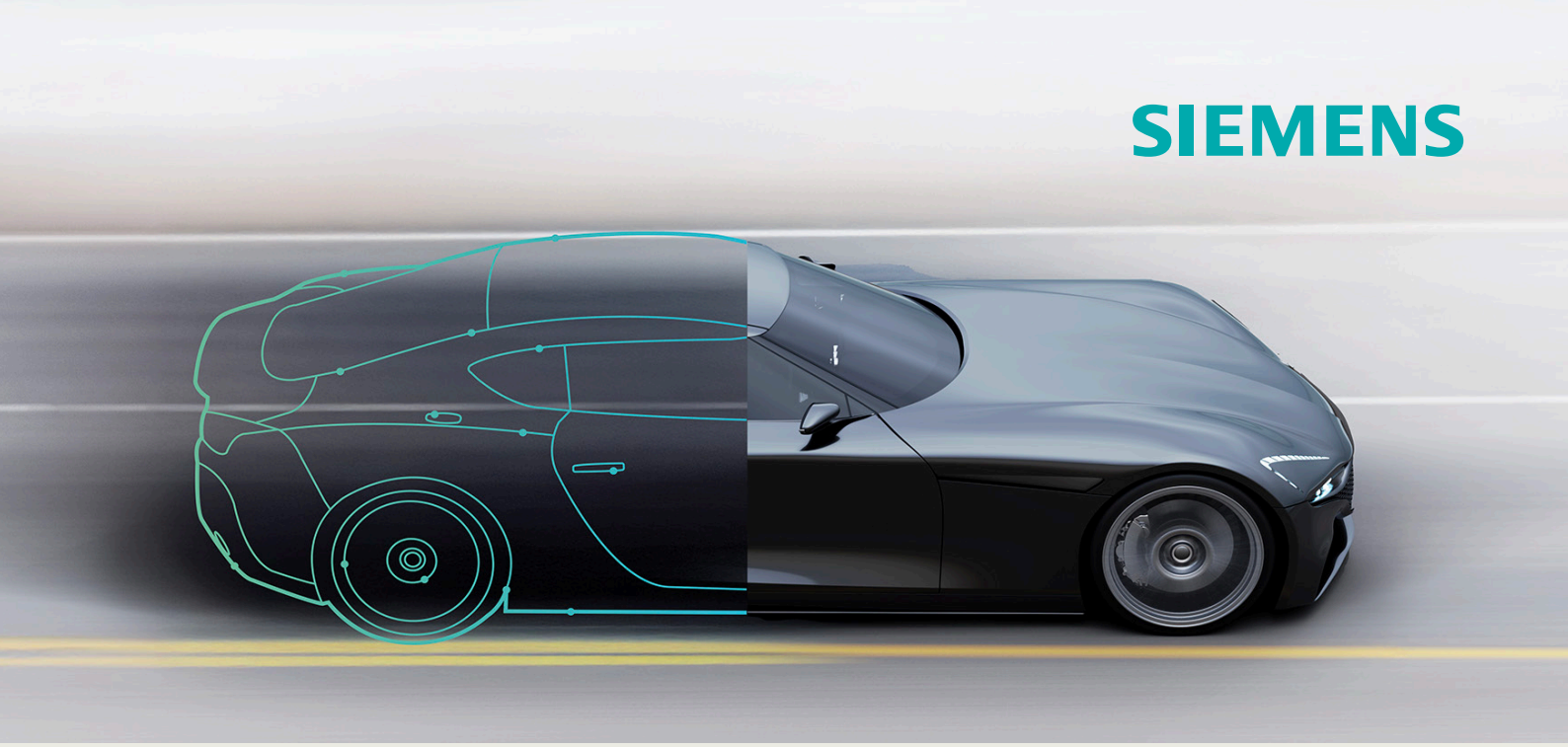
SIEMENS



PAVE360™ Digital Twin Solution

Software-Defined Systems for the SDV

Developers must consider the step change in complexity for the Software-Defined Vehicle (SDV)

SDVs are no longer cars, they are highly complex systems, extensions of the home, the office, entertainment and consumers lifestyle. They are rolling data centers, highly connected, and full of the latest technologies that need to communicate with other vehicles, the roadside and more. This requires a step-change as traditional development methodologies cannot hope to address these new complexities. Original Equipment Manufacturers (OEMs) need to change the way they develop their vehicles.

Until now vehicle developers have relied upon availability of hardware before implementing software. Now SDV developers will need to develop both hardware and software in parallel and use new methodologies to modernize the development process to keep up with SDV requirements.

The SDV needs a new development methodology based on "Software-Defined Systems" (SDS)

There are two new principles that must be at the forefront of SDV development

- **Be "software-defined"** – start with software because that's what consumers will interact with first and for most of the time. Hardware needs to be just another optimization variable to meet the needs of software rather than a fixed constraint. Developing software as early as possible in this new environment is now essential.
- **Be "systems-aware"** – due to the complexity of an SDV system there are potential failure mechanisms that only become apparent during product integration, or worse, when the vehicle is deployed in the field. For successful SDV development, it is critical to address these issues early in the design cycle to reduce risk of failure and ensure better integration and validation of the entire system. This requires a "threaded" approach connecting validation from system level down through unit testing and requirements fulfillment.

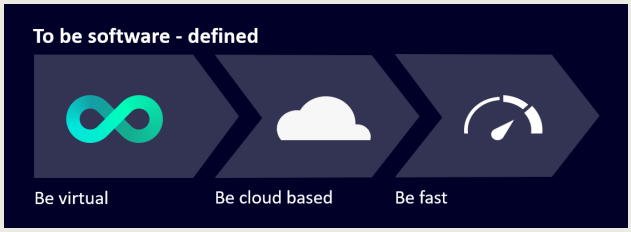
siemens.com/PAVE360

Software-Defined Vehicles (SDV) are a subset of Software-Defined Systems (SDS): a development methodology where software and hardware are flexible and refined in parallel along with connectivity to iteratively ensure the system under development meets requirements. The result is an optimal solution measured by performance, cost, manufacturability, reliability, safety and customer satisfaction. It extends the vehicle's lifespan and creates additional revenue models.

What does it mean to be "software-defined" and "systems-aware"?

To truly be **software-defined** a development environment is needed that is:

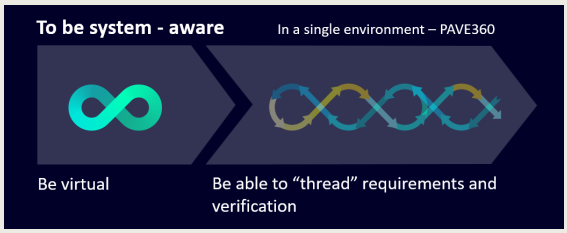
- **Virtual** – allowing the development of software on multi-fidelity virtual models before hardware is available and to allow for changes in hardware to meet software needs
- **Cloud based** – to enable collaborative development between teams, product makers and their supply-chains, and to leverage the performance afforded by high capacity compute available in the cloud
- **Above all FAST!** – Software developers are used to developing on real hardware. To date, virtual models (e.g.QEMU, AFM) have been too slow to keep up.



To be **systems-aware** you need an environment that can:

Link critical points of failure at system level to components and low coverage areas all the way through to requirements fulfillment. Again, this this environment needs to:

- **Mix and match –** to link together models at various levels of fidelity to create meaningful system level simulations with available virtual components and real hardware
- **Be able to "thread" verification** - from system level to unit level, this enables developers to detect integration failures early, isolate causes of complex system level failures, and reconcile low level testing against system level requirements



Delving deeper into the need for "Requirements and Verification threading"

In traditional automotive development flows, requirements are defined at high level and decomposed to more detailed requirements at lower and lower levels of abstraction until individual technology components, or units, are clearly specified. Each unit is then built to specification and verified against it's own set of requirements. Units are then integrated by connecting together real hardware running production software. The finished sub-systems and systems are then validated against requirements to ensure specifications for the overall system are met. More often than not validation against requirements at every level of abstraction was performed manually on paper or spreadsheets and in some cases actually physically "signed off" by a manager.

This was suitable when the industry was basing vehicle technology on discrete controllers, when software was deeply embedded and minimal in nature. For next generation SDV this methodology rapidly breaks down. This is driven by:

- The shift from multiple, isolated discrete functions to **integrated system approaches** like Service-oriented-Architecture (SoA)
- High demand for **advanced software functionality** in ADAS, IVI/HMI driving the need for high performance compute rather than microcontrollers
- **Increased interconnectedness** between internal vehicle systems and V2X, that was not present in traditional vehicle platforms
- **Increasing reliance on AI** to make faster decisions than humans for safety functions making it harder than ever to "sign off" traditionally on timing requirements

These driving factors result in much more complex integrated systems powering vehicle electronics platforms, making it impossible to manually reconcile verification and validation against requirements.

OEMs today are being affected by this causing cancellations of technology platform programs once they realize that their existing antiquated methodologies are not up to the challenges of SDV.

What's needed is an integrated, automated, and threaded approach to reconciling verification results against requirements through all levels of abstraction. Once requirements are defined they need to be linked to requirements in lower levels of abstraction, which then need to be linked to specific test cases run on simulation models in virtual space. When these test cases are automatically executed results are passed up to higher levels of abstraction, running in system simulations until the initially defined high level requirements can be proven to be met. No spreadsheets. No manual intervention. No room for human error when validating a hugely complex system.

PAVE360™ delivers on modernizing the development flow

The concepts of being **“Software-defined”** and **“Systems-aware”** are intrinsic to PAVE360, encompassing:

- Model-based, fast, hardware & software **architecture exploration** enabling objective, data-led decision making
- Threaded approach to **product definition and requirements**
- "Software-defined" approach to **design & implmentation** enabled by fast modeling & simulation environments
- Accurate and automated **verification and validation** against high level requirements enabled by threading and multi-fidelity simulation environments