PARASOFT®

WHITEPAPER

How to Address Software-Defined Vehicle Challenges With Test Automation

WHAT IS AN SDV?

As automobile electronics and the software that controls them mature, the concept of software-defined vehicles (SDV) is here to transform the automotive industry. At its core, an SDV is not merely the code that runs in the car but rather a new paradigm where a vehicle's features and functions are primarily enabled through software rather than being hardwired into its hardware components. This shift from fixed and hardwired to SDV means a range of possibilities and challenges, but many of the software development issues remain.

SERVICE-BASED FUTURE

An SDV operates on a connected car platform rooted in a service-oriented architecture (SOA). Services are provided via mobile networks and providers can provide new or updated software components with ease. One of the key advantages of this approach is over-the-air (OTA) software upgrades. This means vehicles receive new features, performance enhancements, and even critical security patches without the need for a visit to the dealership. An SDV can always be up to date and secure. For example, since software is more central to the features and performance of modern vehicles, an OTA could increase top speed and acceleration for your vehicle.

With these services provided by OTA vendors, there will be a transition to smaller, more manageable units, or microservices. These microservices can be developed, tested, and deployed independently, enhancing the agility of the software development process. Vehicle manufacturers have the ability to offer features and functionality as subscription services, enabling a pay-as-you-go model. While this provides flexibility for consumers to choose the features they want and when they want them, it also introduces a potential for increased long-term costs and subscription fatigue.

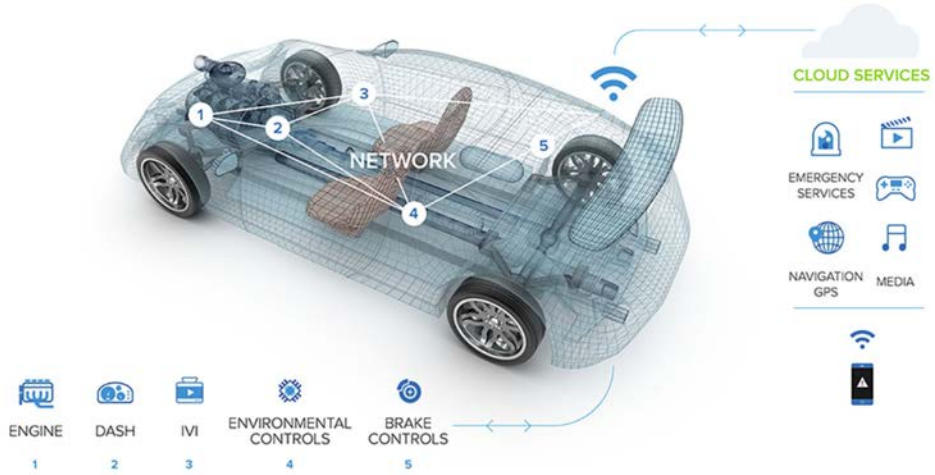


Figure 1: Connected car platform rooted in a SOA

FROM CONNECTED VEHICLE TO SDV

SDVs are intrinsically linked to the concept of connected vehicles. As such, the plan is to harness the power of the mobile internet to share data with other vehicles and infrastructure for real-time communication, enhanced navigation, and advanced safety features. Whether it's vehicle-to-vehicle (V2V) or vehicle-to-infrastructure (V2I) communication, the SDV becomes a node in the broader network, contributing to traffic and accident warnings, and eventually enabling smart cities.

This shift towards SDV also meets the increasing connectivity, computing power and features demanded of modern automobiles. Advanced driver assistance systems (ADAS) and enhanced infotainment systems require immense processing power. SDVs are designed to accommodate these computing demands plus be adaptable as these technologies improve. ADAS is still in its early stages. As its capabilities improve, for example, with better hazard detection, these upgrades can be deployed to vehicles without the need for new hardware.

SDVs are set to revolutionize the automotive industry. However, this shift to more reliance on software means an even higher focus on software development for automobile manufacturers. Given the already large role that software plays in an automobile, the software development approach requires change. A key aspect of this transformation is automation and, in particular, software test automation.

WHAT LED US TO SDVs?

SDVs are an evolution of existing and, to a large extent, a still-developing set of automotive technologies. The goals of SDVs are to encapsulate the technologies into the modern automobile and provide new features based on the services-oriented architecture. The technologies that have brought us to this point include:

- **Advanced driver-assistance systems (ADAS).** With the adoption of the anti-lock braking system, ADAS technologies have been in development since the 1970s. They also include features we take for granted such as stability control and traction control. By 2021, approximately 33 percent of new vehicles sold in the United States, Europe, Japan, and China had ADAS features.
- **Autonomous vehicles.** Experiments on autonomous vehicles (AVs) began in the 1920s and trials started in the 1950s. The first semi-autonomous car was developed in 1977 by Japan's Tsukuba Mechanical Engineering Laboratory. The development of autonomous vehicles has accelerated since the 1990s.
- **Telematics.** The use of telematics, which is a compound of telecommunication and informatics, in vehicles started to become more prevalent in the 1990s, with GPS navigation systems being a common example.
- **Connected cars.** General Motors was the first automaker to bring the first connected car features to market with OnStar in 1996. The features of connected cars have evolved over the years with approximately 88% of new cars expected to feature integrated telematics by 2025.
- **OTA.** The first original equipment manufacturer (OEM) to successfully perform OTA updates was Tesla. Since then, other manufacturers have started employing OTA software and firmware updates.

The aim of SDV is to incorporate these developing technologies and transition away from the traditional hardware-defined approach they currently rely on to a future where these features are possibly available a la carte and upgradeable.

HARDWARE ARCHITECTURE OF SDVs

The hardware architecture of SDVs is a layered structure.

- **Hardware platform.** The base layer that consists of the physical components of the vehicle, including the traditional electronic control units (ECUs) and other hardware elements.
- **System software.** This layer includes virtual machines, system kernels, RTOS, AUTOSAR runtime, and so on. It provides the necessary software infrastructure for the operation of the vehicle.
- **Application middleware.** Includes functional software, service-oriented architecture (SOA), and so on. It provides the necessary platform for developing and deploying applications in the vehicle.
- **Application software.** The top layer includes a smart cockpit human-machine interface (HMI), ADAS, AD algorithms, connectivity algorithms, cloud platforms, and the like. It provides user-facing applications and services.

In traditional vehicle architectures, ECUs are connected via a bus, like CAN, to central systems supporting control, programming, and diagnostics. However, in software-defined vehicles, the approach is shifting from using multiple ECUs to using virtual ECUs (vECUs) or even a single onboard supercomputer.

The individual ECUs are grouped by similar functionality and connected to a central domain controller, such as an IVI. Each domain controller is then connected to a gateway that can pass information between domains as required.

The SDV architecture is evolving from a domain-based approach to a zonal one. In a domain architecture, ECUs and cabling are organized into specific functional domains like the powertrain domain. Zone architecture groups domain functions based on their physical location inside the car. This transformation from a domain to a zone architecture facilitates the independence of sensors and actuators from the central vehicle compute node.

As for communication technology, Ethernet, an established, standardized technology, is rapidly evolving to support markets, including automotive, that have specific requirements, like the determinism and high bandwidth critical to autonomous vehicles. Gateways are used to facilitate communication between different domains. To achieve real-time, deterministic communication within the vehicle, Time-Sensitive Networking (TSN) is gaining traction in automotive applications.

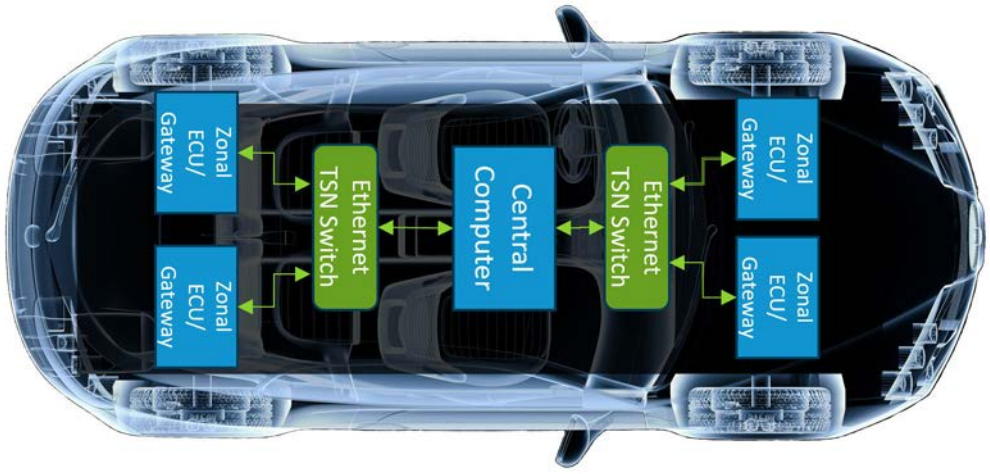


Figure 2: Sample of a SDV zonal architecture

STANDARDIZING SDVs

Standardization is a key aspect of the transition to SDVs. It involves finding consensus around the architecture and interfaces, as well as defining software parameters and hardware components. This fosters the development of an ecosystem of technology providers that can support the model.

The Eclipse Foundation has established the Software Defined Vehicle (SDV) Working Group. This group provides a forum for the organization to build and promote open source software, specifications, and open collaboration. The aim is a scalable, modular, extensible software platform that supports SDV application development and deployment. The working group aims for success by selecting and gathering consensus on various relevant subprojects. For example, the Eclipse openDuT project is concerned with the automated testing and validation of automotive software.

Standardization in SDVs will prompt demand for reusable and rapidly integrated software components. There is a big need to bring state-of-the-art technologies like virtualization, containers, Linux, and open source in general—which all come from the offboard areas—into the vehicle.

Over the next few years, we'll see a shift towards standardization in hardware abstraction layers (HALs) that will help drive the move towards virtualization. Thus, improving the portability of the software layers and moving towards common components shared amongst various brands. This is part of the broader trend of extending the software-defined everything movement to automobiles.

SDV ALTERNATIVES

The SDV initiative isn't the only software-centric vision for automobiles currently underway. For example, China has their own vision based on domestic vendors initiatives that are currently focused on their own version of AUTOSAR, called NeuSAR, an AUTOSAR-based system platform developed for next-generation automobiles. This specification includes safety and security functions, SOA protocol stacks, application dynamic deployment, and vehicle-cloud collaboration solutions.

The intent and purpose of NeuSAR and China's vision aligns with much, if not all, of the SDV goals. However, it remains to be seen if there will be compatibility between these two initiatives.

SAFETY AND SECURITY IN SDVs

The safety and security standards needed for current automotive software are expected to carry over and evolve with SDVs. In addition, there will be demands on the software to meet the goals of SDVs, such as portability.

Automotive grade. The software must be developed to automotive safety and security standards. This means it should be capable of functioning in a wide range of conditions and be robust, reliable, and secure enough to assure the demands of modern connected vehicles. This includes everything from the start to the end of the vehicle life cycle, from concept, development, production, maintenance, and the end of life.

Safety and security standards. The software must adhere to the highest safety and security standards, including ISO 26262 for functional safety and IEC 21434 for cybersecurity.

Open standards. The solutions should use open standards wherever possible to promote interoperability and portability.

Decoupling software from hardware. The code should be agnostic, rely on open source, and, as a result, abide by the best industry standards and software development standards.

Cybersecurity. With the ever-increasing connectivity of vehicles, cybersecurity is always a concern. The software must be designed with strong security measures to protect against potential cyber threats.

SDV SOFTWARE DEVELOPMENT CHALLENGES

The requirements above, especially safety and security, will remain considerable challenges for automotive software development. In addition, there are several other software challenges facing software-defined vehicles (SDVs) listed below.

- **Software focus.** The transition from hardware-based automotive design to a software-based, software-defined vehicle architecture is a complicated process. The lines of code needed for each automobile are growing exponentially, as is the development effort.
- **Artificial intelligence.** The use of AI in SDVs presents its own set of challenges, including ensuring the reliability and safety of these systems.
- **System integration.** Integrating various software systems in a vehicle is a complex task.
- **Budget constraints.** The budget of vehicle is a significant concern in the transition to SDVs. If SDVs aren't competitively priced and return sufficient margins to manufacturers, the transition will be considered a failure.
- **Skill gap.** Automotive OEMs will need to learn to embrace software in the automotive development process.
- **Supply chain.** Initially, most OEMs will rely on tier 1 and 2 suppliers to provide the parts needed for a vehicle. For the transition to work, these OEMs will need to master the hardware-centric to software-centric transition.

TEST AUTOMATION OF SDVs

To make SDVs a reality, a key component of their development revolves around automation. This includes development environments, model-based development, CI/CD pipelines, and testing. The following sections cover some of the key aspects of test automation and how it can improve the safety and security of SDVs.

SAFETY & SECURITY STANDARDS STILL APPLY

Safety and security will remain paramount concerns for SDVs. As such, vehicle software development will be subject to standards and compliance with these standards. Two important standards:

1. ISO 26262: Road Vehicles – Functional Safety
2. ISO/SAE 21434: Road Vehicles – Cybersecurity engineering

ISO 26262 addresses potential hazards caused by the malfunctioning behavior of electronic and electrical systems in vehicles. The standard provides a set of detailed guidelines and requirements for their functional safety, including an automotive safety lifecycle, functional safety aspects of the entire development process, automotive safety integrity levels (ASILs), and requirements for validation and confirmation measures. These ASILs will apply to software systems in SDVs as they do for current vehicles.

ISO/SAE 21434 builds on ISO 26262 and provides a framework for the entire security life cycle of vehicles. The goal of ISO 21434 is to provide guidance on best development practices from a cybersecurity perspective.

As the automotive industry becomes increasingly software-based and SDVs are by definition software-centric, these standards become even more critical. They ensure that as vehicles become more reliant on software, both the functional safety and cybersecurity aspects are thoroughly addressed.

IMPROVING SDV SAFETY AND SECURITY WITH SOFTWARE TEST AUTOMATION

Static Analysis

Many of the quality tasks specified in ISO 26262, including data and control flow analysis and semantic analysis, are supported by modern advanced tools like Parasoft C/C++test. In addition, static analysis tools include metrics and support peer code review with capabilities that assist unit testing and runtime error detection.

Verification methods like static analysis provide teams a practical way to expose, prevent, and correct errors in automotive software systems. The real power from advanced static analysis tools come from the ability to analyze the code based on industry coding compliance standards like MISRA, CERT, and AUTOSAR C++ 14.

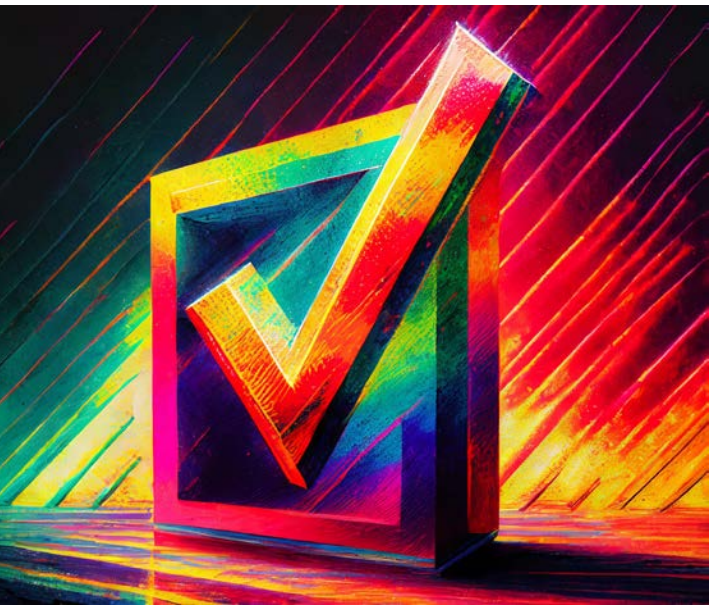
Not only does the analysis report include code rules and directive violations, but also code complexity, quality metrics. This data can be source controlled for historical and auditing purposes. Equally important is the use of a defect tracking and managing system to provide meaningful analytical views and prioritization for the intent of solving the highest risk issues down to the lowest.

Coding Standard Compliance

Coding standards embody the best practices learned from years of experience and aim to harden code by avoiding bad practices that result in inadequate quality and security while promoting good practices that create more resilient code. In the case of automotive standards, they are based on best practices plus guidance on preventing the types of software failures that have been observed over the years.

Coding standards usually define a subset of a programming language deemed safer and more secure to use. The aim of this is to prevent unpredictable behavior in the first place, limiting the risky language features that make them possible.

Here are some examples of coding standards that are likely to be significant for SDV development. These standards are enforceable with static analysis tools like Parasoft C++test.



MISRA C 2023. A set of coding guidelines for the C programming language. The focus of the standard is increasing safety of software by pre-emptively preventing programmers from making coding mistakes that can lead to runtime failures and possible safety concerns by avoiding known problem constructs in the C language.

MISRA C++ 2023. Provides explicit guidance for the avoidance of all instances of undefined and unspecified behavior in support of the C++17 programming language and intended to supersede AUTOSAR C++14.

AUTOSAR C++14. AUTomotive Open System ARchitecture is a worldwide development partnership of OEM manufacturers, tier 1 automotive suppliers, semiconductor manufacturers, software suppliers, tool suppliers, and others that focus on establishing and standardizing automotive software architecture. A key component of AUTOSAR Adaptive Platform is AUTOSAR C++14 coding standard that defines guidelines for the use of the modern C++ language in critical and safety-related systems. This is the only standard on the market that supports modern C++.

SEI CERT C and C++. The Software Engineering Institute (SEI) Computer Emergency Response Team (CERT) has a set of guidelines to help developers create safer, more secure, and more reliable software. Started in 2006 at a meeting of the C Standard Committee, the first CERT C standard was published in 2008 and is constantly developing and evolving.

Rules are guidelines that are detectable by static analysis tools and require strict enforcement, while recommendations are guidelines that have a lower impact and are sometimes difficult to analyze automatically. CERT includes a risk assessment system that combines likelihood of occurrence, severity, and relative difficulty of mitigation. This helps developers prioritize which guideline violations are the most important to investigate. The inclusion of mitigation effort to the guideline priority is an important addition to the CERT secure coding standards, which many other standards lack.

Software Unit Test Automation

ISO 26262 has specific guidelines for testing in accordance with safety integrity levels where requirements-based testing and interface testing are highly recommended for all levels. Fault injection and resource usage tests are recommended at lower integrity levels and highly recommended at ASIL (Automotive Safety Integrity Levels) D. Similarly, the method of driving test cases is also specified with recommended practices.

Test automation provides large benefits to embedded automotive software. Moving away from test suites that require a lot of manual intervention means that testing can be done quicker, easier, and more often. Offloading this manual testing effort frees up time for better test coverage and other safety and quality objectives. An important requirement for automated test suite execution is being able to run these tests on both host and target environments.

Target-based testing for automotive systems. Automating testing for automotive software is more challenging due to the complexity of initiating and observing tests on embedded targets. Not to mention the limited access to target hardware that software teams have. Traceability between test cases, test results, source code, and requirements must be recorded and maintained. So, data collection is critical in test execution.

Parasoft C/C++test is offered with its test harness optimized to take minimal additional overhead for the binary footprint and provides it in the form of source code, where it can be customized if platform-specific modifications are required.

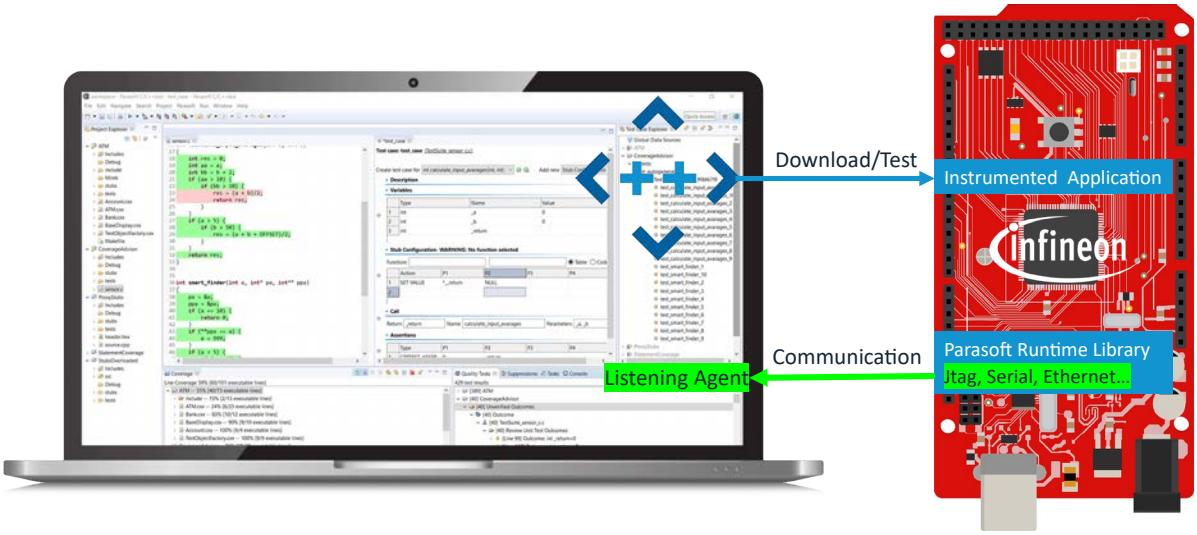


Figure 3: On-target unit testing and/or code coverage with Parasoft C/C++test

IDE integration. Parasoft C/C++test offers dedicated integrations with embedded IDEs and debuggers that make the process of executing test cases smooth and automated. Supported IDE environments include eclipse, VS Code, Green Hills Multi, Wind River Workbench, IAR EW, ARM MDK, ARM DS-5, TI CCS, Visual Studio, and many others.

Automated test case generation. Test data generation and management are by far the biggest challenges in unit testing. Test cases are particularly important in safety-critical software development because they must ensure functional requirements and test for unpredictable behavior, security, and safety requirements. All while satisfying test coverage criteria. Parasoft C/C++test automatically generates test cases in the popular CppUnit format.

Automated regression testing. Parasoft DTP supports the creation of regression testing baselines as an organized collection of tests and will automatically verify all outcomes. These tests run automatically on a regular basis to verify whether code modifications change or break the functionality captured in the regression tests. If any changes are introduced, these test cases will fail to alert the team to the problem. During subsequent tests, DTP will report tasks if it detects changes to the behavior captured in the initial test.

FUNCTIONAL TESTING OF SDVs WITH SERVICE-BASED TESTING

A service-oriented architecture is fundamental to SDVs. Testing vehicles' service interfaces will become a new and critical development activity. The very functionality of the new automobiles will be defined by the services they offer and use. Test automation at the service level is critical if SDVs are to be successful as this approach will support CI/CD, regulatory compliance, in-service updates, and functional validation.

Instead of viewing system quality in terms of meeting individual device requirements, the scope is broadened to consider the quality of the services provided. Testing at the service level helps ensure meeting nonfunctional requirements. For example, performance and reliability are difficult to assess at the unit level or during software unit testing. Service-based testing can simulate the operational environment of a device to provide realistic loads. For example, in an HVAC (heating, ventilation, and air conditioning) system, the temperature sensor can be tested with varying request rates to see if it meets performance requirements.

Security is also a significant concern in automotive systems. Cyber attacks most likely originate from the network itself by attacking the exposed APIs. Service-based testing can create simulated environments for robust security testing, either through fuzzing (random and erroneous data inputs) or denial-of-service attacks. Taking the HVAC example, the sensor might operate correctly with expected requests, but become exposed when overloaded. An attacker might be able to exploit this vulnerability, by gaining access to data or control over parts of the system.

VIRTUALIZED TEST ENVIRONMENTS FOR SDVs

A real test lab requires the closest physical manifestation of the environment in which an automobile is planned to work. Even in the most sophisticated lab, it's difficult to scale to a realistic environment. A virtual lab fixes this problem.

Virtual labs evolve past the need for hard-to-find or nonexistent hardware dependencies. They use sophisticated service virtualization with other key test automation tools.

Service virtualization. Simulates all the dependencies needed by the device under test in order to perform full system testing. This includes all connections and protocols used by the device with realistic responses to communication. For example, service virtualization can simulate an enterprise server backend with which an automobile communicates. Similarly, virtualization can simulate a dependent system, like traffic or weather data, in a realistic manner.

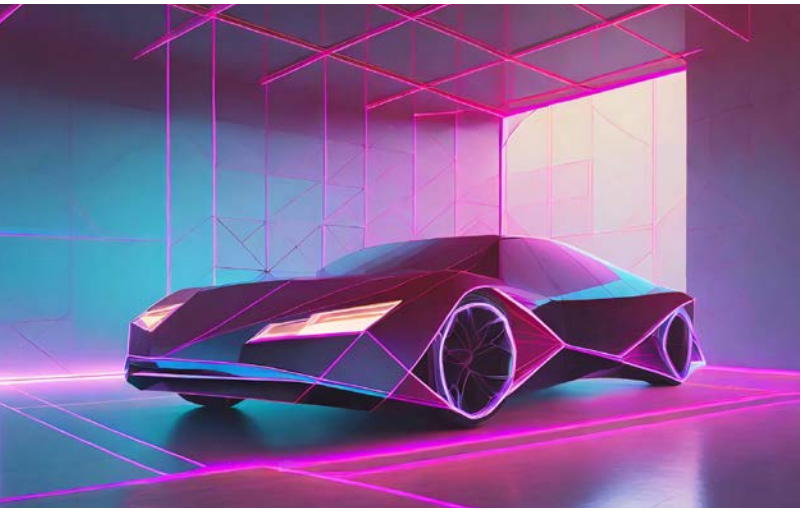
Service and API testing. Offer a way to drive the system under test in a manner that ensures the services and APIs it provides perform flawlessly. These tests can be manipulated via the automation platform to perform performance and security tests as needed.

Runtime monitoring. Detects errors in real time on the system under test and captures important trace information.

Test lab management and analytics. Provide the overarching control of the virtual labs. Once virtualized, the entire lab setup can be replicated as needed and test runs can be automated and repeated. Analytics provide the necessary summary of activities and outcomes.

PARASOFT SOATEST & VIRTUALIZE FOR SERVICE LEVEL TESTING OF SDVs

Developers build integrations earlier, stabilize dependencies, and gain full control of their test data with Parasoft Virtualize. Teams move forward quickly without waiting for access to dependent services that are either incomplete or unavailable. Companies can enable partners to test against their applications with a dedicated sandbox environment.



Parasoft SOAtest delivers fully integrated API and web service testing tools that automate end-to-end functional API testing. Teams streamline automated testing with advanced functional test creation capabilities for applications with multiple interfaces and protocols.

Additionally, as a team's API testing strategy scales, the libraries of test cases grow. When the APIs being tested change, the tests will need to be updated. Ordinarily this causes a significant barrier to scale a test automation strategy, but with SOAtest, teams can manage change in an automated way. Easily configure SOAtest's Change Advisor to automatically scan API interfaces, identify changes in the services, and then create a template that shows how the test assets are impacted by those changes and automatically updates the tests to reflect the changes.

SOAtest and Virtualize are well-suited for network-based system-level testing of various types, including:

- Comprehensive protocol stack that supports, HTTP, MQTT, RabbitMQ, JMS, XML, JSON, REST, SOAP, and more.
- Security and performance testing during integration and system testing with integration into the existing CI/CD process.
- End-to-end testing that combines API, web, mobile, and database interactions into virtual test environments.

SUMMARY

SDVs represent a large transition in the automotive industry where a vehicle's features and functions are primarily enabled through software rather than being hardwired into its hardware components. However, this shift to more reliance on software means an even higher focus on software development for automobile manufacturers, particularly software test automation.

Test automation is a key component in the development of SDVs, particularly for improving testing productivity by orders of magnitude while also ensuring safety and security. One such way is the use of Parasoft C/C++test advanced static analysis to expose, prevent, and correct errors in automotive software systems. This testing solution for C/C++ software development analyzes the code based on industry coding compliance standards like MISRA and CERT, along with reporting code rules, directive violations, code complexity, and quality metrics.

Service-based testing ensures nonfunctional requirements are met and can simulate the operational environment of a device to provide realistic loads. It also allows for robust security testing through methods like fuzzing or denial-of-service attacks.

Virtual labs, which use sophisticated service virtualization and other key test automation tools, offer a solution to the challenges of testing in a realistic environment. Service virtualization simulates all the dependencies needed by the device under test for full system testing.

Parasoft's SOAtest and Virtualize offer solutions for service-level testing of SDVs. Virtualize allows developers to build integrations earlier, stabilize dependencies, and gain full control of their test data. SOAtest provides fully integrated API and web service testing tools that automate end-to-end functional API testing.

TAKE THE NEXT STEP

Request a demo to see how test automation can improve testing productivity exponentially and ensure safety and security in the development of SDVs.

ABOUT PARASOFT

Parasoft helps organizations continuously deliver high-quality software with its AI-powered software testing platform and automated test solutions. Supporting the embedded, enterprise, and IoT markets, Parasoft's proven technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software by integrating everything from deep code analysis and unit testing to web UI and API testing, plus service virtualization and complete code coverage, into the delivery pipeline. Bringing all this together, Parasoft's award-winning reporting and analytics dashboard provides a centralized view of quality, enabling organizations to deliver with confidence and succeed in today's most strategic ecosystems and development initiatives—security, safety-critical, Agile, DevOps, and continuous testing.