CANONICAL

A CTO's Guide to software-defined vehicles

Recommendations for a maintainable future

June 2022

Executive Summary

From entertainment and driver assistance to autonomous driving, software provides huge benefits when integrated into vehicles, both for the industry and for customers. But like mechanical parts that need to be cleaned, greased up and sometimes even replaced, software also needs maintenance.

OEMs and Tier1 suppliers were familiar with choosing and positioning vehicle elements in a way that facilitates regular vehicle maintenance for things like an oil change for the engine, or distribution chain replacement. However, this can be trickier when it comes to software. With increasing complexity, architectural choices made today will drive software maintenance costs up for several years into the future. This requires OEMs and Tier1 suppliers to work on the system and software architecture of the vehicle keeping in mind that they will have to provide regular maintenance later on (like software security vulnerability patching or software upgrades).

In this paper, we will illustrate how the automotive industry can ensure state-of-the-art safety and security solutions for software-defined vehicles. We will also address how the use of open source software can enable OEMs to focus on next-generation application development and meeting consumer needs while keeping costs under control.

Introduction:
software-defined, an industry U-turn

Over the past years, rapid improvements in vehicle electrification and autonomous driving significantly shifted the challenges faced by the automotive industry from mechanical to electronic and software-related challenges.

In the past, most mechanical problems would occur during the vehicle design phase. To guarantee the availability of mechanical parts, OEMs would rely on maintenance contracts with Tier 1 and Tier 2 suppliers to ensure hardware availability for at least 10 years after the vehicle production ends. At the end of the value chain, end-customers would then indirectly pay Tier 1 and Tier 2 suppliers when repairs were needed until the vehicle's end-of-life.

With the shift towards software-defined vehicles, emerging issues are mostly software-related, and are now closer to the end-customer. Customers expect their vehicle to be secured and maintained up-to-date with the latest features, but they do not expect to routinely pay for security updates.

This rapid change is forcing the industry to move fast to catch-up, as illustrated by the incredible growth of software engineering departments in companies like Renault or Volkswagen.

"In 2011, Renault and Volkswagen didn't have a single in-house software engineer[1]" Today, both companies have large software engineering departments.

The same thing is happening on the supplier side. Continental AG, the first company to produce grooved vehicle tires, spent €31 million on internally developed software in 2021 alone[2]. Its Automotive Technologies group (covering Vehicle Networking and Information and Autonomous Mobility and Safety) has about 89,000 employees, while its tire business now only counts about 57,000 employees. Continental is not an exception. We can also name Valeo, known for wiper and lightning systems, or Marelli, known for exhaust systems. Both have followed a similar trend of expanding their skills towards electronic solutions and software.

Although companies are building in-house departments to further their software knowledge, there is a real gap between market expectations and the cost of increasingly complex technologies.

With all of this in mind, the challenge for OEMs will be to manage the growing complexity of systems while controlling annual recurring costs incurred in maintaining them.

Canonical has several years of experience in managing regular software deliveries associated with long-term support and maintenance. In the pages that follow, we will be reflecting on current trends in the automotive industry and share our guidance for automotive OEMs and Tier1 suppliers on how to optimise their spending in the coming years.

[1] https://mondaynote.com/code-on-wheels-a4715926b2a2
[2] https://cdn.continental.com/.../continental_annual_report_2021_1_01.pdf page 61

Setting the stage:
Redefined vehicle architecture in the data age

Looking at the current state of the automotive industry, it's clear that vehicles will handle tens of terabytes of data per day. Data will be gathered by various long and short-range sensors, such as LiDAR, and cameras, among other devices. The number of sensors in a vehicle is also likely to keep increasing.

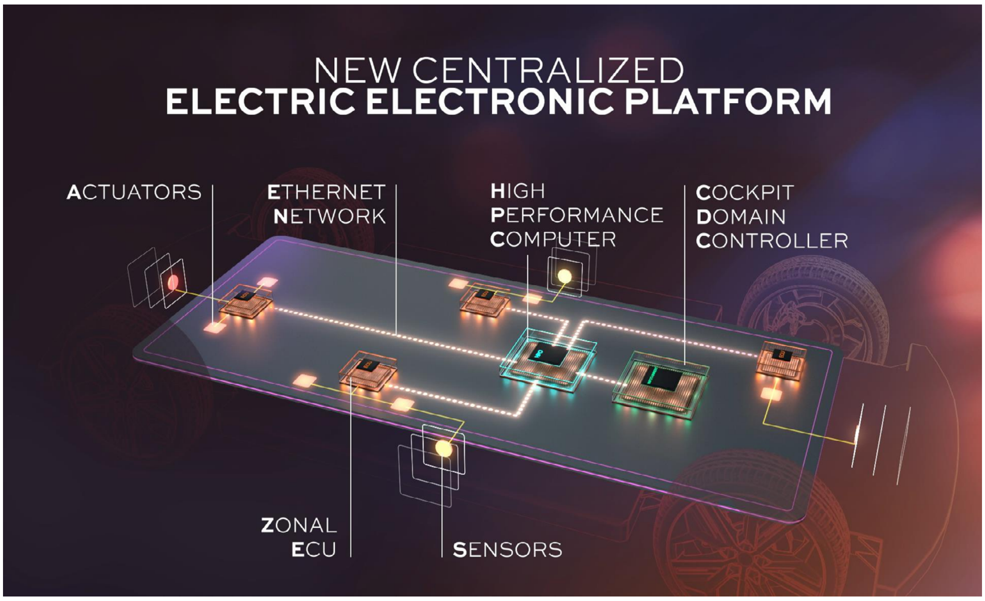
Vehicles need to embed computing engines capable of handling terabytes of data per day

On top of the data and computation required for autonomous driving, there's also passenger entertainment systems, and extended vehicle diagnostics (enhancing the diagnostics previously made through OBD-II[3]). All of these systems need processing power.

Will each vehicle broadcast *all* these bytes of data to cloud servers? That's very unlikely. Instead, some of the data will probably be processed locally inside the vehicle before broadcasting the most meaningful or aggregated data to the cloud.

As a result, an increased amount of data needs to transit on the vehicle network. Previously Electronic Control Units (ECUs) were scattered in several places inside the vehicle, close to associated sensors or actuators. Today, OEMs are moving towards a vehicle Electric and Electronic (E/E) architecture which combines and consolidates ECUs and interconnects them using an Ethernet network. This new architecture also includes 1 to 5 embedded High Performance Computers (HPCs).

As we will see in the following sections, the shift to this new architecture has many implications, especially when it comes to hardware and software maintenance.


NEW CENTRALIZED
ELECTRIC ELECTRONIC PLATFORM
ACTUATORS
ETHERNET NETWORK
HIGH PERFORMANCE COMPUTER
COCKPIT DOMAIN CONTROLLER
ZONAL ECU
SENSORS

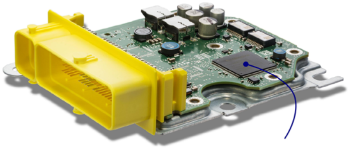
Source : Alliance Renault Nissan[4]

[3] https://en.wikipedia.org/wiki/On-board_diagnostics
[4] https://alliancernm.com/.../Alliance2030-Presentation-January-27-2022.pdf

Increasing complexity

Hardware complexity

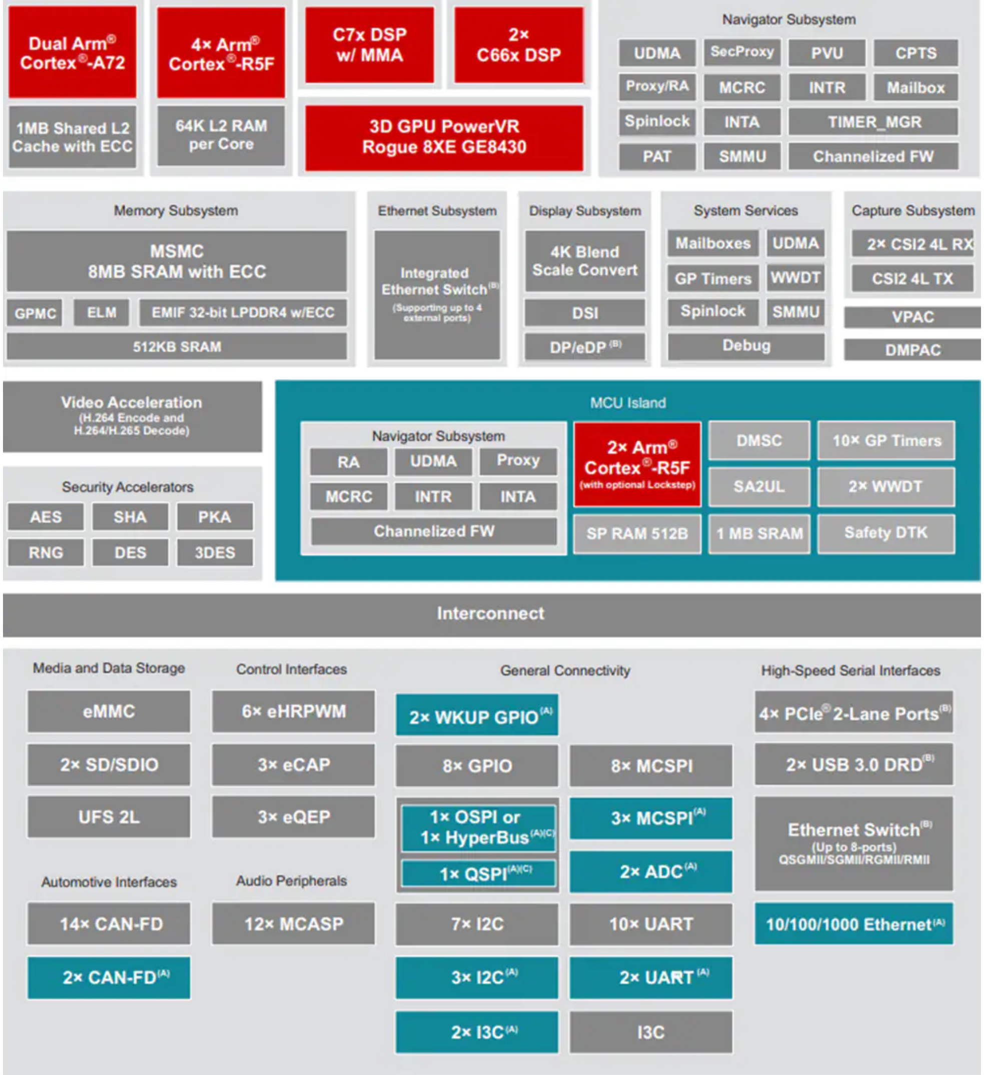
Electronics Control Unit (ECU) without its waterproof casing



Example of ECU
Source: Continental AG

In automotive, electronic hardware design is not based on a Central Processing Unit (CPUs) fitted on a motherboard with PCI[5] extension cards like in the PC world. Instead, it uses a System on Chip (SoC). A SoC is an integrated circuit that includes all or most computer components. Actually, in automotive, SoCs usually include several CPUs for main computing and safety purposes, including Digital Signal Processing (DSP) units, Graphical Processing Units (GPUs), Video Accelerators, Image Processing Units (IPUs), CAN bus interface… In other words, SoCs are quite complex systems.

Hardware is already complex, but it will need to be even more powerful going forward. This means complexity will increase.



Source : Texas Instruments TDA4[6] System on Chip (SoC)

And looking at the evolution of automotive hardware, it is evident that we have yet to reach a plateau in SoCs complexity. Actually, even if today's in-vehicle hardware is not yet reaching the teraflops-region, some chipsets with 16 CPU cores are starting to appear[7] on the market. Soon, automotive will have access to 80 core[8] CPUs and beyond.

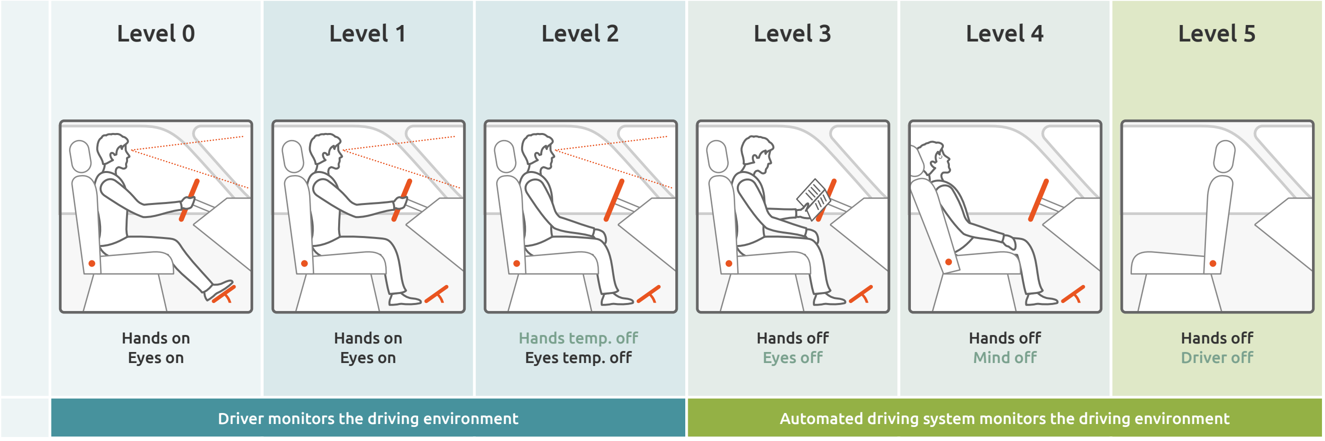
[5] https://en.wikipedia.org/wiki/Peripheral_Component_Interconnect
[6] https://www.ti.com/product/TDA4VM
[7] https://www.nxp.com/products/.../LX2160A
[8] https://www.adlinktech.com/.../hpc-ampere-altra-embedded-edge

Software complexity and microservices

The industry transition from a hardware model to a software model drives a change of development cycles, but also a full pivot in ways of working and required skill sets. This is creating additional challenges for the industry. Companies are hiring a lot but the cost of hiring and training professionals is quite high, and is likely to keep increasing. Attracting the right candidates who will be able to provide a solid base for companies to build a successful software strategy is another challenge. Some existing personnel from OEMs and their suppliers can also be trained to catch up with the new skill sets needed but it can be very difficult to reprogram the mindset of a whole company, especially when the field is very deeply specialised.

In parallel, the requirements for more and more sophisticated features have increased. According to McKinsey, *"the average complexity of individual software projects in the automotive industry has grown by 300 percent over the past decade"*[9]. Today, each vehicle has approximately 100 million lines of code. That's much more than in a military aircraft like the F-35[10] or even the Boeing 787 Dreamliner[11]. This number is expected to reach 200 million by 2025, and it will potentially reach 1 billion lines of code for a Level 5 Autonomous Driving system.

Following the SAE-defined levels



Following the SAE Levels of Driving Automation™

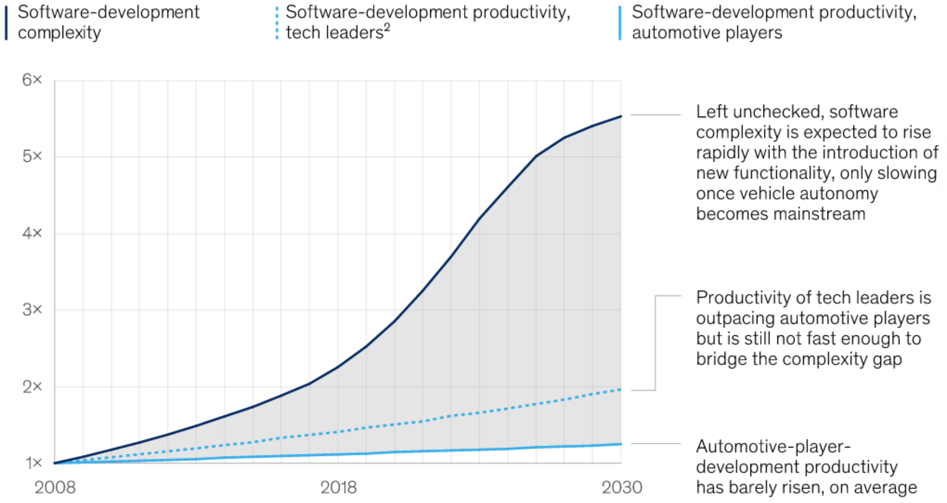
Adding to that, Artificial Intelligence (AI) or Machine Learning (ML) algorithms are also heavily contributing to the growing complexity of the software embedded in vehicles.

[9] https://www.mckinsey.com/.../cracking-the-complexity-code-in-embedded-systems-development
[10] https://www.volkswagenag.com/.../2019/03_march/1_CMD_Diess.pdf page 12
[11] https://www.gao.gov/assets/gao-16-350.pdf Figure 2 page 14/61

Software complexity has grown by 300 percent over the past 10 years

The automotive industry is confronting a widening and unsustainable gap between software complexity and productivity levels.

Relative growth over time, for automotive features,[1] indexed, 1 = 2008



Left unchecked, software complexity is expected to rise rapidly with the introduction of new functionality, only slowing once vehicle autonomy becomes mainstream

Productivity of tech leaders is outpacing automotive players but is still not fast enough to bridge the complexity gap

Automotive-player-development productivity has barely risen, on average

[1]Analysis of >200 software-development projects from OEMs and from tier-1 and tier-2 suppliers.
[2]Top-performing quartile of technology companies.

Source : McKinsey[12]

Cybersecurity threats

One of the first "hackers" in the software and Internet history was reportedly Kevin Poulsen. In 1983, he hacked the Arpanet, the predecessor of the Internet. Back then, stealing a car required a thief to break a window and connect the wires under the steering wheel. Today however, the growing presence of software in vehicles has made this industry vulnerable to cyber criminality.

The 1983 invention of CAN-bus by Bosch preceded the World Wide Web and Internet. It was not designed to be resilient to hackers, and this legacy embedded in modern vehicles is one of the several access points used by hackers. Just as Poulsen exposed the vulnerability of the Internet, the Jeep remote hacking incident[13] exposed the many security softspots in vehicle navigation systems.

A report from 2021 on global automotive security, states that *"there have been 110 Common Vulnerabilities and Exposures (CVE) related to the automotive industry, 33 in 2020 alone compared to 24 in 2019"*[14]. But not all threats are reported as CVE. As mentioned in that same report, in August 2020, over 300 vulnerabilities were found in 40 ECUs developed by 10 different companies. Authorities are seriously looking at the impact it creates on the economy. New regulations are emerging in response to these security threats.

In August 2020, over 300 vulnerabilities were found in 40 ECUs developed by 10 different companies.

The ISO/SAE 21434[15] *"specifies engineering requirements for cybersecurity risk management regarding concept, product development, production, operation, maintenance and decommissioning of electrical and electronic (E/E) systems in road vehicles, including their components and interfaces"*. In addition, the United Nations approved regulation No.155[16]. This new UN Regulation related to cyber security and cyber security management systems for vehicles outlines *"uniform provisions concerning the approval of vehicles with regards to cyber security and cyber security management system"* that will become mandatory as early as mid-2022 for new vehicles.

[12] https://www.mckinsey.com/.../the-case-for-an-end-to-end-automotive-software-platform
[13] https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
[14] https://info.upstream.auto/.../Upstream_Security-Global_Automotive_Cybersecurity_Report_2021.pdf
[15] https://www.iso.org/standard/70918.html
[16] https://unece.org/sites/default/files/2021-03/R155e.pdf

The challenge is not only to deliver a system, but also to maintain it over time. Are automotive companies prepared?

As a consequence of this regulation, OEMs and their suppliers will have to work together to provide software security patches or fixes to prevent and limit cyber criminality over vehicles' lifetime. Each software delivery, whether it is to provide feature enhancements, fix a bug or solve a security issue, will need to go through a cybersecurity assessment and a formal approval process before being pushed to the vehicle.

To meet this mandatory requirement, and customer expectations, OEMs and Tier1 suppliers should understand the complexity of the software they deliver in their product. In the software field, there's very rarely only one way to implement a functionality or a service. Software architects will have to make wise decisions as managing the software complexity is the only way to mitigate the cost of long term software maintenance.

But the cyber risk is not software limited, sometimes the risk lies in hardware too, as shown by the Meltdown and Spectre vulnerabilities[17]. Obviously, this makes designing the whole system even more complicated as it requires anticipating potential hardware vulnerabilities on top of software risks.

Safety considerations

System complexity is far from being the only challenge for the automotive industry. We've previously talked about cybersecurity: actually, security and safety are two separate topics. Indeed, a secure system does not ensure the safety of the people using it. A vehicle is a mobile device which often weighs more than one metric ton: it needs to be safe[18] for its passengers and surroundings. Even the most secure vehicle can be fatal without seatbelts.

As distinct these two concepts are, they are nevertheless closely intertwined. As proven by the Jeep remote accident mentioned earlier, a security issue can also lead to a safety issue. This is especially important to consider when more and more active safety systems designed to preemptively avoid crashes are implemented in vehicles. These rely mainly on software, and are therefore vulnerable to cyberthreats. That is the reason why Functional Safety (FuSa) was introduced in the vehicle industry.

The goal of Functional Safety is to provide a framework for OEMs and their suppliers to keep safety at the core of the different processes, from the conception of the vehicle to its production, delivery and maintenance.

The ISO 26262 , titled *"Road vehicles – Functional safety"* sets an international standard for the functional safety of electrical and/or electronic systems that are installed in serial production road vehicles. Functional Safety (FuSa) is defined as the absence of unacceptable risk due to hazards caused by malfunctioning behavior of electrical and/or electronic (E/E) systems.

It is not a hardware only problem, nor a software only problem. Rather, it is a system concern which involves specific actions and processes on both hardware and software sides.

[17] https://meltdownattack.com/

[18] In this white paper, we make reference to "safe" systems. In reality, a system is not formally safe but rather compliant to specific safety requirements.

Hardware Safety

On the hardware side, SoCs and ECUs must satisfy rigorous safety requirements before they are integrated into an automobile manufacturers' product. The ISO 26262 standard defines an Automotive Safety Integrity Level (ASIL[19]) to help map the applicable requirements and processes, according to the criticality of the system being developed. As an example, to achieve ASIL B, more than 90% of single point faults in a system must be covered by a safety mechanism. For ASIL D, the requirement increases to more than 99%. The standard defines what a single point fault is and how to generate estimated failure rates. There are many more definitions and assessment methods which are provided in the standard to help get a global understanding of system safety.

Estimating hardware failure rates

ISO 26262 describes several techniques to estimate failure rates within a semiconductor component, such as electrical stress, transistor-level failures or package failures. These types of failures and respective rates depend largely on circuitry type, implementation technology, and environmental factors like humidity, temperature, pressure, electromagnetic interference, etc. The standard also makes a distinction between the estimated failure rate and the estimated reliability of the element. Furthermore, it considers that a unique cause can lead to safety-related consequences in several separated elements. The standard addresses these cases through a Dependent Failures Analysis (DFA) and its associated Dependent Failures Initiators (DFI).

Detecting transmission problems

In order to meet these safety requirements, semiconductor companies have created a large spectrum of hardware mechanisms and solutions. These are the most common ones:

- Parity checking adds a so-called parity bit to on-chip communications traffic in order to detect data transmission problems. The simplified principle is quite straightforward: to transmit an even number, the parity bit is set to zero. To transmit an odd number, the parity bit is set to one. The receiver is always receiving an even number and knowing where the parity bit is located, it can remove it to find the number which was transmitted. This way, if the receiver gets an odd number, it means there's been data corruption during the transfer. While parity checks can detect transmission errors, they cannot retrieve the right message, so the message needs to be re-sent.

- Cyclic Redundancy Check (CRC) is another tool used to detect transmission errors. The remainder of a polynomial division of the message to be transmitted - which is the CRC of that message - is defined to have a fixed size and is concatenated at the end of the message, which is then sent to the receiver. Upon reception of the message, the receiver puts aside the portion which contains the CRC and runs its CRC computation on the message it received: the CRC it received along with the message should match the CRC it has computed on the received message, otherwise it means there's been a transmission error.

[19] https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en

- Error Correction Code (ECC) provides an enhanced solution to parity checks as they include a way to retrieve the initial message even in case of corruption. There are many flavors of ECC, even the wireless 5G standard is providing a new ECC mechanism.

While the goal is to detect transmission problems and retrieve the initial message transmitted, these techniques require additional bits to be transmitted along with the message. This has a direct impact on the actual network bandwidth, which then gets reduced: if for each byte you transmit 1 bit for error detection and 7 bits of actual data, then the bandwidth is reduced by 1/8, which is 12,5% in that case.

But sometimes the communication may simply break. Such a situation may occur when the communication is physically broken or more often when a process hangs for too long. So communication time-outs are used to detect communication losses.

Detecting runtime problems

As mentioned earlier, the standard covers issues which may occur at transistor-level. Semiconductor companies are using hardware checkers to verify correctness of operation. But they are also implementing safety controllers which gather error messages throughout the system, make sense of them, and communicate higher up the system.

Another recent technique to improve safety compliance at a lower cost, has been to use Built-In Self-Test (BIST). Initially, semiconductor manufacturers needed a way to identify post manufacturing defects at the factory plant, as quality check activity. By leveraging these enhanced in-system test capabilities, and using them while the processor performs its normal activities, these in-system tests provide means to identify run-time failures.

Redundancy as a detection mechanism

One usual way to handle safety requirements is to duplicate the system. Safety even increases if the two systems have been implemented independently, although based on the same requirements. Dual and Triple Modular Redundancy (DMR/TMR) refers to the general approach of doubling or tripling a system, or an element of the system, providing them with the same inputs and comparing the outputs. The triple modular case implements a majority driven decision process: if at least 2 systems provide the same outputs, then it must be the correct output. While in the dual modular case, if the outputs are different, an error is detected. TMR and DMR are used at different hardware levels from module-level to chip-level and even system level in some cases.

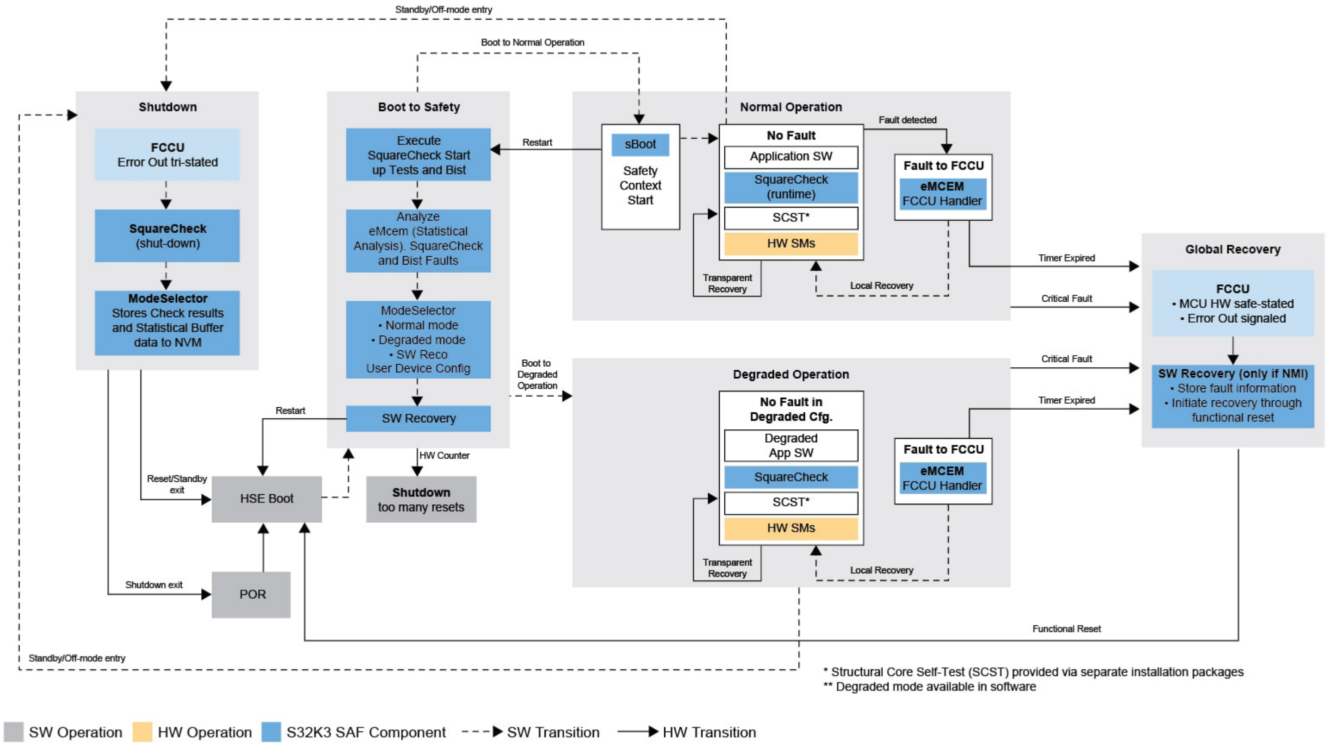
Some SoCs support the dual lock-step functionality. This safety hardware mechanism is a DMR implementation where two exactly same CPU cores are provided with the exact same software and are sharing the same clock cycles. At every clock cycle, a logic comparator block checks that the output of the two cores are identical, if not, an error is raised.

But higher performance CPUs are often more complex and less deterministic, so the dual lock-step approach won’t perform well. In that case, some more complex alternatives exist, like redundant execution using a “safety island” executed in a high safety integrity CPU core. Another solution called “CPU split-lock” is an optimised variation of the dual lock-step.

Like mentioned for the error detection techniques which were impacting the communication's actual bandwidth, the CPU redundancy techniques have also an impact, this time on the available processing power. Should it be using dual lock-step or its variants, the effective computing capacity of a SoC is then reduced by 50% (slightly less in the case of Split-Lock mode). Moreover, this leads to higher development and hardware costs as the redundancy has to be taken into account.

Software Safety

Semiconductor companies often rely on dedicated software to ensure hardware safety compliance[20]. In other words, some SoCs require specific software functions to be executed in order to reach their full safety potential, for example the Built-In Self-Test mentioned in the Hardware Safety section. Looking at the S32 Automotive Processing Platform, one of the trending automotive SoC families, the semiconductor company NXP says: *"The S32 Safety Software Framework components are involved during boot, runtime, and fault recovery."*[21]

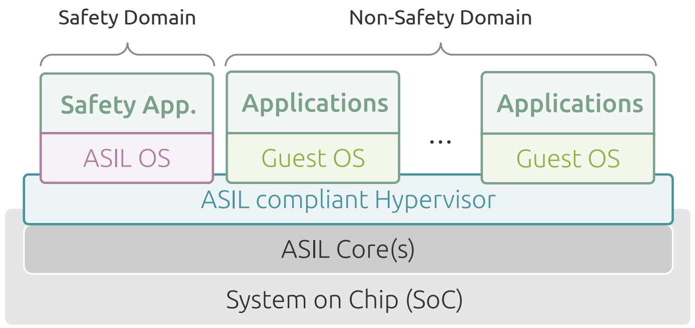


Source : NXP S32 Safety Software Framework[22]

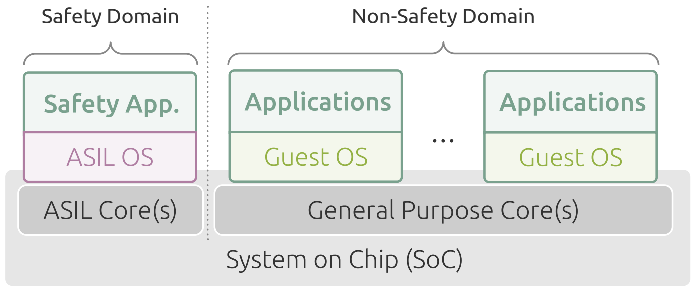
Not all software is safety compliant nor adapted to safety requirements. On top of that, certifying software represents a big investment. Therefore, as the ECUs are consolidated, the trend is to mix different safety criticality levels inside the same SoC or ECU. This allows for cost and performance optimisation . The in-vehicle infotainment ECU is an interesting example. The trend is to converge the centre stack, instrument cluster, heads-up display and passenger monitoring functions in the same ECU. Certain display and sound elements of these converged systems have a higher safety requirement than the others. The following drawing presents the main software approaches to support the ECUs consolidation.

[20] https://armkeil.blob.core.windows.net/.../state-of-the-art-stl-and-asil-b.pdf
[21] https://www.nxp.com/docs/en/product-brief/S32SAFPB.pdf
[22] https://www.nxp.com/design/automotive-software-and-tools/s32-safety-software-framework-saf:SAF

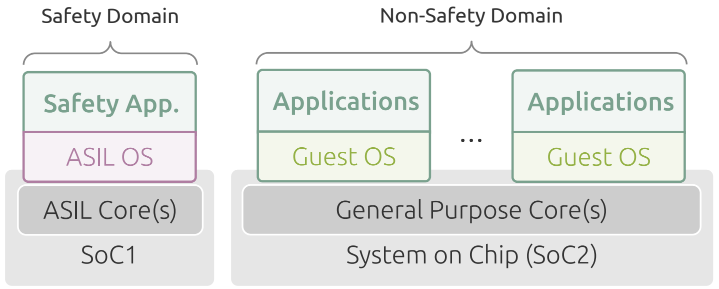
a) Single type of Core + Hypervisor



b) Hardware Partitions



c) Hardware Separation



The main differences between these options is where the processing of safety-compliant functions is performed.

- In option a), the separation is managed by the ASIL/Safety compliant hypervisor.
- In option b), it is done by using 2 types of cores inside the SoC, with for example one or several ARM Cortex-M cores for vehicle communication and safety needs and a set of general purpose cores for higher-end compute functions.
- In case c), the separation is made outside the SoC, at the hardware level, using 2 different and dedicated SoCs (one for safety and one for general purpose).

Developing software on such architectures is not only complicated, it creates complex integration and debugging challenges.

Many other variations and approaches are possible, like introducing an AUTOSAR Adaptive stack to the system. These are optimised options often based on specific SoC capabilities which then usually bring more complexity.

Hardware complexity will simply grow to bring more computing power to SoCs. The safety requirement will be adding complexity on top. It will become more and more complicated to combine computing power and safety into one SoC. Semiconductor companies will have to increase the cost of SoCs supporting

Safety requirements. Also, the more complex the whole system is, the more difficult and costly the certification. And we expect that the price increase will not be linear but more likely exponential.

Automotive companies must brace for the impact of high software maintenance costs.

But there's more to do on the software side to ensure Safety compliance. ISO 26262 requires systems to be developed by following a V-Cycle. It means that first the customer requirements have to be captured, then features have to be defined according to these requirements. Features then need to be broken down into hardware and software elements; software elements into functions. Specific tests need to be defined and created, or re-used, for application at the function level, software level, hardware level and then at a system level. This system development approach comes with its set of processes to implement software, do code reviews, capture deviations to requirements, run tests, measure test coverage, etc.

Now let's assume for a minute that the existing 100 million lines of code embedded in current vehicles are all Safety Compliant, which is not the case. How long would it take to write another 100 million lines of code in order to get to 200 million by 2025, following the ISO 26262 V-Cycle model? How many engineers?

Looking back at the existing 100 million lines already produced, OEMs and Tier1 suppliers should be able to make a rough guess of how much it will cost them to add another 100 million lines based on COCOMO or any other method. That would be the forecasted cost, if they don't change anything.

Then what do these effort estimates look like for 1 billion lines of code for a Level 5 Autonomous Driving system? Again, how long will it take to write the specifications, implement and test such a size of software, if reusing is not possible for the most part?

The only sustainable way forward is to use open source and we will provide more details further down in the recommendations section.

Freedom from Interference

As mentioned earlier, the trend is to consolidate hardware ECUs. Most of the time, it means mixing safety-critical and general processing, into the same ECU. But a Safe system must guarantee "Freedom from Interference". In other words, if a system combines Safety-critical and non-Safety-critical elements (like inside the same ECU), it should be proven that the non-Safety critical environment cannot interfere with the Safe part of the system, leading to a safety issue[23].

This means, for instance, ensuring the scheduler cannot be corrupted, a process cannot stall the system, and the memory heap is resilient to buffers-overflows. If we look at the Common Vulnerabilities and Exposures (CVE), most of them are based on backdoors, memory overflows, or unexpected/undesigned behavior of a software (or hardware) element: so the CVE highlight safety threats. Everyday new CVE are discovered. Even ASIL D compliant Real Time Operating System (RTOS) are not exempted from Safety vulnerabilities, even if they were designed according to ISO26262, following Automotive Spice processes.

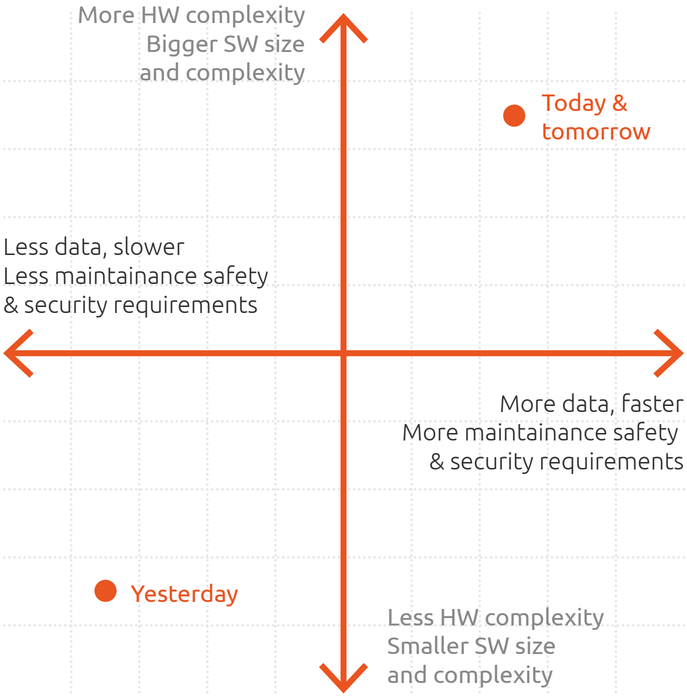
At the end of the day, everyone agrees that there's no Safety without Security.

[23] https://www.tuvsud.com/...top-misunderstandings-about-functional-safety.pdf page 20

The more complex the system is, the more complex it is to assess and prove its safety compliance.

ISO 26262 alone is insufficient to guarantee automotive Safety. The state-of-the-art considers that analysis of a system should be enlarged to cover Safety Of The Intended Function (SOTIF), as documented in the standard ISO/PAS 21448[24]. Yet another level of complexity to be managed.

Recommendations:
a well-oiled approach to tackle complexity and safety issues



At this stage we covered:

- The need to process more data, faster, and within the vehicle
- The challenges of increasing hardware complexity
- The issues with software proliferation
- The additional complexity introduced by cyber threats
- The need for long term software maintenance of the software
- The safety requirements which impact software and hardware, separately and jointly

In the pages that follow, we will explore a set of best practices to tackle these challenges, using examples from industries like avionics, open source software development, and others.

[24] https://www.iso.org/standard/70939.html

Limit safety contamination

We have already proven that the only sustainable way forward is to use open source. But this should not mean expecting open source software to be rewritten to meet automotive requirements. For instance, safety contamination refers to when safety starts to get required on more and more parts of the system. Not aiming to increase end-user safety, but rather driven by shorter time to market bypassing a proper architecture study. Safety contamination significantly limits the capability to use open source, as the vast majority of open source software is not safety-compliant, and will never be.

Over-requesting safety compliance from every part of the system, like asking every element to be ASIL D, may seem easier for the OEM to get ASIL B to ASIL D compliance on the system. But it increases complexity and cost while reducing functionality. For instance a safety-compliant OS supports a reduced set of functionalities as compared to a more generic OS. Moreover, maintaining safety and security compliance together raises another set of complexity since each change, like a CVE patch, will require a Safety assessment.

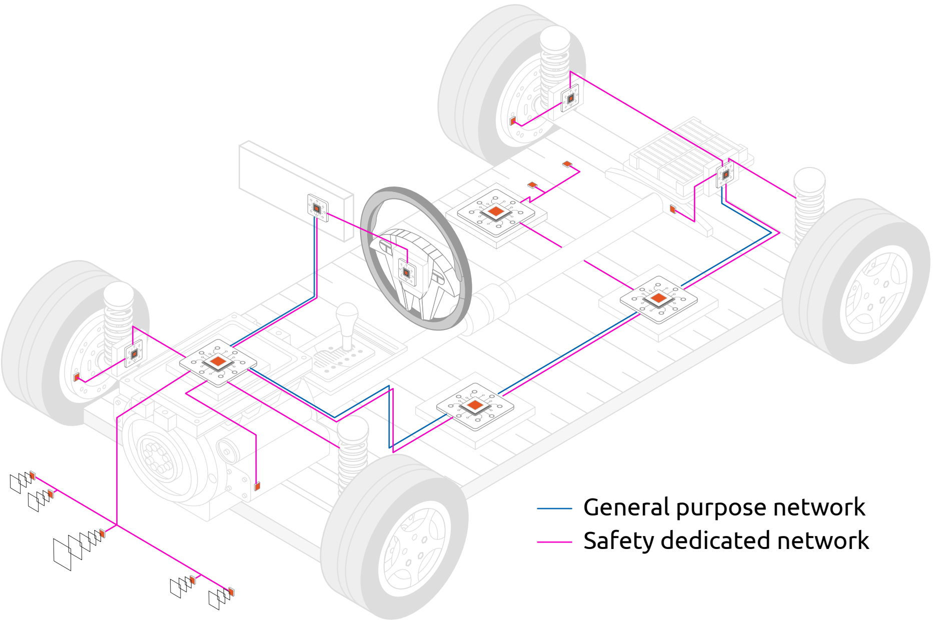
As a consequence, once deployed in a vehicle, the cost of operation for such a system can increase rapidly. And once deployed, the system architecture might be difficult to change. The only way to avoid a cost explosion for OEMs is to wisely control the safety boundaries of the system.

Recently, in a rush to Autonomous Driving and ECU consolidation, Safety requirements have contaminated many elements of the vehicle, like the in-vehicle infotainment system.

To avoid safety contamination, automotive companies can learn from other industries. For example, avionics architecture defines a clear separation between safety and non-safety related elements of the system. For instance, the in-flight infotainment system doesn't interfere with the safety elements of the aircraft. However, some safety elements can take control over non-safety ones, like the captain's announcement over the in-flight infotainment system.

Our recommendation for future Electric and Electronic Architecture is to consider a way to support two types of communication channels (at the hardware level, between ECUs): a safety channel and a general-purpose one. A general-purpose element of the system may interact with a safety-compliant element (asking and reading a status from a sensor or an actuator), using cloud-native types of messaging systems - the safety element replying when available: safety first. The general-purpose element should not be allowed to set a parameter or control a Safety element.

The recommendation is not to recreate safety and non-safety ECUs. The idea is that the safety and general-purpose features would be handled by two different types of SoCs/CPUs, even if they are located inside the same ECU (as presented earlier in option c) *Hardware Separation*). This way, not all the HPC/edge SoCs/CPUs will need to be connected on the safety communication channel, hence reducing the complexity of their hardware and software design.



McKinsey expects that Hardware and Software will be sourced separately making *"both sourcing more competitive and scaling less complex and allow for a standardised platform for application Software while maintaining competition on the Hardware side"*[25]

For the same reason but going further, the next step is for OEMs and Tier1 suppliers to source Safety and general-purpose elements separately, both on hardware and software sides.

Hardware Safety requirements create a barrier to entry, preventing a larger competition. Working on the system architecture to remove Safety requirements from some hardware modules (like HPC), will allow a renewed competition, leading to Bill of Material (BOM) cost reduction.

For instance, semiconductor companies might take the opportunity to get some CPUs compliant with AEC-Q100 to serve the automotive industry, without going through the process of ASIL compliance. We can expect to see more powerful CPUs, with more Cores, without much price increase compared to the PC/Server prices.

Now, how many times have prototypes been created on Linux and then had to be reworked, ported, and adapted to finally run on a "Safe RTOS"? Imagine developing a Proof of Concept (POC) and being able to run it almost instantly on the target hardware. Separating Safety-compliant from general-purpose will highly reduce the need for Safety compliant virtualisation solutions, and will make a much more tailored use of a Safety-compliant OS (based on Linux, or not). OEMs will then be able to use Linux and open source software. And like for the Safe OS usage, getting the Safety requirements in limited areas will also improve resources usage: reducing the number of required engineers having Safety knowhow. Hiring will be easier.

[25] https://www.mckinsey.com/.../automotive-software-and-electronics-2030-final.pdf page 8

Embrace open source and Linux

Are Facebook/Meta, Airbnb, Netflix (and many others) successful because they tried to replace Microsoft Windows or Linux with their own operating system, or because they used open source and focussed on their customers? They definitely have a very good grasp on their software stacks, but they don't waste effort on re-inventing what already exists in the open source community. If they find a bug, or develop an enhancement to an open source module, then they publish it to the community so that their feature gets upstream, benefits to others and gets maintained by the community. This is how open source improves and grows. We recommend OEMs and Tier1 suppliers to partner with commercial-grade Linux providers to get long term support and security maintenance for both the Linux kernel and extended open source packages. They will then be able to focus on their software solutions and drive competitive differentiation with applications and services developed for customers.

And because the vehicles are connected to the cloud, OEMs should really consider the benefits of working with a single Linux distribution provider for their cloud servers, engineers desktops, and in-vehicle Linux software. OEMs will then get access to premium support at a lower fee, while reducing the burden of managing multiple suppliers, contracts and development environments.

Canonical has a 17 years track record of delivering a Linux distribution release every 2 years and then maintaining it for 10 years[26]. Our partners can benefit from our reliability in delivering Ubuntu and continue to receive security updates for the Ubuntu base OS, as well as critical software packages and infrastructure components with Extended Security Maintenance (ESM)[27]. We can certainly help OEMs and Tier1 suppliers in their Open Source journey.

Develop next generation automotive applications

This expansion of software code bases is not specific to the automotive industry. To maintain a balance between software complexity and increasingly sophisticated features, the IT industry at large moved away from a static and monolithic approach to adopt the concept of microservices and high availability software designs.

The case for microservices

A microservice-based application architecture implements the idea that a single application can be developed as a suite of small, narrowly focused, independently deployable services. Each microservice runs in its own process and communicates with a lightweight mechanism, often an HTTP resource API. Those services are encapsulated for specific business capabilities and are deployed independently using a fully automated mechanism.

[26] https://ubuntu.com/support

[27] https://ubuntu.com/security/esm

Canonical provides security patches for more than 28,000 Open Source package[28] and the list keeps on growing

As an analogy, a way to understand the transition to microservices could be via the metaphor of an old house. Transitioning to a microservices model is like opening the wall between the kitchen and the living room, removing the handcrafted kitchen's cabinets and replacing them by modular systems from a big brand, adding kitchen appliances and television sets which can communicate on the house local network and through the internet. The outside walls of the house have not changed, there's still a kitchen and a living room, but it's all modernised, much more comfortable and brings a new set of functionalities.

Along this process of refactoring the legacy applications, part of the legacy code gets usually replaced by open source counterparts, simply because this open source is maintained by others (i.e. the community), which means the one using the code doesn't need to bear the full cost of maintaining that code. In addition, the open source code is used by many more users, covering many more use cases, than the legacy code it is replacing: in other words, this open source code is expected to be more reliable.

The transition to a microservices based model should aim at replacing 70% (or more) of the legacy software by open source equivalent software. From there, the effort can focus on the remaining 30% (or less) of the software which creates a key differentiator and generates the value-added for the company.

How to handle a Single Point of Failure

High-availability software designs aim at identifying Single Point of Failure (SPOF). A SPOF is a part of the system that, if it ceases to function properly, will cause the entire system to stop working. For instance, a single engine plane cannot take off if its engine doesn't work, while a commercial aircraft with a twin engine can take off even if one of its engines stops during take off. In the case of an application designed using microservices, it's easier to identify which are the SPOF and try to remove them. Several methods exist to remove a SPOF: mirroring, load balancing, replication, self-healing… While the purpose of this document is not to get into deeper details, the key takeaway is that high-availability designs are used by the IT industry to ensure fault tolerance and 99.9% uptime of the servers. In addition, it also allows maintenance, upgrades and experiments on-the-fly without having to shutdown the machines. Shouldn't the vehicles be fault tolerant and remain functional most of the time, having a high uptime? Shouldn't we be able to push upgrades without having to stop the vehicle? When the IT industry set these goals for itself, they looked quite ambitious. But it works and there's no reason it cannot be applied to the automotive industry.

The Internet has proven to be stable and running 24/7. High Availability software and microservice-based application architecture are key pillars to meet this goal. Software running in data centres is largely hardware agnostic. It can easily be deployed in many places in the world, can be hosted by public clouds like Amazon Web Services, Google Cloud Platform, Microsoft Azure or even on premises, with no need to rewrite the applications/services when switching hosts. And that's only a subset of the advantages.

Microservice applications are containerised software: it means they run in a "box" (i.e. the container) which contains what they need to be executed. Several advantages come from this approach:

[28] https://ubuntu.com/support#apps-support

- If a microservice misbehaves, it's only the context or containment it is running in which is affected and not the entire system which gets down.

- To provide the microservice with what it needs, a virtual hardware and OS environment is made available in the container. This creates a hardware abstraction which makes the microservice agnostic from the real physical hardware.

Websites, web-based games or web-based applications are hardware agnostic: they can run on many devices which have hardware specificities like screen size, orientation, GPU, webcam.

For embedded mission-critical software, we think snaps[29] enable the required hardware abstraction (for an application to be hardware agnostic) while increasing stability across the system (one function misbehaving in an application cannot endanger the whole system).

A snap is a bundle of an application and its dependencies that works without modification across all major Linux distributions and even bespoke distros that integrate snap support[30]. There are already thousands of snaps used by millions of people across 41 Linux distributions[31]. Tutorials, support materials, tools to create and deploy new applications or patches - all these things already exist. It's also possible for OEMs and Tier1 suppliers to create snaps and deploy them to both new ECUs and legacy ECUs running Linux. Everyone is free to use them on Linux computers (and IoT devices), there's no fee to pay, no lock-in.

In addition, Canonical can support OEMs and Tier1 suppliers in creating their private store[32] where they can host their middleware, applications and services specific to their use cases and product. The Snap ecosystem is rapidly growing and Snap users can use open source snaps.

One of the key benefits of snaps is operations and application management. Indeed, they provide out of the box security and can benefit from hardware portability, without compromising system integrity and reliability. Any OEM can very easily deploy snaps that can be operational without additional porting efforts on any device.

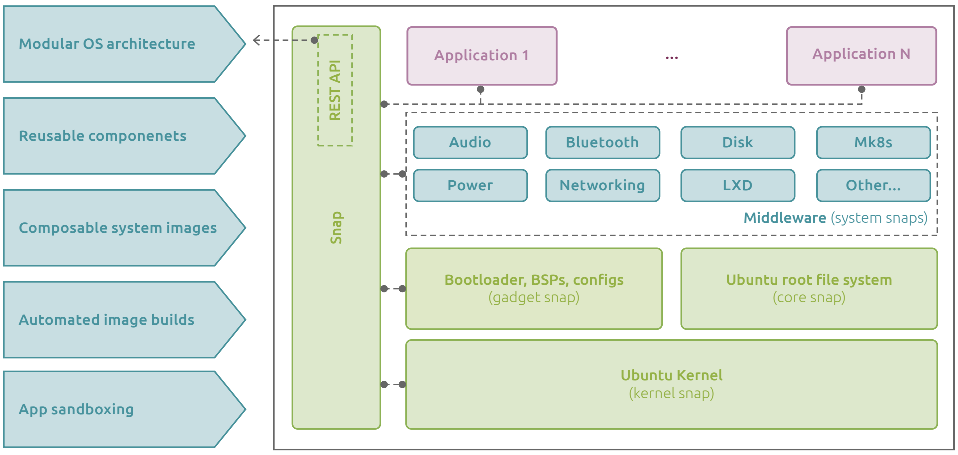
Canonical took this approach even further by designing a new Linux OS paradigm, which is fully containerised based on snaps: Ubuntu Core[33]. Ubuntu Core is secured by design: it provides isolation via AppArmor and Seccomp. It also provides TPM support, Secure boot and Full disk encryption.

With this holistic approach of snaps, embedded systems benefit from security and immutability, as well as modularity and composability. Software is updated over-the-air through deltas that can automatically roll back in case of failure (from the Linux OS to any individual application).

Canonical supports Ubuntu Core long-term, delivering kernel patches and bug fixes continuously for 10 years (more on request).

[29] https://en.wikipedia.org/wiki/Snap_(software)
[30] https://snapcraft.io/docs/installing-snapd
[31] https://snapcraft.io/store
[32] https://ubuntu.com/internet-of-things/appstore
[33] https://ubuntu.com/core

Containerised Ubuntu for embedded Linux:



It took more than 30 years for the automotive industry to transition from CAN bus to Ethernet (and it's only a partial transition). Building failure-resilient systems is a must-have for automotive and there are many ways to achieve this. Moreover, handling cybersecurity requires system designs enabling cost-effective maintenance. Due to the ISO/SAE 21434 cybersecurity regulations enforcement, it is the right time to reconsider the vehicle hardware and software architecture in the light of the safety, security and scalability triangle.

We recommend the same approach used in enterprise datacentres, with microservices and High Availability software designs, combined with the latest containerisation software environments, down to the OS itself.

Adopt a customer focused approach

The infotainment systems in our vehicles were outdated by more than 4 years, compared to the design and functionalities we were getting from our mobile phones.

From Hardware to Software defined devices

The smartphone industry shift

Software	Hardware specific	Open Source low-layers and Hardware-agnostic applications
Staffing	Savvy resources, Difficult to recruit	Extended pool of resources from Open Source community
Scalability	Very specific, low scalability	Scalable across a whole ecosystem
Cost of Development (per lines of code)	Expensive: all tailor-made and specific	Adjusted cost: leverage Open Source and limited specific software usage

Equivalence to mobile phone features shall not be our goal, but only a milestone. Vehicles are capable of doing much more than mobile phones (like moving by themselves).

Firmware or software over-the-air update (FOTA/SOTA) is a first step, but it’s not groundbreaking: our phones and computers have been doing this for a while. And the question is what the update brings to the end user. If it’s a 2 years old set of features or aesthetic-only, then it’s very likely far away from the customer’s expectations.

OEMs and Tier1 suppliers need to provide faster software development and delivery, in order to keep in sync with the pace of innovation coming from other industries (like mobile phones). And again, vehicles are the most advanced devices for consumers. They shouldn’t only reach the same level as other industries, they have the power to set new expectations.

Software development needs to be cheaper for OEMs and with a faster time to market. This will be possible if a major part of the software - the part which will delight the end customer and the part which will host 3rd party services - runs in a non-safety environment. The hardware also needs to become easier to source (at least for those non-safety environments), using less automotive specific SoCs.

Then OEMs will be in a position to offer new services to their customers and an ecosystem of 3rd parties. A big part of the smartphone's success comes from the ecosystem of 3rd party applications and services; enabled by the platform (i.e. the smartphone and its application store).

Integrate a software company mindset

While the automotive industry is moving from hardware-defined vehicles to software-defined ones, it might be worth understanding differences between these two worlds:

- Hardware design is usually driven with cost saving in mind: how to make $1 cost reduction on the hardware? Because saving $1/device on 1 million devices generates a $1 million saving.

- Software is usually looking at scaling factors: how can we make the software solution available to a million users? Because charging each user $1/month, generates a $1 million/month revenue.

This is a quite fundamental difference.

Traditional carmakers used to look very carefully at the hardware BOM. Automotive development processes require clear and detailed specifications. From there, selections are made towards the most cost effective hardware options to meet the specs. As a result, the vehicle is kind of limited by these hardware choices and will remain that way for its entire lifetime.

Companies like Tesla tend to choose more powerful hardware options. It doesn't mean they don't care about the BOM cost. They exercise greater latitude to consider components which yield a competitive advantage. Tesla made it to the point of being able to play games in a car. Not the kind of games available on flight entertainment devices, but rather the ones requiring a gaming computer at home. Is it useful to be able to play such games in a vehicle? Maybe not, but it's not the point. The question is if customers are happy to know they can do it? Because the feature itself matters less than customer satisfaction.

Such hardware and SoCs are more expensive but they also provide the compelling advantage to enable deployment of new features year on year. Again, customer satisfaction is the key driver. And for sure, most customers will prefer the evolving vehicle: it's like getting a vehicle refresh on a regular basis. Yes, such choice leads to a BOM cost increase compared to lower SoCs options. But every OEM makes strategic choices, be it bending a complex door shape, adding a retractable rear spoiler, using specific wheel rims… These choices serve to differentiate the OEM and build its own brand recognition. Some OEMs consider in-vehicle hardware and software computing power as a very valuable asset to customer satisfaction.

But there's another interesting factor: 30 years ago, a consumer electronic device, a phone or a vehicle had an electronic hardware cost over the lifetime of the product which was flat at zero cost for the manufacturer. It's not the case anymore since software took an important role in those devices. Today, customers expect to receive new features over time, and in addition, the system needs to stay secure, as a consequence, the software comes with recurring annual costs. For that reason, more than the hardware BOM cost, the Total Cost of Ownership (TCO) of the Electric and Electronic Architecture will need to be measured and optimised. Reducing system complexities will be key to drive the lowest maintenance costs.

Finally, OEMs opting for an architecture mostly based on off-the-shelf components, requiring minimal adaptation to automotive (for instance, limited to AEC-Q100 and extended product availability, in case of CPUs), will be more resilient to sourcing shortage problems: on the chip sides but also on the engineering and hiring side.

Preparing for the road ahead

After decades of building impressive combustion engines and working on their respective brands positioning and identity, OEMs have recently seen all cards redistributed: new brands came up, electric vehicles have proven to be faster, sportier and also very profitable (considering Tesla's market capitalisation and profits).

To catch-up with all these recent market changes, they launched initiatives in many directions. But today's decisions will have to be dealt with tomorrow, when the time comes to maintain the vehicles deployed in the field. Our recommendations aim to help OEMs and their suppliers in taking the correct decisions today, for a maintainable future.

Authors @canonical:
Gordan Markus, Silicon Alliances Partner Manager
Bertrand Boisseau, Product Manager, Automotive
Matthieu Sarrazin, Commercial Director, Automotive

Canonical Limited, Registered in England and Wales, Company number 110334C Registered Office:
12-14 Finch Road, Douglas, Isle of Man, IM99 1TT VAT Registration: GB 003 2322 47

ubuntu®
Delivered by Canonical