift can be seen in the trajectories of traditional automakers and new startups alike. This is affecting not only the automotive OEMs, but their global suppliers.

But what does SDV mean and how does it impact your business? This white paper explores how the SDV is propelling the future of transportation. Whether your company has been building cars for one hundred years or one, we'll provide you with the context and the insights behind the SDV to make strategic decisions for the future.

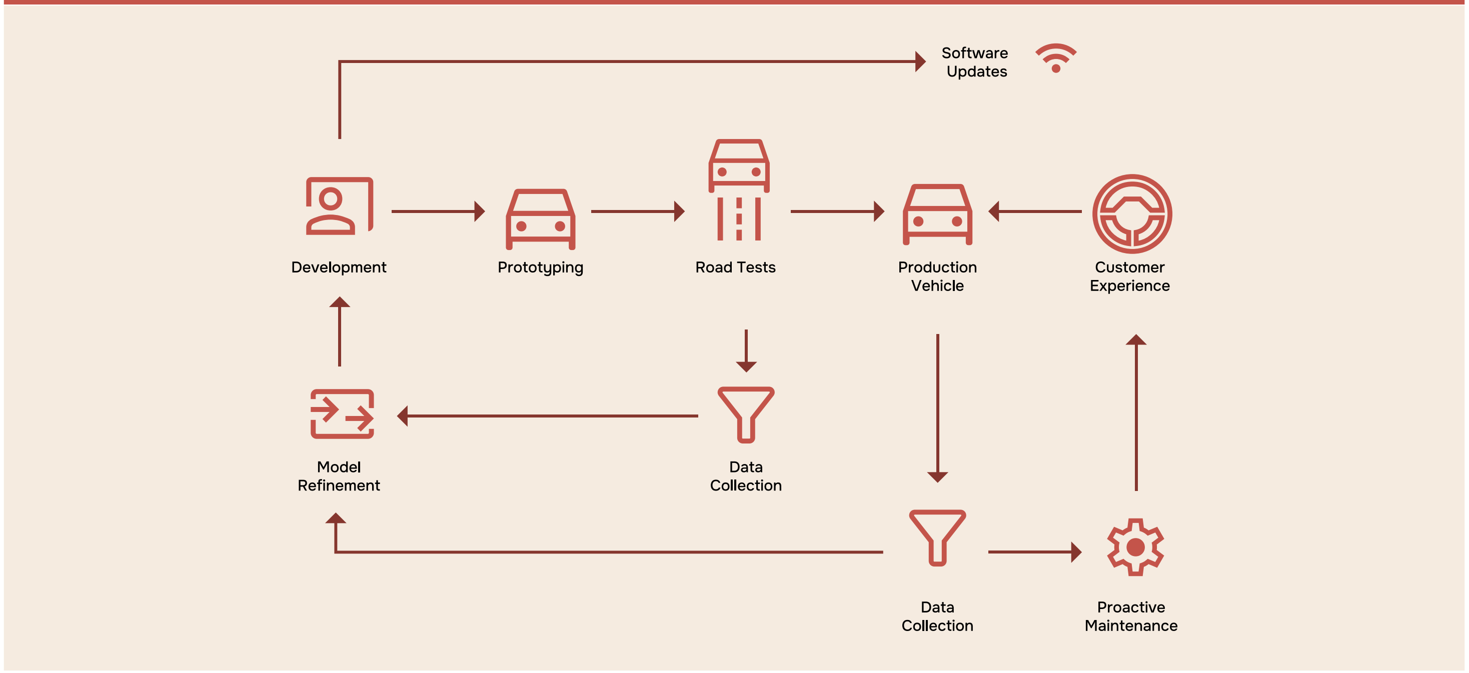
1. Millions of lines of code in modern aircraft and cars compared | CHOICE
2. Rethinking car software and electronics architecture | McKinsey

## What Is the SDV?

There isn't a single industry-standard definition for the SDV. For our purposes it is a vehicle where many features are enabled by software, the software requires both safety and non-safety domains, and the software-defined features are changeable throughout the lifetime of the vehicle.

While today's traditional vehicles have a lot of software, the SDV has many of its mechanical and hardware systems completely under software control. With an SDV, consumers can get software updates that change not only the user interface, but the driving behavior, long after they've driven the vehicle off the lot. This continually improves vehicle systems and provides updated, new, and customized experiences.

### SDV Lifecycle



The SDV lifecycle is an example of continuous improvement; feedback from driving tests and consumer usage drives software updates that continually improve the product and strengthen customer relationships long after the vehicle has been sold.

## The Road to Software

The automotive industry has been gradually moving from building hardware-driven machines to building SDVs over the last two decades. During this time, four significant trends have been shaping customer expectations and driving automakers to increasingly turn to software to address them.

- **Connectivity.** The industry has evolved from thinking about connectivity as an enabler for safety and security applications to a critical capability. This connectivity not only lets automakers change their vehicle's software and improve the in-vehicle experience, it also builds better relationships with customers. Equally important, connectivity to the vehicle allows automakers to collect and capitalize on vehicle data to inform their next-generation products.

- **Autonomous driving.** The race to develop autonomous vehicles relies on machine learning and artificial intelligence and requires huge amounts of processing power with sophisticated software stacks. Because this technology is at the cutting edge and has tremendous safety needs as well as societal impacts, autonomous software must be updated continually throughout the life of the vehicle to be both market-viable and safe.

- **Sharing.** This trend focuses on business models other than the tried-and-true personal ownership model: car sharing, ride sharing, and subscription models. By adapting the car for each new driver or passenger, the OEM can keep customers loyal to the brand and offer personalized on-demand services. This personalization requires significant software support.

- **Electrification**. The move from internal combustion engines (ICE) to electric vehicles (EV) has rapidly accelerated in the last couple of years to the point where nearly all traditional automakers have announced schedules for a fully electric fleet within the decade. The simpler mechanical design of an EV has also allowed many new automakers to enter the market. Since software is at the heart of an EV[3], both types of automakers are starting with fresh software-dominant architectures and designs.

Taken together, connectivity, autonomous driving, sharing, and electrification (CASE) provide automakers with powerful motivation for creating the SDV. While the CASE trends are perhaps the most visible ones, cost pressure and value considerations are also driving more software definition into vehicles for both safety-critical and non-safety-critical applications.

- **ECU consolidation.** The trend of ECU consolidation[4] —replacing many independent vehicle modules with a single more powerful module—is primarily driven by the relentless need to reduce costs. ECU consolidation removes plastic housings, mounting hardware, wiring harnesses, extra microcontrollers, chips, and circuitry resulting in huge savings. It also combines the functionality of components duplicated across modules like processors, RAM, and flash, moving many pricy specialty components off the car's bill of materials (BOM) and replacing them with one more commonly available variant for less total cost.

- **Software value.** One final trend is the commoditization of hardware and mechanical components. This makes it difficult for both automakers and their suppliers to add value and differentiate – unless they do it through software. Similarly, the consumer's idea of value in a new car is no longer solely based on body shape and engine power; it increasingly comes from software-enabled capabilities such as a user-friendly navigation system, smart voice assistant, intuitive UI, and reliable hotspot.

3. Wall Street Journal: Electric vehicles are more software than hardware
4. BlackBerry QNX working with Android Open Source Project (AOSP) for Virtualization in Automotive Digital Cockpits

## Additional SDV Benefits

Beyond the trends driving this new software-centric approach and the benefits they give automakers, the SDV has several other advantages over its fixed-function predecessor. Although this isn't an exhaustive list, here are a few of the more noteworthy.

- **Commercialization**. The ability for automakers to extract additional value from new features and customized services throughout a vehicle's lifetime moves the automaker business from a strictly per-product pricing model to a more profitable subscription or service model.
- **Loyalty.** Because of its ability to readily adapt and improve, the SDV fosters brand loyalty. Customers who feel cared for by their automaker are more inclined to return for future sales.
- **Abstraction.** A properly architected SDV necessarily abstracts the software from the underlying hardware. This insulates the automaker from the pain of hardware changes, making it far easier to adapt to situations like the ongoing semiconductor shortages, deprecated silicon product lines, and upgrades to increase capabilities.
- **Resale.** Because the SDV continues to improve through feature upgrades years after it's been sold, it holds its value better, raising its resale worth.
- **Data monetization.** As the vehicle becomes part of the internet of things (IOT), it transfers one to two terabytes[5] of raw data to the cloud each day. Automakers use this data not only to enable continuous product and service improvements but also to create recurring revenues from premium connectivity services, point-of-sale (POS) advertisements, and personalized content.

5. Unlocking the full life-cycle value from connected-car data

## Impacts of the SDV

For all the gains that the SDV provides, there can certainly be some challenges. This is especially true for established automakers who need to make several adaptations and changes to current methodologies in order to realize the SDV.

### In-House Software

Because of the growing importance of software, automakers must move away from the supplier black-box procurement model and build their own in-house software competency. This means designing and developing software within the organization instead of outsourcing to Tier 1 or other suppliers.

The dramatic increase in software capabilities may require OEMs to hire whole new departments, acquire small software companies to redeploy their expertise, and/or build up new partnerships with software specialist suppliers. It may also require building up IT systems and new testing techniques to handle new software development and maintenance workflows.

### Software Centralization

As mentioned, automakers are consolidating ECUs whenever possible. This means centralizing vehicle software in a single, well-structured software application on hardware platforms with enough compute power to grow. Much of the ECU software will have safety implications, but some will not. Any centralized approach must gracefully handle this critical difference, allowing automakers to scale and reuse their software platforms and to change their vehicle's capabilities.

However, it also implies changes to vehicle wiring architecture and harnesses. It means less reliance on "off-the-shelf" ECUs. And it generally requires a holistic rethink about how the vehicle is designed.

### Purchasing

The SDV impacts the entire company including departments like procurement, which must adapt to the different reality of software. Pivoting large organizations with well-entrenched procedures can be a significant challenge.

Automakers can no longer assume that software pricing will follow the same economies of scale established by hardware and that they can avoid software-specific terms and conditions. These factors may imply changes to supplier agreements, supplier selection processes, or supplier relationships.

### Mixed-Criticality

New automakers avoid many of the challenges listed above because they don't have existing systems that require adaptation. However, the design of the SDV does introduce a problem that every automaker needs to solve—mixed-criticality.

The SDV does not make functional safety requirements like ISO 26262 go away. Indeed, there is even more importance on safe software execution when vehicles drive themselves. Simply put, centralizing the vehicle's software into one module mixes safety-critical systems and ADAS capabilities alongside other vehicle systems like infotainment and mobile phone connectivity that are unnecessary or impossible to safety certify. This drives sophisticated solutions such as hypervisors, virtualized environments, and mixed operating systems into the SDV design. These require careful consideration within the SDV architecture to ensure they are properly supporting both safety-critical and non-safety systems as well as development workflows.

## SDV Building Blocks

To really understand where the SDV is going and how it's going to evolve, we need to understand how it's built. That is, what are the technologies that make up the SDV? While we don't lay out an actual blueprint (we do that in this video), we do talk about the significant changes to development tools and processes that make the SDV unique.

## Development

The SDV requires new software development methodologies to live alongside traditional embedded processes. This blend of cultures results in a dramatic mash-up of agile development and rapid innovation with rigorous testing and functional safety.

## Lifecycle

In standard automotive development methodologies such as waterfall or V-model, development is finished once software complies to a given specification. It works well with specification-based development as it divorces the OEM from the actual development process, allowing them to switch suppliers if necessary. This development also causes a “ship and forget” model, where software is only touched after the vehicle ships when necessary: to fix critical bugs or recalls.

The SDV demands updates throughout the vehicle lifespan, and abandons a fixed path of software development. The software process becomes much more responsive and adaptable to short-term requests and rapid market dynamics. While the initial time to develop the software isn’t shortened, all subsequent releases will be. Of course, we cannot lose sight of the safety demands of the car in implementing this agile approach. The safety demands and certification requirements of each component and the entire system must be considered throughout planning, design, and implementation.

If done properly, the vehicle’s software—like modern applications—becomes a periodic snapshot from a changing and continually maintained code base. This requires different workflows and responsibilities, but it transforms the vehicle innards into a software platform that can be replicated for multiple vehicles instead of developed for each independently.

## Reuse

Standard automotive development often recreates a “clean-room” version of functionality found outside the automotive world but using automotive-approved tools. While it may allow code to adhere to automotive standards more easily, this practice of rewriting code does not reuse software and contributes to a much longer time to production.

There are already fundamental building blocks that provide value and offer much of the functionality automakers need. By using open-source libraries, leveraging existing software functionality, and borrowing tools from web development, automakers can refocus their software firepower on differentiating areas. Of course, automakers also must apply proven techniques to maintain needed levels of reliability and functional safety with any outside code that’s imported into the vehicle.

## Cybersecurity

Embedded software came of age when cybersecurity issues weren’t a concern. Unfortunately, a significant part of existing automotive development practice still is coming to terms with this understanding. Modern software acknowledges that, in exchange for the tremendous benefits that connectivity provides, we need to actively manage cybersecurity concerns and build software to remove vulnerabilities whenever possible. Building the SDV requires cybersecure practices such as defense-in-depth and principle of least privilege, new testing techniques such as penetration and fuzz testing, and software that’s architected with cybersecurity from the beginning—not bolted on at the end.

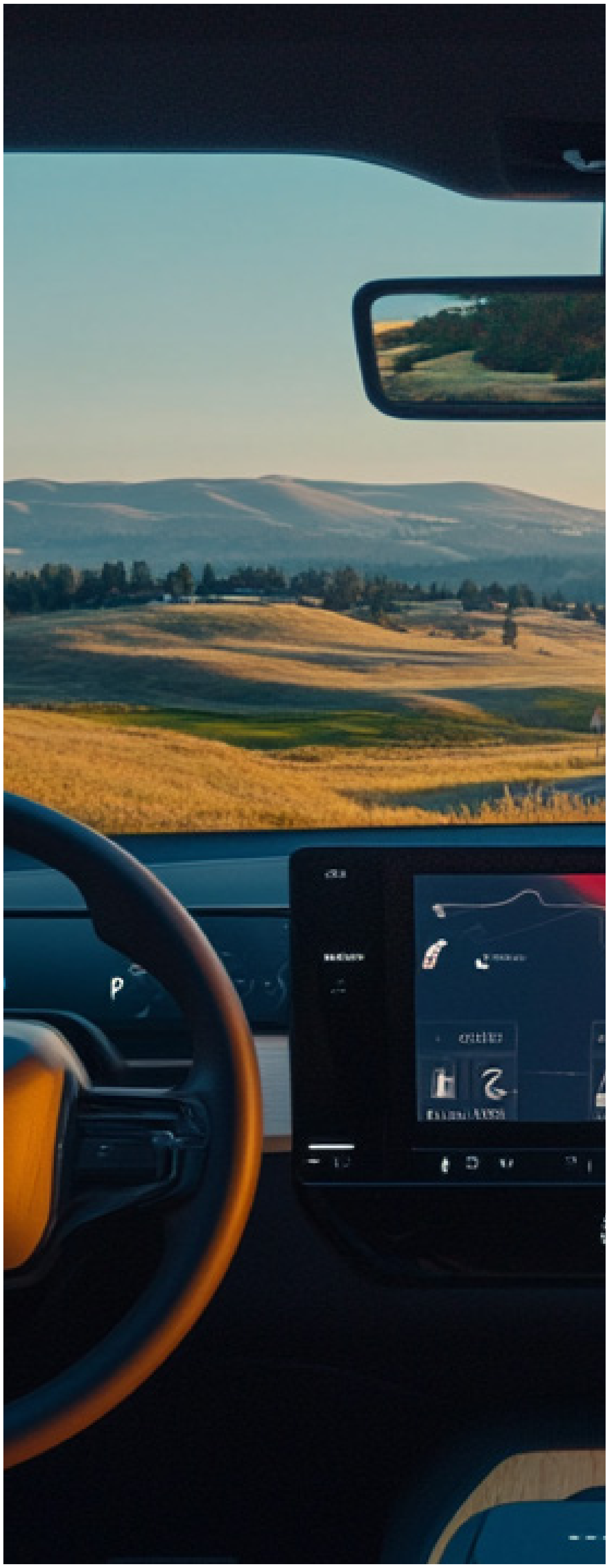
## Tools

Modern development tools are reliant on the Internet, full stop. For IT departments that are used to completely controlling the software engineer's development environment and providing all the necessary tools, this requires a major adjustment. SDV development in particular, requires software repositories that live outside the company's walled garden and development tools that are browser-based or externally hosted. This is also true with the complex systems used in self-driving where engineers must use cloud resources across multiple suppliers and third-party companies. As a result, the OEM's IT department must learn new ways to secure corporate assets while flexibly working with internal engineers and external parties.

## The Edge (Inside The Vehicle)

Connected cars and trucks are at the edge of the automaker's network, and the edge is where the most software modernization is required. The SDV requires an extension of techniques already prevalent in automotive software, such as domains. Within the car, a domain consists of a set of related features (such as ADAS, body functions, or user experience), while a domain controller consolidates those features into a single module.

This decomposition of the vehicle's features by domain is the natural conclusion of ECU consolidation and using domain controllers to tame vehicle complexity has been a phenomenon gradually gaining traction. An SDV contains a connected collection of individual domains acting as one, consisting of one or more system-on-a-chip (SoC) modules. At the level of the individual SoC there are design pressures and solutions to address them, including high-performance computing platforms, hypervisors, and microservices.



## High-Performance Compute

Relocating software tasks from hundreds of distinct microcontrollers into a single SoC brings us to one obvious conclusion: the SDV's SoC needs to be at least as powerful as the hardware it replaces. This fact, plus the SDV's additional sophisticated processing demands, leads us to high-performance computing (HPC) platforms within the vehicle.

The HPC in the vehicle implies Gigahertz processing speeds, 64- and 128-bit CPU architectures, multiple cores, and many multiple parallel processing units from graphics processing units (GPUs). The in-vehicle HPC is expected to process efficient workloads for hundreds to thousands of tasks.

## Heterogeneous Design and Hypervisors

As mentioned before, the consolidation of the vehicle's functionality onto a single HPC system means that we need to be able to mix software of differing criticality. A mixed-criticality environment allows us to use a consumer-grade OS to execute streaming applications, mobile phone connectivity, WiFi hotspots, and other noncritical components while we can use a safety-certified and hardened OS to run self-driving software, vehicle X-by-wire systems, battery charge and monitoring, ADAS features, and other safety-critical features.

To implement these mixed-criticality features, a heterogeneous design uses a safety-certified hypervisor to place software into different virtual machines (VMs). (Not all software requires a VM: some of these safety-critical hypervisors may even be able to run high-priority, low-latency, and safety-critical tasks directly within the hypervisor environment.) The hypervisor VMs allow different operating systems to run side-by-side on a single SoC. Because the hypervisor allows hardware separation of different safety integrity levels (SILs), the development team can apply different test methodologies to software in these distinct compartments as needed, letting them employ rigorous safety certification just to the parts of the system that are needed.

## Microservices

A microservice architecture splits the division of labor into many independent smaller components. Each component —a microservice—provides a small, dedicated set of functions. This modularity lets each microservice be easily replaced or upgraded without impacting the rest of the system in contrast to a more traditional monolithic design where functionality is dispersed across a handful of very large and complex applications and can't easily be removed or substituted.

A microservice architecture is one of the best ways to achieve platform abstraction—in other words, creating a cross-platform code base that decouples the software from the hardware. This is a necessary step if automakers want to maximize their software investment in as many vehicles as possible across their fleet. Microservices also help in scalability, letting more powerful vehicles spin up several instances of the service, or more powerful versions—all without changing the main application.

Microservice management, however, requires technologies that protect and isolate each independently running service. Current options are virtual machines, containers, cloud-native binaries (using WebAssembly), or other OS-specific protections. Each technology has pros and cons—situations where they may not work at all or conversely where they may be absolutely required. A completely flexible SDV architecture will require multiple —perhaps all—of these technologies to work together. Orchestrating these technologies together in an optimal way remains one of the larger complex and unsolved problems in building a fully capable SDV today.

## Beyond The Edge (Outside The Vehicle)

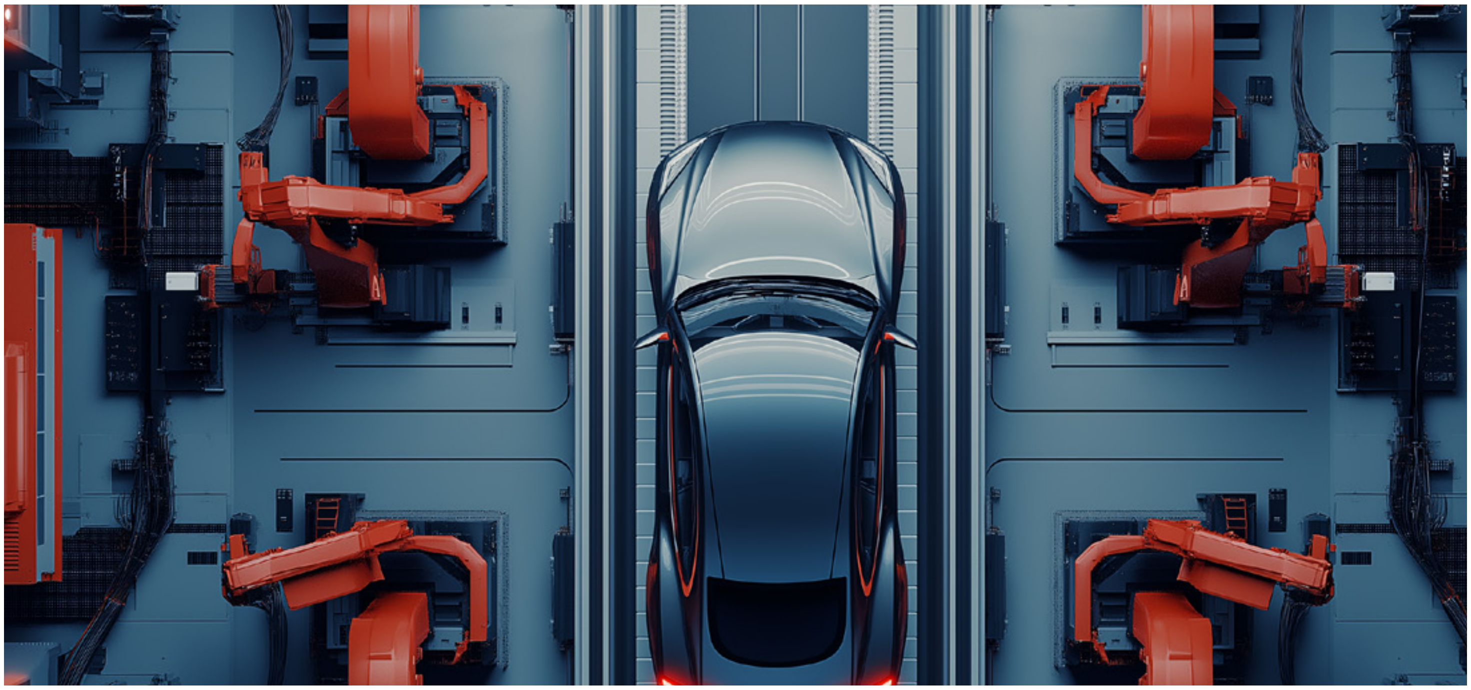
As mentioned, some of the in-vehicle technologies the SDV requires are leveraged from modern cloud software development. But the automaker technology portfolio isn't just changing within the car. Other aspects critical to SDV development are found outside the vehicle.

### AI and ML

Self-driving is a dominating feature within the SDV—one that would be impossible to build without incorporating modern artificial intelligence (AI) techniques. A self-driving vehicle uses machine learning to interpret camera images for lane markings and lidar and radar for surrounding traffic and obstacles. This process requires repeatedly training algorithms with huge amounts of simulated or real driving data so that they can handle as many conceivable real-life situations as possible. While the AI algorithms run in the car, the big data sets and processing power to train the AI system are all outside it.

No automaker wants a self-driving vehicle to make poor decisions when encountering novel situations, which means that the self-driving algorithms must be fed with as much data as practical. However, using poor quality or insufficiently broad data can lead to implicit bias and/or overfitting of the model. Both insufficient data and bad data can introduce problems and blind spots in the vehicle's behavior. This puts a huge premium on gathering large amounts of high-quality data that is consistent across all model lines and trim levels. Unifying data from multiple vehicle years and models, along with changing software versions require a thoughtful approach to maintaining clean data architectures and processes that can deal with this fuzziness without corrupting valuable models or drawing incorrect conclusions.

The SDV can help with this by using on-the-road vehicles as collection agents. The SDV and self-driving refinement are in a virtuous circle. Cars are tools to collect data, that data improves the driving performance, and those improvements are downloaded into cars, continuing the improvements for the next round of data collection.



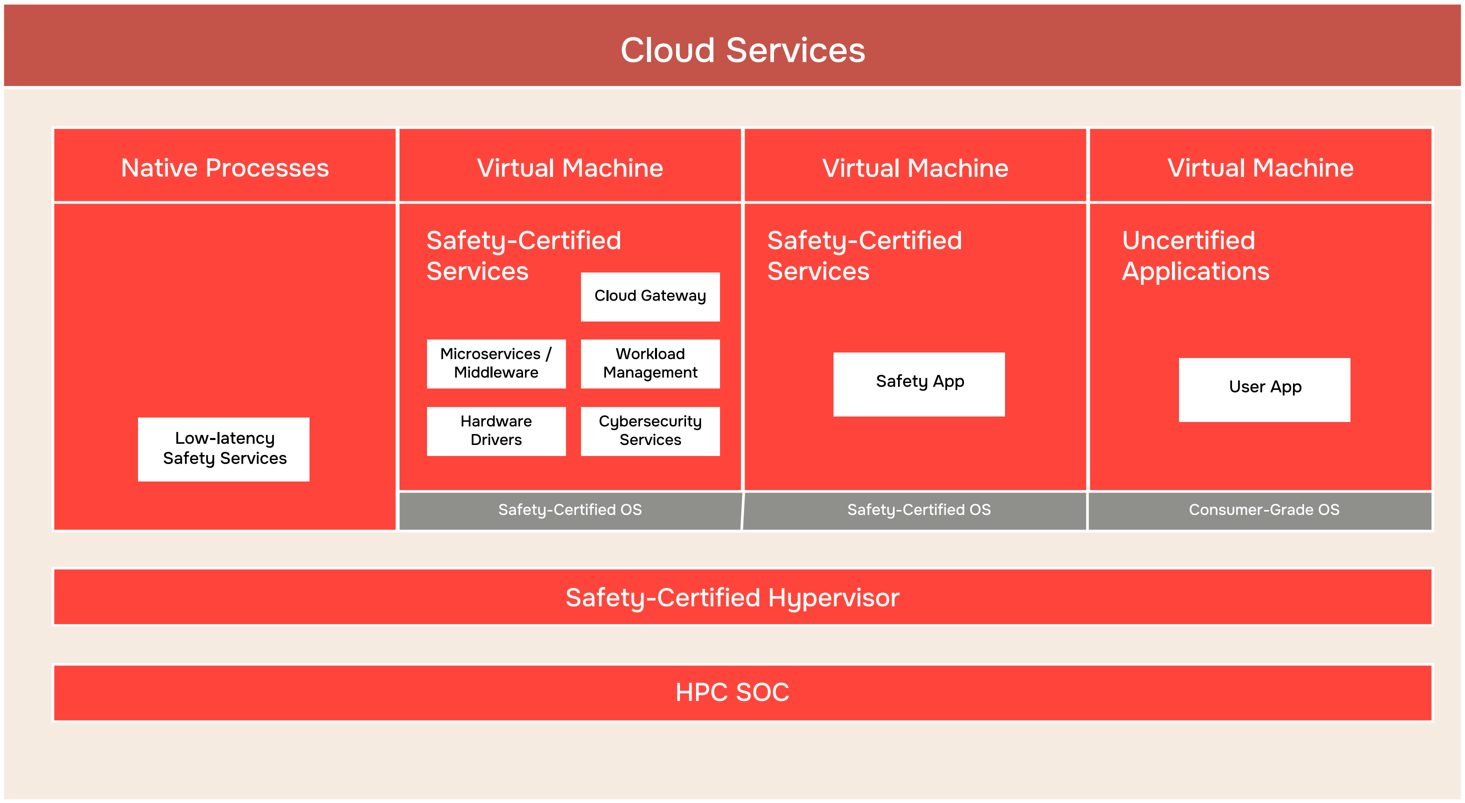
## The Cloud

One of the advantages of the SDV is that, due to its connectivity and flexible architecture, it can access resources in the cloud when necessary. This allows the vehicle to provide services that would be completely impractical to implement wholly within the car. Possibly even more important, software can offload non-safety-critical software tasks from the vehicle to the cloud when necessary due to complexity, timing, cost, or other factors.

Cloud software has two primary advantages over software running in the vehicle. First, it has nearly indefinite computing power at its disposal. Second, changes to the software in the cloud simultaneously affect the software in all vehicles.

Cloud services are often implemented with a service-oriented architecture (SOA) and integrating them into a vehicle application is often easier if the vehicle already employs a microservice architecture—something that we've already discussed as an SDV staple.

Another benefit is the automaker can build an SDV that can dynamically allocate non-safety-critical services in the cloud or in the vehicle as required. An architecture with a blurry line between car and cloud software execution can give the automaker a tremendous advantage, but it does require a sophisticated system that knows how and when to shift resources back and forth between the cloud and the edge.



The high-level software architecture for an SoC in the SDV provides multiple methods of extensibility.

## SDV Architecture

The most dynamic characteristic of an SDV architecture is that it can be extended both internally with new applications and OS instances, as well as externally through cloud-based interfaces. Because of the mix of safety and non-safety critical software, the SDV must run individual functions within software isolation mechanisms (such as VMs, hypervisors, containers, and Cloud-native binaries) that protect the rest of the vehicle software against any new extensions in case they don't operate as intended.

- **Hardware:** Supports several heterogeneous operating systems, multiple levels of abstraction, and all the necessary middleware layers, absorbing the functionality of dozens of other processors within the vehicle.
- **Safety-Critical Hypervisor:** Provides a heterogeneous, mixed-criticality environment.

.

- **Safety-Certified Services:** Contains many common services required throughout the platform by safety and non-safety applications alike.
- **Microservices:** Contains a wide variety of commonly used services and middleware components such as policy management, diagnostics, monitoring, logging, service discovery and directory, state management, messaging, multimedia, and connectivity.
- **Hardware Drivers:** Provides resources for all applications such as memory, graphics, power management, and audio drivers, as well as those dedicated to applications with safety requirements like the vehicle bus, camera, lidar, ultrasonic sensors, steering, powertrain, and battery management.
- **Cloud Gateways:** Provides an interface to the cloud so that vehicle applications can call cloud services as well as access over-the-air downloads.

    - **Workload Management:** Coordinates starting non-safety reliant services on the device and in the cloud and orchestrates moving resources between the two using the cloud gateway.

    - **Cybersecurity:** Includes in-vehicle services such as authentication, encryption, code signing, certification management, intrusion detection, intrusion prevention, and firewalls.

- **Safety-Certified OS:** Runs the safety services listed above.

- **Safety-Certified Applications:** Contains all the vehicle applications that require safety certification.

    - **Safety Apps:** Includes applications such as advanced driver assistance services (ADAS) like Lane Keep Assistance Systems (LKAS), blind-spot detection, and adaptive cruise as well as other safety-critical applications like steering and braking, instrument cluster, sensor fusion, and self-driving.

    - **Safety-Certified OS:** Provides low-level support and running environment for safety applications.

- **Uncertified Applications:** Provides the environment for vehicle applications that don't require safety certification.

.

    - **User Apps:** Includes applications that don't require certifications like mobile connectivity, navigation, music streaming, voice assistants, driver personalization, and vehicle visualizations.

    - **Consumer-Grade OS:** Provides low-level support and running environment for consumer applications.

- **Cloud Services:** Contains services within the cloud that the vehicle may need, such as vehicle insights, location-based features, remote diagnostics, user experience augmentation, cybersecurity monitoring, traffic databases, vehicle provisioning, and fleet management.

Each of the applications running within the safety-certified OS or consumer OS is placed within its own container, insulating it from the rest of the system as well as making it easier for the system to create new instances.

## Blending The Edge

SDV architecture makes it possible to move non-safety relevant services fluidly between the vehicle and the cloud. This is incredibly useful for several reasons.

### Dynamic Services

A dynamic service application allows the automaker to easily roll out new services to the vehicle at any time. When cars and trucks are updated after release, they provide the benefits we've discussed previously about additional revenue streams, vehicle revitalization, and customer loyalty.

This helps improve development timelines, decoupling them from the vehicle production process. Ideally, every car would ship with every software feature fully complete. Without the SDV, the only options an automaker has if a feature isn't fully complete and debugged when production vehicles need to ship is to delay the vehicle release or rip the feature out. Neither one is a desirable option.

With the SDV, cloud-based services and downloadable software, vehicles don't need to have every feature compete when they're shipped. New features can even provide a burst of customer excitement when they become available.

## Cloud Augmentation

Applications within the SDV don't care if the services they call are running within the vehicle or in the cloud. Clearly, many software applications require immediate access by the user or real-time control of the vehicle mechanical systems—and these aren't practical to put in the cloud.

However, many non-safety reliant services can be successfully augmented with cloud services. Music streaming and voice assistants are clear applications of this today, but these applications require the Internet and breaks in connectivity cause them to fail. Services that are augmented by the cloud when connectivity is available but fail gracefully without it are preferable for the driver.

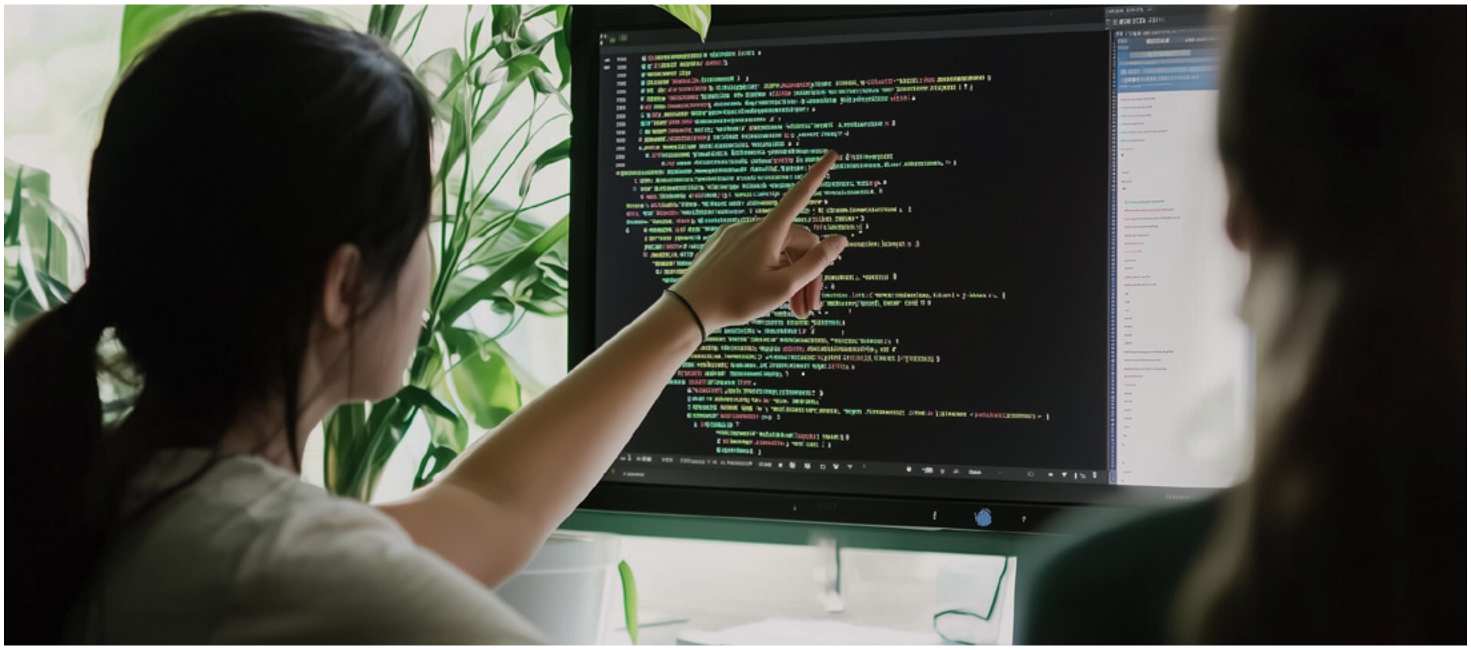
## Unified Platform

With the investments required to own and develop software, it makes sense for automakers to use their vehicle software platforms in as many vehicles as possible. Thankfully, the flexibility of the SDV architecture and its abstraction from the underlying hardware allows the same software to scale across many hardware platforms with a wide assortment of features and applications selectable by vehicle. With one single software-defined platform, the automaker can support every vehicle within their fleet.

## Conclusion

The era of the software-defined vehicle is here. While there are still several adaptations that automakers need to make to fully realize the SDV, the benefits will be well worth the effort. The integration of services between the edge and the cloud is just beginning. With the flexibility and abstraction that an SDV enables, tomorrow's cars will be continually improving, able to add functionality that isn't even dreamed of today.

The ideal SDV software architecture is quite complex. New techniques and approaches are actively evolving, and many practical challenges still exist in rolling out flexible solutions that can comply with safety-critical requirements while still working within modern software best practices. While we don't pretend to have the answers to every question, we're researching the challenges and building out software architectures that solve them, one at a time. If you're interested in working with us on any of the SDV challenges outlined in this paper, we'd love to hear from you.



July 2025

## About QNX

QNX, a division of BlackBerry Limited, enhances the human experience and amplifies technology-driven industries, providing a trusted foundation for software-defined businesses to thrive. The business leads the way in delivering safe and secure operating systems, hypervisors, middleware, solutions, and development tools, along with support and services delivered by trusted embedded software experts. QNX® technology has been deployed in the world's most critical embedded systems, including more than 255 million vehicles on the road today. QNX® software is trusted across industries including automotive, medical devices, industrial controls, robotics, commercial vehicles, rail, and aerospace and defense. Founded in 1980, QNX is headquartered in Ottawa, Canada.

**Learn more at qnx.com →**