HighIntegritySystems

WITTENSTEIN

# Issues Facing Automotive Software Developers

Issue 1.6 - October 14, 2020

**WITTENSTEIN high integrity systems**

| | |
|---|---|
| Americas: | +1 408 625 4712 |
| ROTW: | +44 1275 395 600 |
| Email: | sales@highintegritysystems.com |
| Web: | www.highintegritysystems.com |

HighIntegritySystems

WITTENSTEIN

# Contents

## List of Figures

## 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 |

# CHAPTER 1 Introduction

## 1.1 Introduction

There has been an amazing growth of software used within automobiles in recent years, with cars quickly becoming super computers on wheels. There are now significantly more lines of code within a premium branded car than on a jet aircraft. The challenges facing engineers developing embedded software for automobiles are great, and cover a very broad range of issues.

First to be touched on in this paper include the type of automotive software in question – important, as the system in which it will function affects not only the complexity but also the risk . We will see how the greater the control the software has over the vehicle, the more safety critical it becomes, hence Automotive Safety Development standards must be considered.

Closely tied to the issue of safety is security, as shown by some real life examples. There are a variety of considerations for security, including high level concerns such as supply chains, right down through to standards and security features within the code.

Finally, there is the re-usability of existing software modules to be taken into account, as well as the traditional challenge of ensuring the software has the technical ability to meet the requirements of the application.

This white paper introduces and discusses these issues that face embedded software engineers who are developing software for the automotive industry.

# 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 we have the software implementing Advance Driver Assistance Systems (ADAS). These systems can take limited control over a vehicle, however, the driver is still within the control loop of the vehicle. ADAS systems include Adaptive Cruise Control, Automatic Parking, Blind Spot Detector and Collision Avoidance Systems. It is an area of rapid development.

## 2.3 Autonomous Driving Software

Finally, there is the software that will implement the autonomous driving algorithms. Here the car uses a very broad array of sensors to make sense of its environment, and takes full control of the vehicle. In the UK fully autonomous driving will be phased in following a general strategy of "Hands Off, Eyes Off, Mind Off".

The result of all this software development is that driver and pedestrian protection has moved from impact survival, to active avoidance. Håkan Samuelsson, President and CEO of Volvo Cars, even states that their vision is "that by 2020 no one should be killed or seriously injured in a new Volvo car"[1].

## 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.

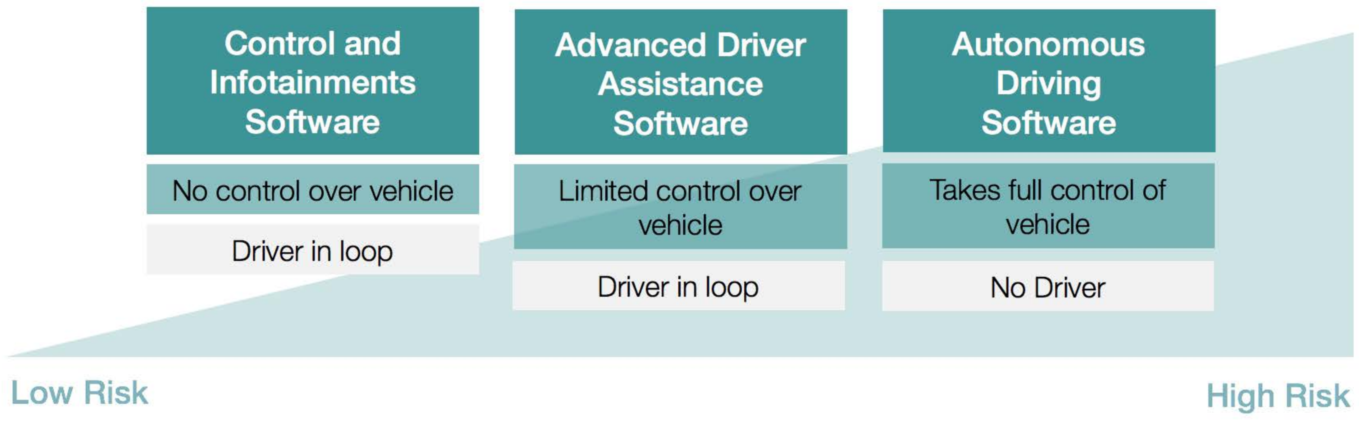
Control and Infotainments Software
No control over vehicle
Driver in loop
Advanced Driver Assistance Software
Limited control over vehicle
Driver in loop
Autonomous Driving Software
Takes full control of vehicle
No Driver
Low Risk
High Risk

Figure 2-1 *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.

# 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.

## 3.2 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 enforcing the implementation of good coding practises resulting in safer and more predictable code behaviour.

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.

# 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 Tyre 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, analysed 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

For safety, ISO 26262 Part 6 standardises software development, however for security there is no such dominant, overreaching design regulation. Just some of the many different security guidelines and standards are below.

### 4.1.1 Secure Hardware Extension (SHE)

Secure Hardware Extension (SHE) is a security standard which defines the operation of a security module. It allows a secure zone to be created in any ECU (electronic control unit), as well as effectively storing and managing security keys, among other features.

### 4.1.2 EVITA and PRESERVE

EVITA and PRESERVE are guidelines resulting from the European funded projects. EVITA specifies the design, verification and prototyping of software architectures, as well as a range of functions and parameters for the management of security and encryption keys.

PRESERVE has sprung from EVITA, and has focuses including the secure transmission of data and control information for the future Intelligent Transportation System (ITS), and the use of public key cryptography in the hardware security module.

### 4.1.3 Cert C

Cert C is a software coding standard from the Software Engineering Institute (SEI), which provides coding guidelines for those developing software that requires a degree of security. Cert C provides rules and recommendations which, when followed, result in less vulnerabilities for the hacker to exploit.

## 4.2 Additional Security Specific Features

When developing for automotive security, software developers need to consider closely the features to use.

For example, is the system using Verified Boot, and if so, does the device contain the correct software at boot time?  Or features such as authentication- is it the correct device? Then there are Encryption and Decryption algorithms to keep the data secret, and public and private key management. In general, security is normally implemented by a combination of Software, with Hardware providing acceleration and isolation.

## 4.3 Physical isolation

Physical isolation is very important. Access should be restricted to the intended devices only, and the software should disable all unused ports and unwanted points of access. The security software should also be isolated from other software. How this is achieved depends on your selected hardware. For example, ARM provide Trust Zone, and Synopsys ARC provide SecureShield. Good isolation can also be achieved using the processors MPU/MMU correctly.

## 4.4 The Supply Chain

For those involved in purchasing software used within a security system, the length of the supply chain should be an important consideration. Every line of code in a software component needs to be verified and accounted for, which becomes simpler the shorter the chain. A long supply chain may be a result of companies including third party software within their project, which could be many levels deep.

Typically, good security results from a short supply chain, ideally from a single trusted and respectable company.

# CHAPTER 5 Reusable Software Platforms

As is becoming apparent, Automotive software is highly complex. Therefore, to simplify integration standards OSEK and AUTOSAR have come into play, standardising software architecture and increasing the use of compatible software components.

## 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 manufacturers with their VDX standard, 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 5-1 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.

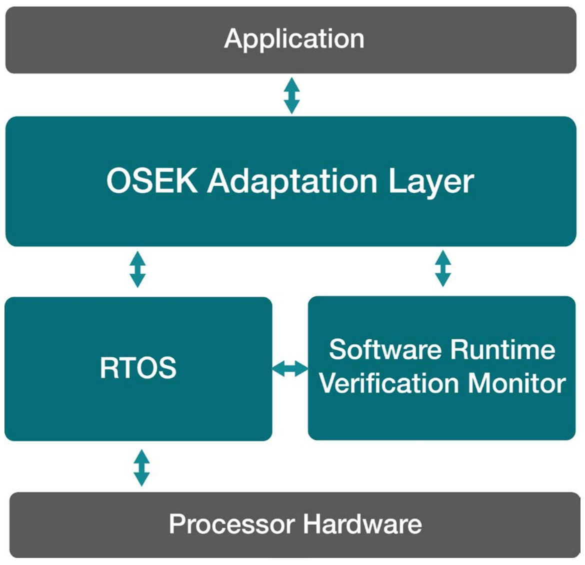

Figure 5-1 *An Example of an RTOS with an OSEK Layer*

## 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 standardised 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: standardised 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 5-2 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.

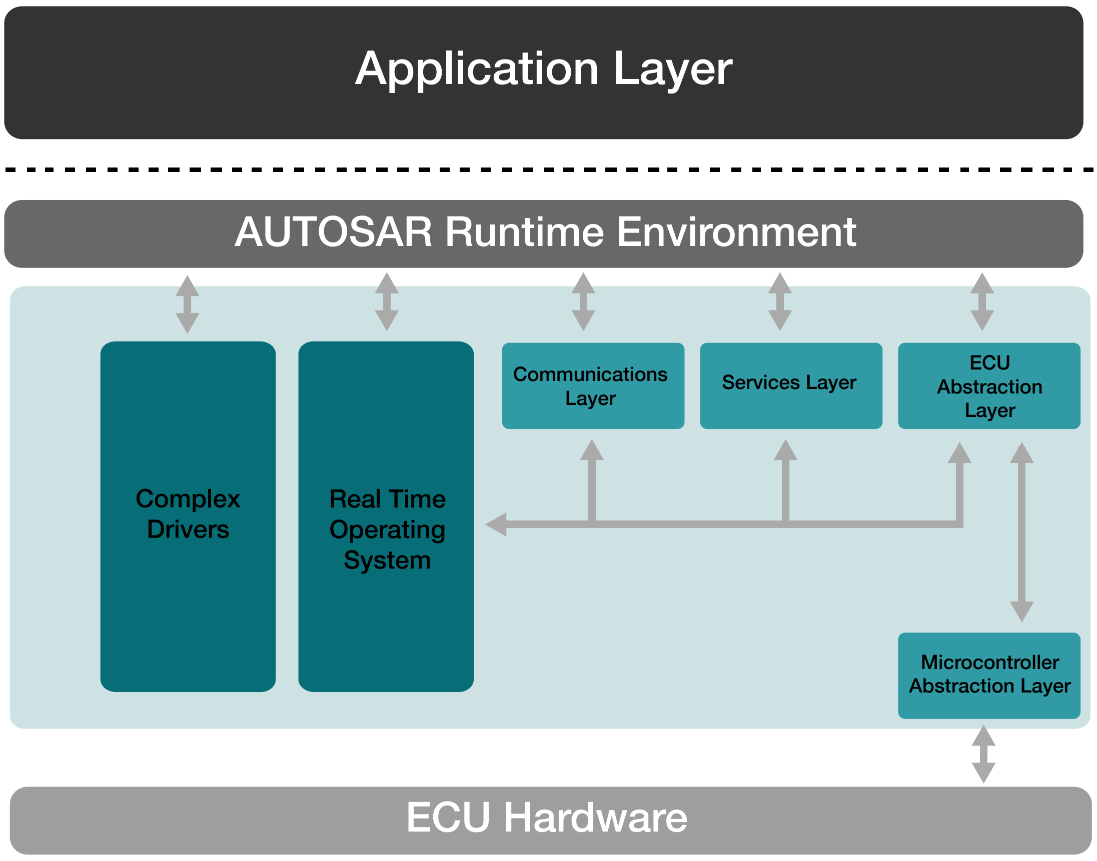

Figure 5-2 *An Example of ECU Hardware, Basic Software, AUTOSAR and Application Layer*

# 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 into 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 synchronisation.

Due to cost, space or procurement constraints, automotive software within a single processor may have to provide functionality supporting different Safety Integrity Levels. 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.

Part of the solution is to use the processor's 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.

A degree of spatial separation can be achieved by selecting an RTOS that provides the tools enabling the definition and manipulation of MPU regions on a per task basis. 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.

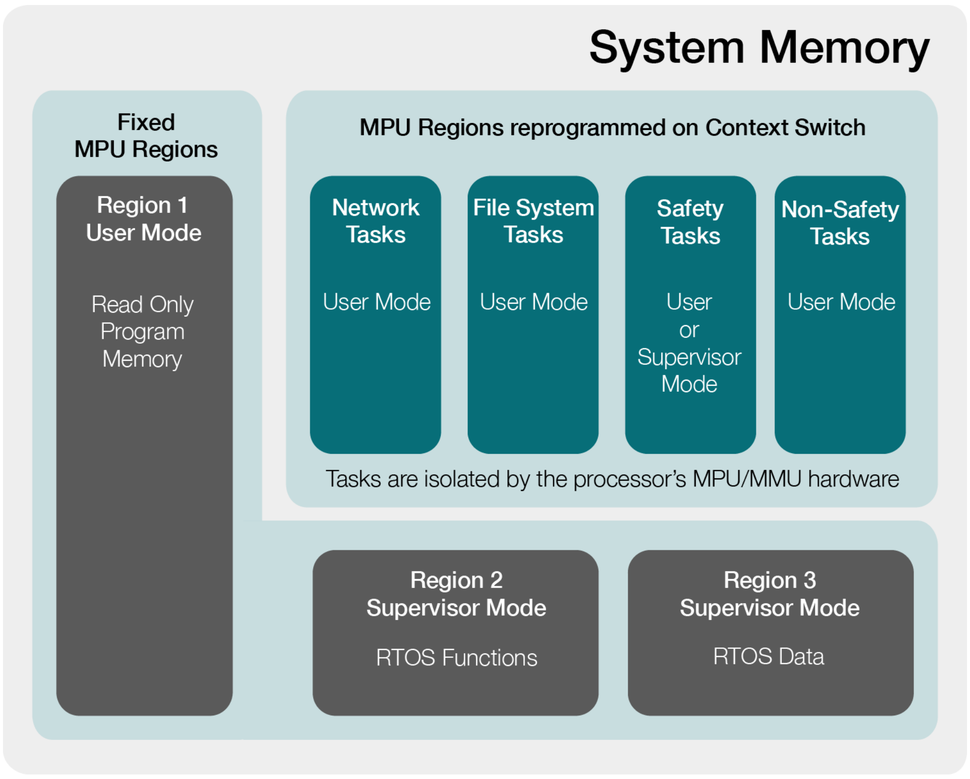

Figure 6-1 *An Example of a Memory Protection Unit*

### 6.1.1 “Should we Use an RTOS?”

There are well-established techniques for writing good embedded software without the use of an RTOS. In some cases, these techniques may provide the most appropriate solution; however as the solution becomes more complex, the benefits of an RTOS become more apparent. These include:

**Priority Based Scheduling:** The ability to separate critical processing from non-critical is a powerful tool.

**Abstracting Timing Information:** The RTOS is responsible for timing and provides API functions. This allows for cleaner (and smaller) application code.

**Maintainability/Extensibility:** Abstracting timing dependencies and task based design results in fewer interdependencies between modules. This makes for easier maintenance.

**Modularity**: The task based API naturally encourages modular development as a task will typically have a clearly defined role.

**Promotes Team Development**: The task based system allows separate designers/teams to work independently on their parts of the project.

**Easier Testing:** Modular task based development allows for modular task based testing.

**Code Reuse:** Another benefit of modularity is that similar applications on similar platforms will inevitably lead to the development of a library of standard tasks.

**Improved Efficiency:** An RTOS can be entirely event driven; no processing time is wasted polling for events that have not occurred.

**Idle Processing:** Background or idle processing is performed in the idle task. This ensures that things such as CPU load measurement, background CRC checking etc will not affect the main processing.

## 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 on 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 re-testing on the target hardware.

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

### 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.

# **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.

It has been seen that Security is a rapidly increasing field with a variety of standards and regulation. While there is room for improvement and standardisation, it's clear that good practices such as code structure, architecture, and supply chain are still as important as ever.

We've seen how OSEK and AUTOSAR encourage integration and re-usable components. Now an established part of the Automotive software landscape, they increase engineering efficiency across the globe.

Finally, we've looked at the functionality of the software itself, and how, as in all industries, care must be taken to optimise for the application. We've seen how the trend to using pre-certified software modules and compilers is making automotive software development easier by allowing the developer to quickly add functionality to their projects with minimum risk.

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 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.

## References

1. Vision 2020 by Håkan Samuelsson, Volvo. http://www.volvocars.com/intl/about/vision-2020

2. THE JEEP HACKERS ARE BACK TO PROVE CAR HACKING CAN GET MUCH WORSE by Andy GreenBerg. 08/01/16. https://www.wired.com/2016/08/jeep-hackers-return-high-speed-steering-acceleration-hacks/

3. Wireless Car Sensors Vulnerable to Hackers by Robert Lemos. 10/08/2010. https://www.technologyreview.com/s/420168/wireless-car-sensors-vulnerable-to-hackers/

4. On Board Diagnostics: Risks and Vulnerabilities of the Connected Vehicle by Dan J. Klinedinst & Christopher King. 04/16. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=453871

# Contact Information

## Contact WITTENSTEIN high integrity systems

| | |
|---|---|
| Address: | WITTENSTEIN high integrity systems<br>Brown’s Court, Long Ashton Business Park<br>Yanley Lane, Long Ashton<br>Bristol, BS41 9LB<br>England |
| Phone: | +44 (0)1275 395 600 |
| Fax: | +44 (0)1275 393 630 |
| Email: | support@HighIntegritySystems.com |
| Website | www.HighIntegritySystems.com |

All Trademarks acknowledged.