HighIntegritySystems

WITTENSTEIN

Issues Facing Automotive Software Developers

Issue 1.8 - October 01, 2026

WITTENSTEIN high integrity systems
Americas: +1 408 625 4712
ROTW: +44 1275 395 600
Email: sales@high**integrity**systems.com
Web: www.high**integrity**systems.com

Contents

Contents....2
List of figures....3
List of notation....3
References....3

CHAPTER 1 Introduction....4

CHAPTER 2 Types of Automotive Software....5
2.1 Control, Infotainment and Usability Software....5
2.2 Advanced Driver Assistance Software (ADAS)....5
2.3 Autonomous Driving Software....5
2.4 Safety-Critical Automotive Software....6

CHAPTER 3 Automotive Software Safety Development Standards....7
3.1 ISO 26262....7
3.2 Safety of the Intended Functionality and AI....8
3.3 Automotive Coding Standards....8

CHAPTER 4 No Safety Without Security....9
4.1 Security standards and regulation....9
4.2 Additional Security Specific Features....10
4.3 Isolation and Freedom from Interference....10
4.4 The Supply Chain and Vulnerability Management....10

CHAPTER 5 Reusable Software Platforms....11
5.1 OSEK....11
5.2 AUTOSAR....12
5.3 Software-Defined Vehicles....12

CHAPTER 6 Software Architecture Considerations....14

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

6.2 Pre-Certified Software Modules.....15

CHAPTER 7 Conclusion.....16

List of Figures

Figure 1 Increasing in Complexity and Risk 6
Figure 2 An Example of an RTOS with an OSEK Layer 11
Figure 3 An Example of ECU Hardware, Basic Software, AUTOSAR and Application Layer.....13
Figure 4 An Example of a Memory Protection Unit 14

List of Notation

ADAS Advanced Driver Assistance Software
API Application Programming Interface
ECU Electronic Control Unit
MCU Microcontroller Unit
MPU Memory Protection Unit
MMU Memory Management Unit
RTOS Real Time Operating System
ASIL Automotive Safety Integrity Level
SEooC Safety Element Out Of Context

References

1. ISO 26262:2018, Road vehicles - Functional safety.
2. ISO 21448:2022, Road vehicles - Safety of the intended functionality.
3. ISO/SAE 21434:2021, Road vehicles - Cybersecurity engineering.
4. ISO 24089:2023, Road vehicles - Software update engineering.
5. ISO/PAS 8800:2024, Road vehicles - Safety and artificial intelligence.
6. SAE J3016:2021, Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor
7. UN Regulation No. 155, Cyber Security and Cyber Security Management System.
8. UN Regulation No. 156, Software Update and Software Updates Management System.
9. Regulation (EU) 2024/2847, Cyber Resilience Act.
10. WITTENSTEIN high integrity systems, Safety Islands: Enabling Safe Software-Defined Vehicles, Issue 1.0, 2026.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

CHAPTER 1 Introduction

Software has become a defining element of the modern vehicle. Functions once implemented by relatively independent electronic control units (ECUs) are increasingly connected through complex software platforms, while advanced driver assistance, automated driving, connectivity and software-defined vehicle architectures continue to increase both software scale and system complexity. The challenges facing engineers developing embedded automotive software therefore extend well beyond functional correctness to include functional safety, cybersecurity, updateability, integration and long-term lifecycle management.

The considerations are many and are discussed in this paper. The type of automotive software remains important because the system in which it operates affects both complexity and risk. As software takes greater responsibility for vehicle functions, functional safety standards must be considered alongside cybersecurity engineering, secure software updates and the evidence required to demonstrate that risks have been controlled. Reuse and integration of existing software platforms remain important, while software-defined vehicle architectures increasingly separate application software from underlying hardware and extend software lifecycles beyond initial vehicle production. Finally, engineers must still ensure that software meets the technical requirements of the application, including deterministic real-time behavior, boot time, responsiveness and resource constraints.

This white paper introduces and discusses these issues facing embedded software engineers developing software for the automotive industry.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

CHAPTER 2 Types of Automotive Software

Automotive software is a broad term; there are many different types of software covering a wide range of functions. One way of classifying software functionality within a vehicle is by how much control the software has over the vehicle.

2.1 Control, Infotainment and Usability Software

Firstly, there is the software that allows the driver to safely control the vehicle, but which can't take direct control over the vehicle. This software is within the command, control and information systems, providing functionality including Engine Management, Powertrain, Braking, External Lighting and Instrument Panels.

There's also the software within the vehicle's comfort, usability and Infotainment systems; managing Climate Control, Windows Control, Central Locking, Interior Lighting, etc. These make the driving experience easier and more enjoyable, but if the software within these systems fails they would only become an inconvenience to the driver.

2.2 Advanced Driver Assistance Software (ADAS)

At the next level is software implementing Advanced Driver Assistance Systems (ADAS). These functions can provide sustained steering and/or acceleration and braking support, but their capabilities and the responsibilities of the driver depend on the specific level and feature. Examples include adaptive cruise control, lane centering, automated parking and collision-avoidance functions. As these functions become more capable, their dependence on sensing, high-performance processing and robust real-time software increases.

2.3 Autonomous Driving Software

Driving automation covers a range of functions rather than a single state of being "autonomous". SAE J3016 defines six levels of driving automation, from Level 0 (no driving automation) to Level 5 (full driving automation). Levels 1 and 2 are driver-support features, while Levels 3 to 5 are automated-driving features with progressively broader responsibility for the dynamic driving task. This distinction is important to software developers because system responsibilities, fallback behavior, monitoring and validation differ significantly between the levels.

Automated-driving functions depend on information from multiple sensors, which may include cameras, radar, lidar, ultrasonic sensors and vehicle-state data. Sensor fusion combines these inputs to build a sufficiently accurate representation of the vehicle and its surroundings. The software must process this data within bounded timing constraints, identify inconsistent or degraded sensor information, and make control decisions while preserving the behavior expected by the safety concept.

Artificial intelligence and machine learning can introduce additional assurance challenges where behavior is learned from data rather than expressed entirely through conventional deterministic logic. ISO/PAS 8800:2024 addresses safety-related road-vehicle systems that use AI technology and considers risks arising from output insufficiencies, systematic errors and random hardware errors. In parallel, ISO 21448:2022, Safety of the Intended Functionality (SOTIF), addresses hazards caused by functional or performance insufficiencies where no malfunction covered by ISO 26262 is present. These considerations are particularly relevant to perception and other functions where correct situational awareness is essential.

At higher levels of automation, the system must also address degraded operation and failures without assuming that a human can always intervene immediately. This increases the importance of fail-operational or controlled-degradation architectures, independent safety monitoring and clearly defined fallback strategies. Real-time performance is therefore not merely a throughput target: timely detection, decision-making and actuation form part of the safety argument for the complete function.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

2.4 Safety-Critical Automotive Software

As established, within the vehicle there are many systems that rely on software. The software becomes more complex and processing intensive but must still remain very responsive. The greater the control the software has over the vehicle, the higher the risk, and the more safety critical it becomes.

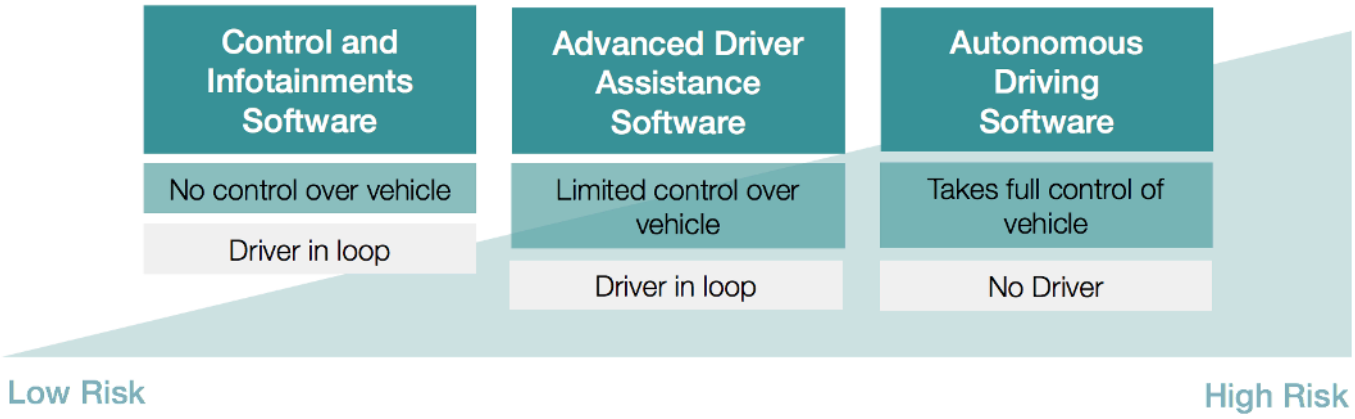


Figure 1. *Increasing in Complexity and Risk*

As a result, to keep drivers, pedestrians and the built environment safe, development and integration of automotive safety-critical software is being driven forward by international standards and regulations.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

CHAPTER 3 Automotive Software Safety Development Standards

3.1 ISO 26262

The ISO 26262 standard is an adaptation of the Functional Safety standard IEC 61508 for Automotive Electric/ Electronic Systems. ISO 26262 defines Functional Safety for automotive equipment, applicable throughout the lifecycle of all automotive electronic and electrical safety-related systems.

Part 6 of ISO 26262 specifies the requirements for the development of software for automotive applications. These are the requirements for initiation of:

- Product development at the software level;
- Specification of the software safety requirements;
- Software architectural design;
- Software unit design and implementation;
- Software unit testing;
- Software integration and testing;
- Verification of software safety requirements.

ASIL A is the lowest safety rating under this standard. ASIL D is the highest, and is achieved by performing a risk analysis of a potential hazard that examines the severity, exposure and controllability of the vehicle operating scenario.

If a generic software component such as an RTOS is under development, the final application where software will be used may not be known. Hence the software will have to be certified as a “Safety Element out of Context” (SEooC). When designing such software, assumptions will be made about its safety goals and ASIL level required. These safety goals must be described within the Safety Manual along with the installation and integration instructions. Developers using the software will need to confirm that the safety goals defined during the software’s development meet the requirements of their projects.

3.1.1 Self-monitoring Software

An interesting ISO 26262 requirement is the expectation that software runtime verification monitors will be used to detect, indicate and handle systematic faults within software rated ASIL C and D. This means that despite the fact that all the software has been designed and verified following a robust and rigorous safety critical development life cycle, ISO 26262 still requires self-verification of the software.

This can be illustrated with an example of an automotive software component - a Real Time Operating System (RTOS). As standard the RTOS may include a range of built-in error checking routines, however to fully satisfy this requirement additional monitoring is required. Some RTOS vendors can provide a plugin module that provides a Task Monitoring capability, ensuring the scheduling of Tasks is occurring as intended, and within the expected time frame, hence satisfying the requirement.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

3.2 Safety of the Intended Functionality and AI

ISO 26262 addresses hazards caused by malfunctioning behavior of safety-related E/E systems, but modern driver-assistance and automated-driving functions also create safety questions when a system is functioning as designed, yet its specification, sensing or implementation is insufficient for a particular situation. ISO 21448:2022 provides a framework for Safety of the Intended Functionality (SOTIF), including design, verification, validation and operational activities for functions in which situational awareness is important.

For systems incorporating artificial intelligence, ISO/PAS 8800:2024 extends the automotive safety landscape by addressing safety-related E/E systems that use AI technology. Together with ISO 26262 and SOTIF, this reinforces a broader view of assurance in which safety arguments may need to address conventional faults, functional insufficiencies and AI-specific output limitations. The standards are complementary rather than interchangeable, and their applicability depends on the system and functions being developed.

3.3 Automotive Coding Standards

MISRA C is a set of software development guidelines for the C programming language initially developed by MISRA (Motor Industry Software Reliability Association) but now widely used across many industries.

Its aims are to facilitate code safety, security, portability and reliability in the context of embedded systems, specifically those systems programmed in C, by providing a restricted subset of a standardized C programming language that is deemed to be safer.

There are many design tools and static code checkers that can enforce MISRA rules within the software. However, having good coding style guidelines is only one small part of the overall solution, it does not replace the need for a formal design system.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

CHAPTER 4 No Safety Without Security

On a modern connected car there can be no Safety without Security. Car Hacking is an emotive topic, and can include remote access using the cars Cellular Network Connection[2], close to the vehicle via the Tire Pressure Monitoring System[3], or connected to the vehicle via the on-board diagnostics software[4].

Having software that is both safe and secure is an interesting mix. Safety software is designed for use in a structured environment, where the risks are identified, analyzed and mitigations are put in place to reduce the risks to an acceptable level. Safety software takes a long time to develop and verify, is robust and reliable, and consequently is rarely updated. Whereas the security threat is constantly evolving, with attacks growing in complexity as hackers learn and develop knowledge on how to exploit the software. This results in software being regularly updated to counteract hacker attacks.

4.1 Security Standards and Regulation

A modern automotive cybersecurity program must address risk throughout the vehicle lifecycle rather than treating security as a collection of individual technical features. ISO/SAE 21434:2021 defines engineering requirements for cybersecurity risk management across the lifecycle of road-vehicle E/E systems, including concept, development, production, operation, maintenance and decommissioning. It also provides a common framework for cybersecurity activities across OEM and supplier boundaries.

Threat Analysis and Risk Assessment (TARA) is central to this risk-based approach. Assets, damage scenarios, threat scenarios and attack paths are analyzed so that cybersecurity risks can be evaluated and appropriate cybersecurity goals and controls established. Because threats and vulnerabilities evolve after a vehicle enters service, cybersecurity management must continue in the field through monitoring, vulnerability management, incident response and remediation.

Regulation has also become a direct part of the automotive cybersecurity landscape. UN Regulation No. 155 addresses vehicle cybersecurity and Cyber Security Management Systems (CSMS), while UN Regulation No. 156 addresses software updates and Software Update Management Systems (SUMS). Software updates therefore require controlled processes as well as a secure delivery mechanism. ISO 24089:2023 complements this area with requirements and recommendations for software update engineering at organizational and project levels and for vehicles, ECUs, infrastructure and update packages.

The EU Cyber Resilience Act (CRA), Regulation (EU) 2024/2847, establishes horizontal cybersecurity requirements for products with digital elements, but its relationship with automotive products must be considered carefully. Products already covered by specified EU vehicle type-approval legislation are excluded from the CRA where the sectoral framework provides the relevant protection, and further exclusions have been made to avoid overlapping requirements for certain vehicle categories. The CRA is nevertheless relevant to the wider digital-product and supply-chain environment surrounding vehicle software, tools and connected services, subject to the scope of the applicable legislation.

These developments make cybersecurity increasingly an evidence challenge as well as an engineering challenge. Organizations must be able to demonstrate that risks have been analyzed, requirements have been allocated, implementation and verification activities have been performed, vulnerabilities are monitored, and changes or updates remain controlled across long product lifecycles and distributed supply chains.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

4.2 Additional Security Specific Features

Technical cybersecurity controls are selected from the system threat and risk analysis rather than applied as a fixed checklist. Typical mechanisms include secure or verified boot, authentication, cryptographic protection of data and communications, secure key storage and management, protected diagnostics, least-privilege access control and mechanisms that support secure software update and rollback strategies.

Hardware security features can provide acceleration, protected key storage and stronger isolation, while software implements policy, protocols, monitoring and application-specific controls. The effectiveness of these mechanisms depends on the complete architecture and on correct integration throughout the product lifecycle.

4.3 Isolation and Freedom from Interference

Isolation remains important as mixed-criticality and connected workloads are consolidated onto more capable processors. Hardware mechanisms such as Memory Protection Units (MPUs), Memory Management Units (MMUs), privilege levels and processor-supported security or virtualization features can help separate software domains and restrict access to memory, peripherals and communications. Unused interfaces and unnecessary services should be disabled or constrained in accordance with the system security design.

Isolation is also significant at the boundary between functional safety and cybersecurity. A security compromise in a non-safety application must not silently undermine a safety-critical function, while safety monitors and recovery mechanisms need sufficient independence to remain effective when other software is faulty or compromised.

4.4 The Supply Chain and Vulnerability Management

Automotive software is assembled from software developed by OEMs, Tier-1 and lower-tier suppliers, commercial products, open-source components and development tools. The security objective is therefore not simply to minimize the number of suppliers; it is to understand provenance, interfaces, dependencies and responsibilities well enough to manage risk throughout the lifecycle.

Vulnerability management must continue after release. Organizations need processes to identify and assess newly disclosed vulnerabilities, determine which vehicle configurations are affected, define and verify remediation, and deploy updates in a controlled manner. This places a premium on accurate configuration records, traceability and clear cybersecurity agreements across the supply chain.

Automotive software is highly complex. Standards and platform architectures such as OSEK and AUTOSAR have long supported reuse and integration, while the move towards software-defined vehicles is extending the platform concept across increasingly centralized vehicle compute architectures.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

CHAPTER 5 Reusable Software Platforms

5.1 OSEK

A consortium of German automobile manufacturers began the standard Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen (OSEK). It is an open standard, designed to provide a standard software architecture for the various Electronic Control Units (ECUs) in a vehicle. The OSEK consortium was later joined by a similar group of French automobile manufactures VDX, and is now officially designated as OSEK/VDX.

The OSEK/VDX standards cover various different aspects of the vehicle's software architectures, including:

OSEK OS	Operating system
OSEK Time	Time triggered operating system
OSEK COM	Communication services
OSEK FTCOM	Fault tolerant communication
OSEK NM	Network management
OSEK OIL	Kernel configuration
OSEK ORTI	Kernel awareness for debuggers

Some elements of OSEK are scalable, for example OSEK OS defines a set of four conformance classes (BCC1, BCC2, ECC1 and ECC2). These classes are designed to cover different capabilities of the hardware and software, reflecting the performance requirements of different applications and features, allowing partial implementations to be defined as compliant where full implementations are not needed.

Figure 2 shows an example OSEK architecture involving an RTOS. Some RTOS can be supplied with an optional OSEK OS adaptation layer, allowing them to be used as a drop-in component within OSEK OS compliant systems.

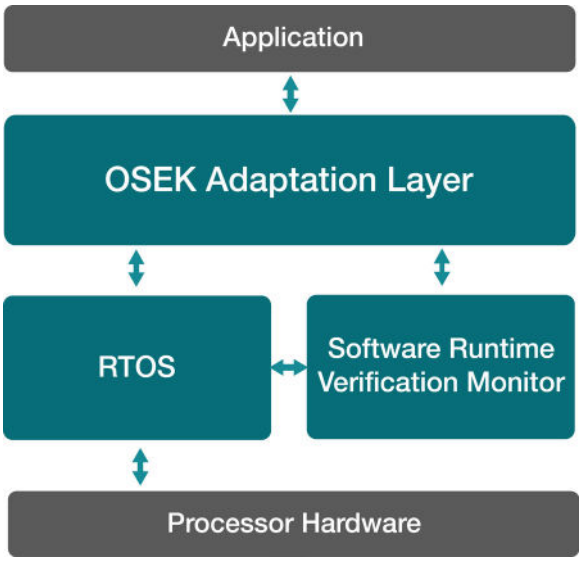


Figure 2. *An Example of an RTOS with an OSEK Layer*

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

5.2 AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) is a worldwide development partnership of automotive interested parties. Its objective is to create and establish an open and standardized software architecture for automotive electronic control units (ECUs) excluding infotainment. AUTOSAR provides a standard software development base for industry collaboration on basic functions while providing a platform which continues to encourage competition on innovative functions and promoting reuse of software components.

AUTOSAR uses a three-layered architecture:

- Basic Software: standardized software modules that provide a Hardware Abstraction Layer.
- Runtime environment: Middleware which abstracts from the network topology for the inter- and intra-ECU information exchange between the application software components and between the Basic software and the applications.
- Application Layer: Provides a common software API for automotive applications in terms of syntax and semantics.

The AUTOSAR specification for operating systems according to scalability class 1 is based on the OSEK/VDX standard. Figure 3 shows an example of this structure, illustrated with an RTOS. Many modern RTOS can be supplied with an OSEK OS API wrapper, allowing them to be integrated within an AUTOSAR environment.

5.3 Software-Defined Vehicles

Software-defined vehicles (SDVs) extend the platform approach by making software a primary means of defining, differentiating and evolving vehicle functionality. Instead of mapping every function permanently to a dedicated ECU, newer E/E architectures increasingly consolidate processing into domain, zonal and central compute platforms. Zonal controllers aggregate local sensors and actuators, while central compute platforms provide the processing capacity required for cross-vehicle applications and increasingly integrated software services.

This architectural shift supports a clearer separation between software functionality and the underlying hardware platform. Hardware and software can therefore evolve on different timescales, provided that interfaces, resource assumptions and safety and security constraints remain controlled. The potential benefits include greater software reuse, fewer duplicated computing resources and the ability to introduce or improve features through controlled software deployment rather than hardware replacement.

Over-the-air (OTA) updates are a key enabler of this lifecycle model. They allow defects and vulnerabilities to be remediated and vehicle functionality to evolve after production, but they also extend the engineering responsibility into vehicle operation. Update authenticity and integrity, compatibility, safe installation, recovery from failed updates, configuration management and the effect of an update on safety and type-approved characteristics all need to be considered. UN R156 and ISO 24089 provide important frameworks for these activities.

Consolidation also changes the safety and cybersecurity problem. A central compute platform may host applications with different criticality levels and different update cadences. Strong isolation and freedom from interference are therefore required so that faults or compromises in one domain do not propagate into another. Verification and validation strategies must also account for platform dependencies and for the possibility that frequently changing application software interacts with safety-related functions.

One architectural response is the use of a safety island: an independent or strongly isolated execution environment dedicated to safety supervision, fault handling and safe-state management. Separating this functionality from the main application domain can reduce dependencies between fast-moving application software and the safety-related software that requires stronger assurance. The safety island is not a substitute for a complete ISO 26262 safety process, but it can provide a useful architectural boundary for partitioning safety responsibilities and limiting the impact of change.

As described in the WITTENSTEIN high integrity systems white paper “Safety Islands: Enabling Safe Software-Defined Vehicles”, such an architecture can support independent monitoring, deterministic real-time behavior and bounded interaction with the main application domain. This approach is particularly relevant where high-performance central compute runs complex or frequently updated software while a smaller safety-oriented environment retains responsibility for monitoring and recovery.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

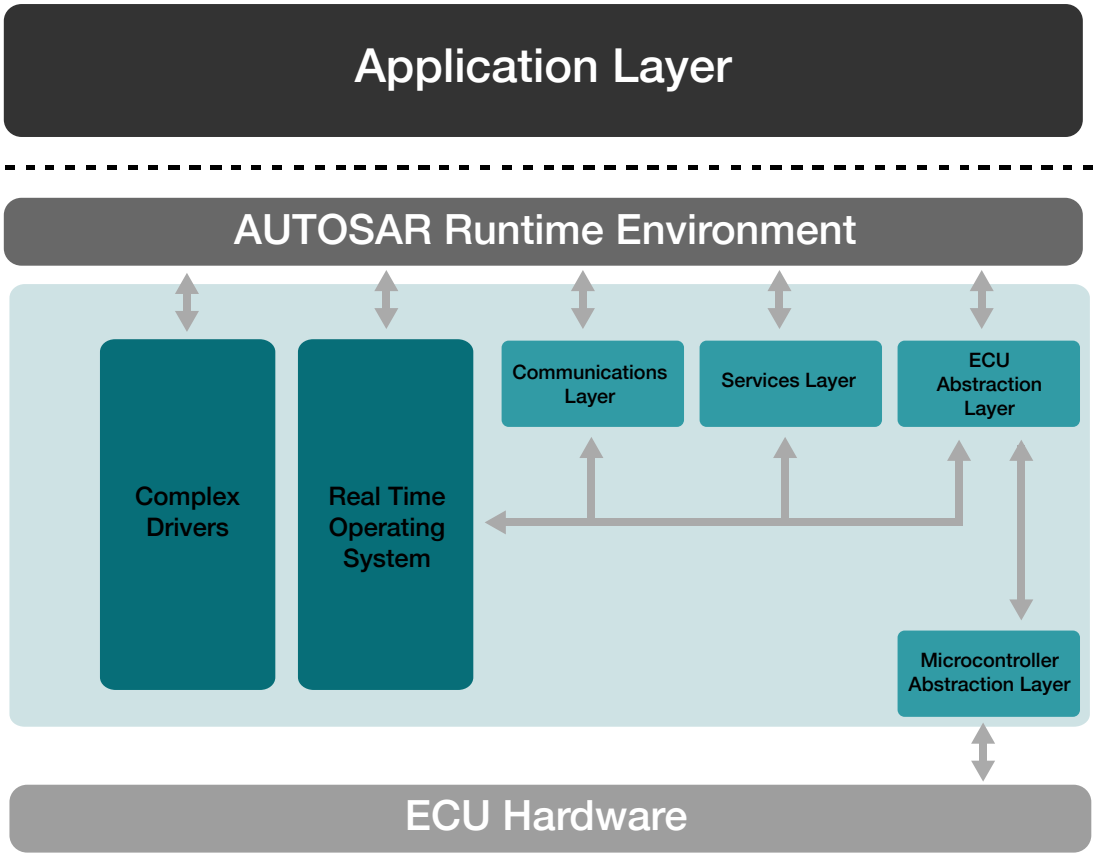


Figure 3. *An Example of ECU Hardware, Basic Software, AUTOSAR and Application Layer*

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

CHAPTER 6 Software Architecture Considerations

6.1 General Considerations

There are many considerations for the software within the Automotive sphere, but the traditional care taken over software architecture must not be neglected.

For most systems a quick software boot time is essential, placing the car in a working and safe state as quickly as possible. Responsiveness is also very important, as events happen very fast when a car is travelling at speed. To manage this, most automotive architectures support parallel processing. Therefore, the software will need to support core to core communications and synchronization.

Due to cost, space or procurement constraints, automotive software within a single processor may have to provide functionality supporting different Safety Integrity Levels (SIL). To do this the developer needs to ensure software designed to support lower Safety Integrity Levels cannot interfere with software designed to support higher Safety Integrity Levels.

This problem is normally solved by using the processors Memory Protection Unit (MPU) or Memory Management Unit (MMU), whereby specific memory regions can be allocated to software sections that have been designed to the same SIL level.

This can be achieved by selecting an RTOS that supports Task Separation and Isolation by using the processors underlying MPU/MMU infrastructure. Here each Task is allocated its own memory regions. Each time there is a context switch and a new Task is activated, the MPU/MMU registers are re-programmed with the memory regions relating to the new Task. If the software tries to access a memory region outside of its permitted range an exception will be triggered.

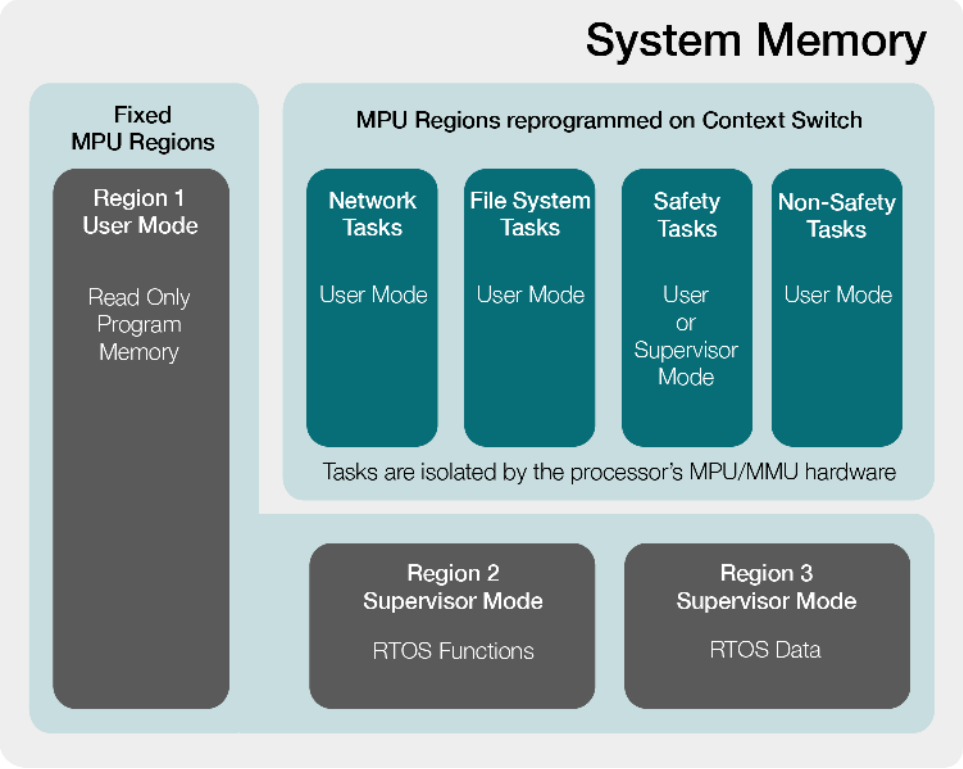


Figure 4. *An Example of a Memory Protection Unit*

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

6.2 Pre Certified Software Modules

Developing automotive software is complex and time consuming, but solutions are available. The market trend is to construct automotive software from pre-existing modules. Many of these modules have been pre-certified against ISO 26262.

Pre-certified software modules provide robust and reliable software. There is a variety on the market, but be wary of the terms “certifiable” and “certified”, as there is a large difference between the two. It’s recommended to select pre-certified software that has been designed and verified upon your specific processor and compiler combination, if possible, even down to your version of the compiler and your compiler setting, as it removes the need for retesting on the target hardware.

If you are considering using pre-certified software, you should also consider using a certified compiler.

6.2.1 SAFE**RTOS**®: An example of a pre-certified software module

A good example of a pre-certified software module is SAFE**RTOS**®.

SAFE**RTOS**® is a safety certified Real Time Operating System (RTOS) for embedded processors, developed by WITTENSTEIN high integrity systems (WHIS). It has been designed for the highest standards of functional safety and certified by TÜV SÜD to IEC 61508 SIL 3 and to ISO 26262 ASIL D. SAFE**RTOS**® also contains features that support the development of safety critical automotive software.

- Available pre-certified to ISO 26262 ASIL D by TÜV SÜD;
- Supports a wide range of automotive processors;
- Quick boot time, highly responsive;
- Task separation and isolation feature;
- OSEK OS adaptation layer available;
- Widely used across the automotive industry.

To learn more about SAFE**RTOS**® please see https://www.highintegritysystems.com/safertos or contact WHIS.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

CHAPTER 7 Conclusion

From the above discussion, it's easy to understand that the challenges facing embedded engineers developing software for automotive applications are many. They include issues such as safety, security, integration and managing software containing a mix of different Safety levels.

Cybersecurity is now a lifecycle engineering and regulatory discipline. ISO/SAE 21434, UN R155 and UN R156 place greater emphasis on systematic risk management, controlled software updates and evidence that cybersecurity activities are sustained after production. Functional safety and cybersecurity remain distinct disciplines, but increasingly connected vehicle architectures require both to be considered together.

OSEK and AUTOSAR established important foundations for integration and reusable software components. Software-defined vehicles extend this evolution towards centralized and zonal compute platforms, stronger hardware/software separation and continuous software deployment. These architectures can improve reuse and scalability, but they also increase the importance of isolation, deterministic execution, configuration control and carefully bounded safety responsibilities.

Finally, the functionality itself must continue to meet demanding real-time and resource requirements. Pre-certified software components can reduce duplicated assurance effort when used within their documented assumptions and integration constraints, while architectures such as safety islands can help limit the scope affected by rapidly changing application software. The objective remains the same: to enable innovation without weakening the evidence needed to justify safety and security.

Automotive Software is an exciting field to be a part of, especially at this time of exponential growth. WITTENSTEIN high integrity systems is glad to be one of the forerunners providing help to engineers in this vital industry.

SAFE**RTOS**® was put forward as an example of a pre-certified software module widely used across the automotive sector. To learn more about SAFE**RTOS**® please see https://www.highintegritysystems.com/safertos or contact WHIS.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.

Contact Information

Your Feedback

We would be pleased to receive your comments and any suggestions for improvement to our software and documentation.

Please email us at: support@highintegritysystems.com

Contact WITTENSTEIN high integrity systems

Address: WITTENSTEIN high integrity systems
3 Rivergate, 1st Floor
Temple Quay
Bristol, BS1 6EW

Americas +1 408 625 4712
ROTW +44 1275 395600

Email: support@HighIntegritySystems.com

Website www.HighIntegritySystems.com

All Trademarks acknowledged.

Copyright 2026 WITTENSTEIN high integrity systems Ltd. V1.8. Correct at time of issue.