Papier scientifique présenté lors du congrès "SIA CESA 2025"

Software changes the Automotive Industry - Software Stacks enforces new Development Paradigms

Detlef Zerfowski, ETAS GmbH, Stuttgart, Germany

Abstract: The automotive industry is undergoing a period of radical change. A key driver of this development is the steadily growing share of software in the automotive value chain and beyond that in the entire mobility sector. Starting from an industry that has been mechanically and electronically determined in recent decades, the importance of software is becoming ever greater. This results in a compelling need for stable software stacks that can be used across entire industries, rather than being company-specific. Only then will OEMs be able to focus on differentiating parts of their software. As a result, a mindset change needs to be established in the industry that emphasizes the reuse of software components from a wide variety of sources. The aim is to adapt one's software to the software components from other sources and not vice versa. This results in new collaboration models in the automotive industry.

The article examines the above aspects for the different types of automotive SW stacks and discusses their special features. It shows how different SW stacks contribute to the so-called decoupling of the software from the hardware and where technological limits exist for this. The use of open-source approaches (including the industries' approach within the European public- funded projects like FEDERATE [3] under the umbrella of the European Chips JU [9]) will also be considered in this context.

Target group of the presentation: Software-savvy managers who want to better understand the software technology challenges in the automotive industry across application domains and industry partners.

Keywords: Software defined Vehicle, AUTOSAR Adaptive, AUTOSAR Classic, Eclipse Foundation, Open Source, Transformation, Collaboration.

Introduction

Over the past two decades, the automotive industry has developed from a purely mechanical industry, via electronic systems, to a software-driven industry. As part of iterative, evolutionary development, engineering approaches from one engineering discipline were adopted into others. In particular, standardization has been and is an essential means of mastering the complexity of integrating components in vehicles. This is not just limited to mechanical components, but especially to electronic components. Without standardized, automotive-specific communication protocols (e.g. CAN, LIN, FlexRay, etc.), the construction of complex E/E architectures in the vehicle, sometimes with more than 100 electronic control units (ECU) from many different suppliers would not have been possible. If proprietary protocols had been established here, specific hardware and software solutions would have had to be developed for each OEM based on specific automotive requirements. The repeated specification and development costs would have had to be borne in full by each OEM. With the increasing importance of software and the decoupling of software from hardware, the automotive industry is undergoing a disruptive transformation. Besides technology, this transformation impacts culture, organizations, and business approaches within each automotive company. New collaboration models across the industry and beyond need to be established. Especially, open-source approaches become more relevant.

Software – An Industry Disruption

Even though, the automotive industry is experiencing a disruption, it seems to be unique. This transformation creates a lot of uncertainties across the entire industry.

Looking to the past, quite often transformations happened in a disruptive manner, with a lot of resistance. Examples are:

- End of 18th century: Introduction of steam engines.
- Mid of 19th century: Introduction of railroads ("Human beings could suffer serious damage if they travel on railways faster than 30km/h").
- Beginning of 20th century: Introduction of automobiles (Kaiser Wilhelm II: I believe in the horse, the automobile is a temporary phenomenon").
- 1950th: Electronics, nuclear power.
- 2000th: IT, social media, multimedia, AI ("AI will take all our jobs and leave us unemployed").

During every transformation period, there are promoters, shaping the transformation and enemies, who fear the transformation.

Within the automotive, you could see similar patterns on a shorter time scale as well.

- 1970th: Mechanical engineers designed mechanical injection components to drive the powertrain.

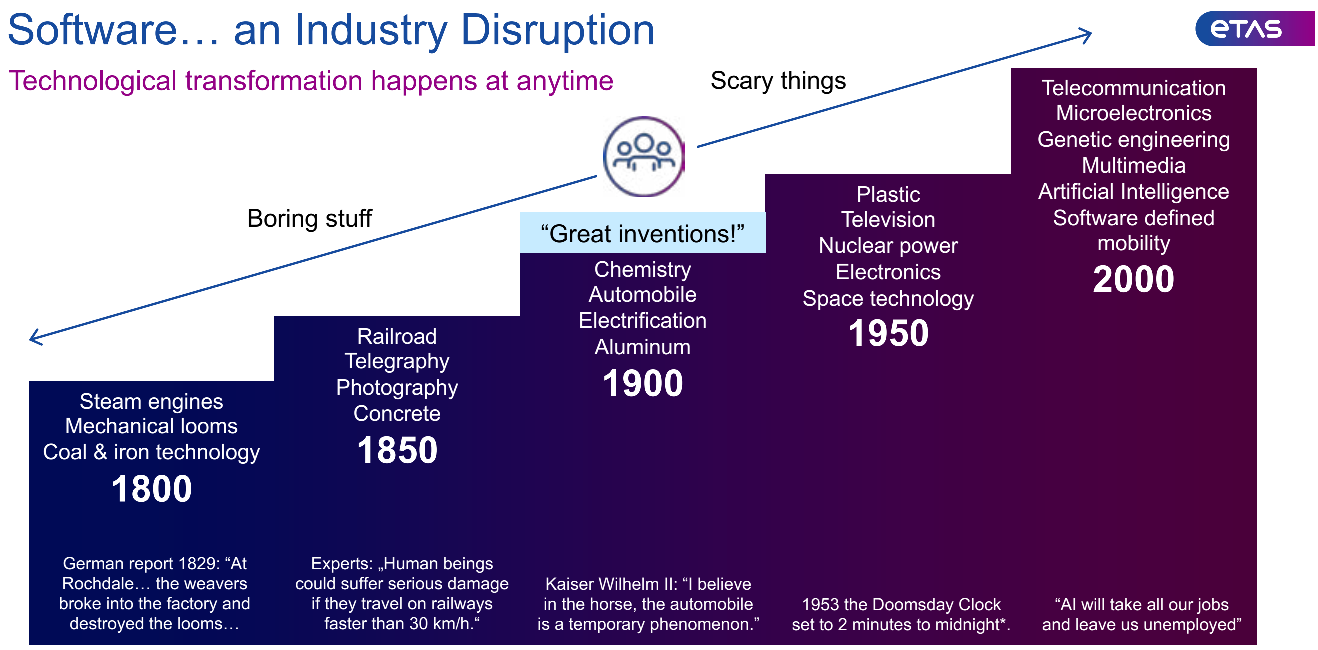

Fig. 1: Transformations over last centuries.

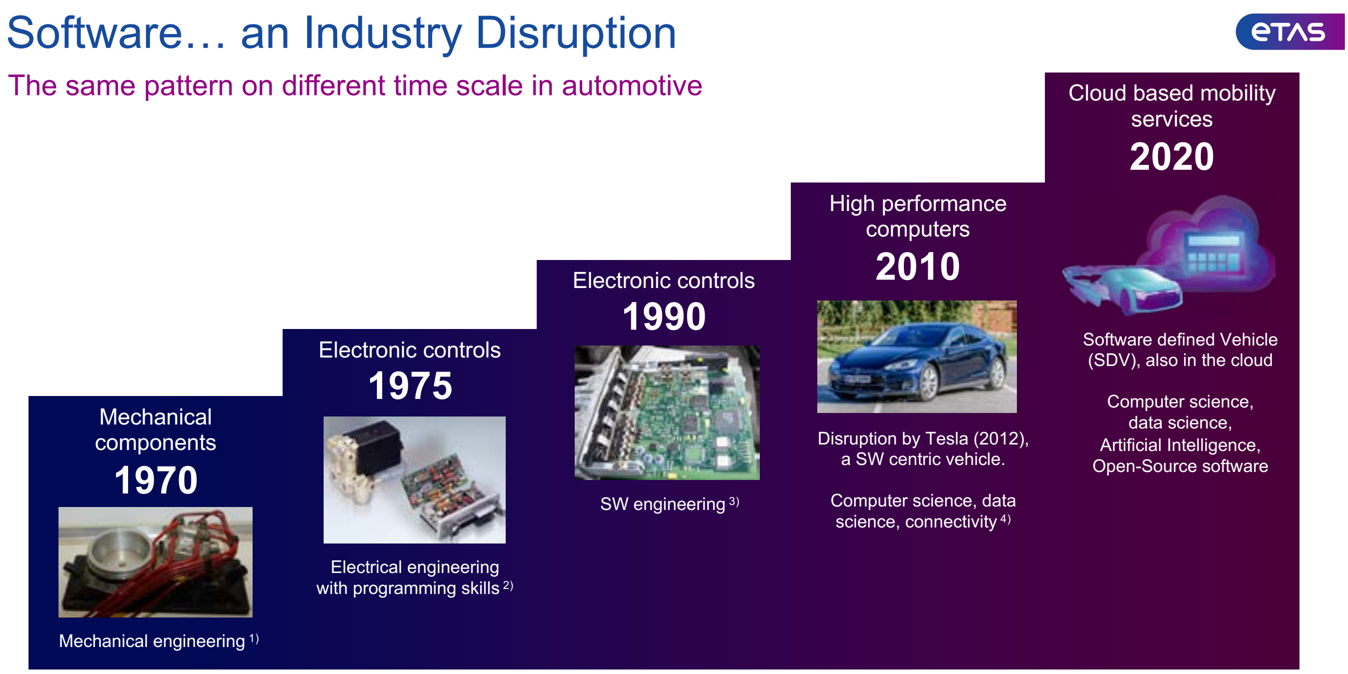

Fig. 2: Transformations over last decades in automotive.

- Some years later electronic components were entering into the domain and electrical engineers got into the lead, which was quite uncomfortable for mechanical engineers.
- In 1990th electronic control units with little portions of software (mainly assembler) started to spread around. Some years later, discussions started on the utilization of programming languages like C. Assembler programmers did not appreciate the advent of this high-level programming language and actively tried to block this new technology. Especially electrical and control engineers with programming skills jumped on this technology.
- In 2010th the sunrise of automotive high- performance computers entered vehicles. TESLA's disruptive approach surprised the entire industry. Suddenly, high-performance, data-driven computers got into focus. For the classical deeply embedded focused automotive industry it was like a shock.
- In the 2020th computer science is taking over concerning innovative applications and services inside vehicles and offboard. Everybody speaks about cloud services, AI, open source, data science and the Software-defined Vehicle.

Software is unequal Software

Previous paragraphs describe common patterns of transformation, especially in the automotive domain. In the following, we dig deeper into recent situations and challenges of the transformation coming with the Software- defined Vehicle.
The term Software-defined Vehicle leads to a lot of confusion. Different people have a completely different understanding of the meaning of a Software-defined Vehicle. Many managers from the classical, deeply embedded automotive domain try to explain the concept as a vehicle with an increasing amount of software, which can easily be updated over the air. Quite often one could hear the explanation “smartphone on wheels” (which I disagree with, because vehicles carry many different types of ECUs on four wheels).
Another observation is, that the automotive industry speaks about THE software. Already these kinds of statements lead to misunderstanding. People coming from different application domains and therefore from different software domains are working in different contexts and speak in different terminologies. To get a better understanding we look at different software categories of the Software-defined vehicle.
The time of automotive electronic control units (ECU) was driven by deeply embedded, hard real-time applications like engine, braking, airbag ECUs, etc. This kind of ECU hosts real- time operating systems on which typically AUTOSAR Classic stacks are running. The application software are signal-based control algorithms, closely linked to the physics of connected sensors and actuators. The µ- Controller based implementations are strongly hardware-dependent due to severe hardware constraints, especially with respect to run-time and memory consumption. Additionally, many of these mechatronic systems must fulfill stringent Functional Safety requirements (according to ISO 26262). The software stacks are project- specific implementations and are delivered as integral parts of manufactured ECUs.
With the first µ-Processor based ECUs (vehicle computers), a completely new software category entered vehicles. µ-Processor based systems come with POSIX (Portal Operating System Interface) operating systems like Linux derivates. These operating systems provide an abstraction from the underlying hardware, enabling the decoupling of the software from the underlying hardware. Due to fewer constraints regarding runtime and memory consumptions, modularity transferability and other software architecture drivers became more important. Processing and fusion of huge amount of data from sensors like cameras, lidar, ultrasound, etc. got implemented. Compared to signal-based deeply embedded algorithms, software engineers with strong computer science skills and backgrounds are required. Therefore, shifting deeply embedded developers to vehicle computer development is not as easy as many managers think. An analogy would be to shift an experienced digital electronic developer to become a power electronic developer.
A third software category in the context of the Software-defined Vehicle is cloud services. Even though, this type of software is not part of the onboard software, it plays a significant role with the SdV. Services in the cloud enable vehicle features based on offboard data from many other sources. For this reason, connectivity became a key in the automotive domain. As a trade-off, cyber security challenges increased heavily.
In many non-automotive domains, cloud services are well established for many years. In this area Continuous Integration / Continuous Delivery (CI/CD), i.e. DevOps approaches have been state-of-the-art for a long time.
Therefore, µ-Processor based development paradigms from outside automotive entered automotive engineering practices.
Through the infotainment domain, another software category entered vehicles. Today many infotainment systems are based on consumer electronic software stacks, especially Android and iOS. These stacks come together with an entire development ecosystem and dedicated Software Development Kits (SDK). The way of designing and implementing apps completely differs from the engineering approaches with earlier described categories.
Finally, this leads to a fifth software category (see Fig. 3): software development tools. For each of the described

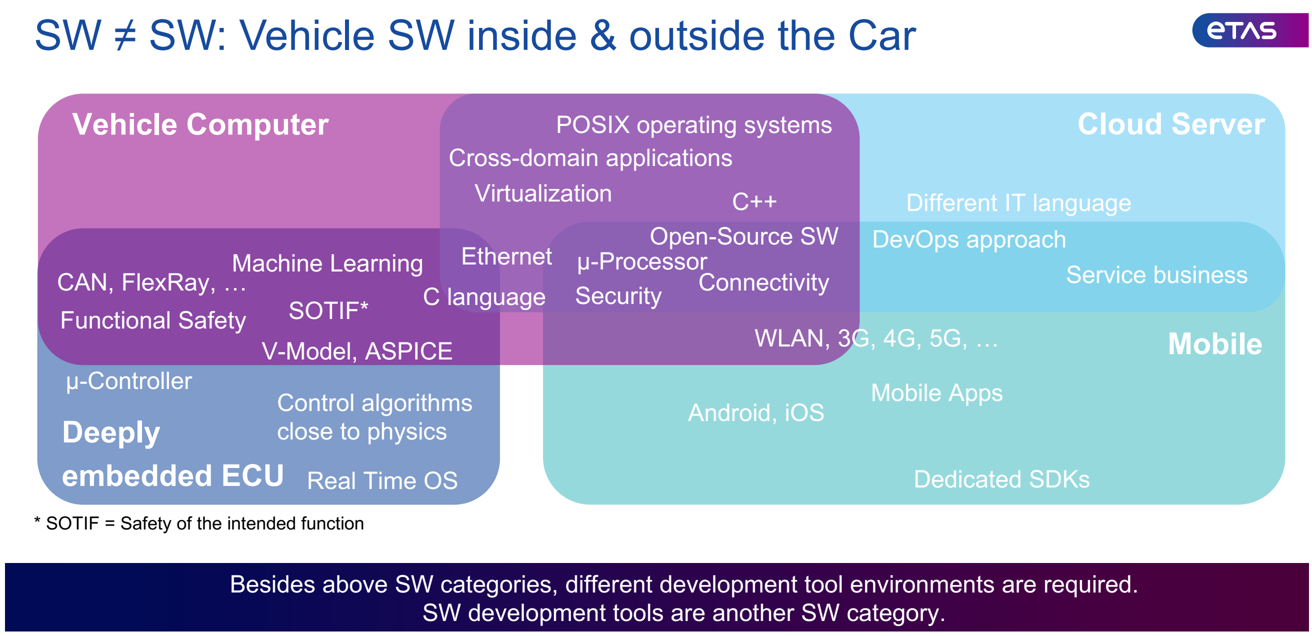

Fig. 3: Software is unequal software.

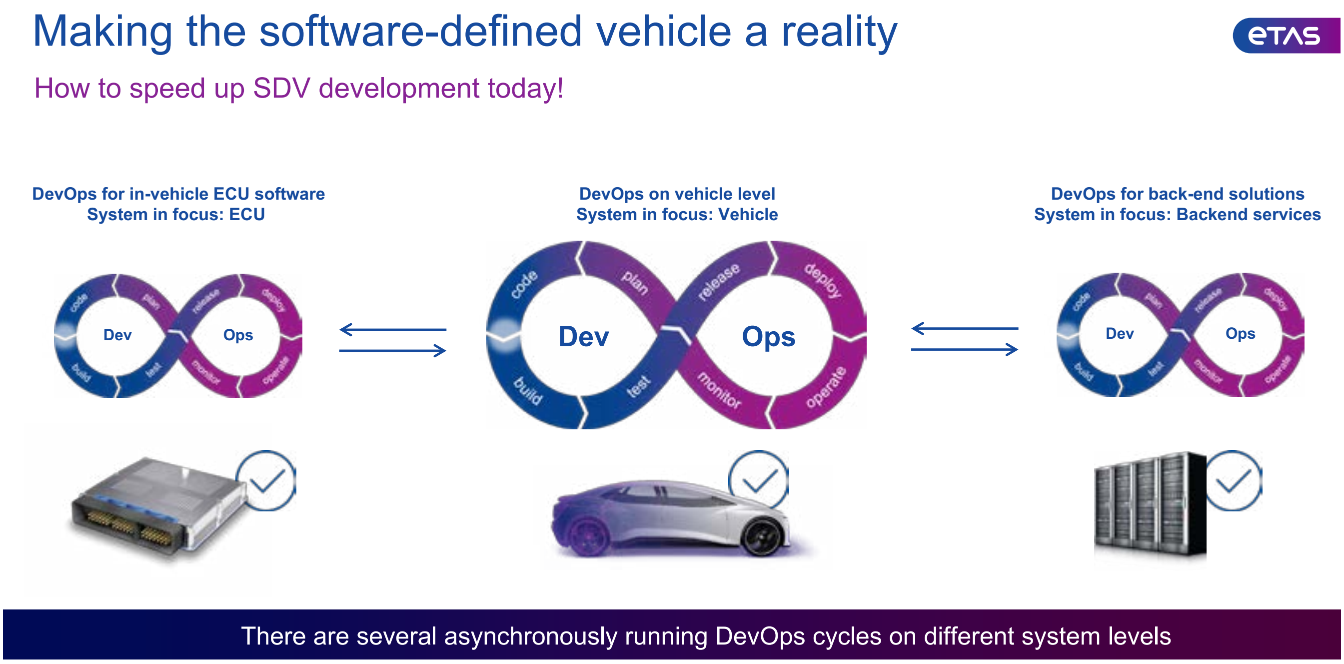

Fig. 4: Different DevOps cycles with different speed.

software categories, different types of software development tools are required. Special tool companies provide automotive- specific engineering tools, like measurement and calibration tools, code generators, model-based development tools, virtualization tools, testing tools, and many more. Without this type of software, the development of the Software- defined Vehicle would not be possible.

Software Stacks and Ownership

In the previous section, we described the difference of software categories in the automotive domain. A closer look at the software stacks and providers of stack components, significant changes can be seen. In deeply embedded ECUs the software stack consists of real-time operating systems provided by Tier2 like ETAS. The next stack layer, often called basic software, are AUTOSAR Classic stacks, again provided by Tier2 like ETAS, Vector, Elektrobit, etc. The topmost layer is the application software typically provided by Tier1, which integrates the entire software stack in an automotive-grade manner. Of course, software parts from OEM and other suppliers can be integrated as well. In general, the big Tier1 together with their Tier2 daughter companies are capable of providing and completely developing the entire software stack. So, the development depth of the deeply embedded ECUs is up to 100% with the big Tiers.
In the µ-Processor based domain, the situation changes dramatically. 3rd parties provide POSIX operating systems (e.g. Linux-based operating systems) and hypervisors, which are even not coming from the automotive domain.
The next layer adds automotive-specific, application-independent features and services. This middleware layer can be AUTOSAR Adaptive stacks [7] and/or application-specific middleware layers, e.g. for automated driving.

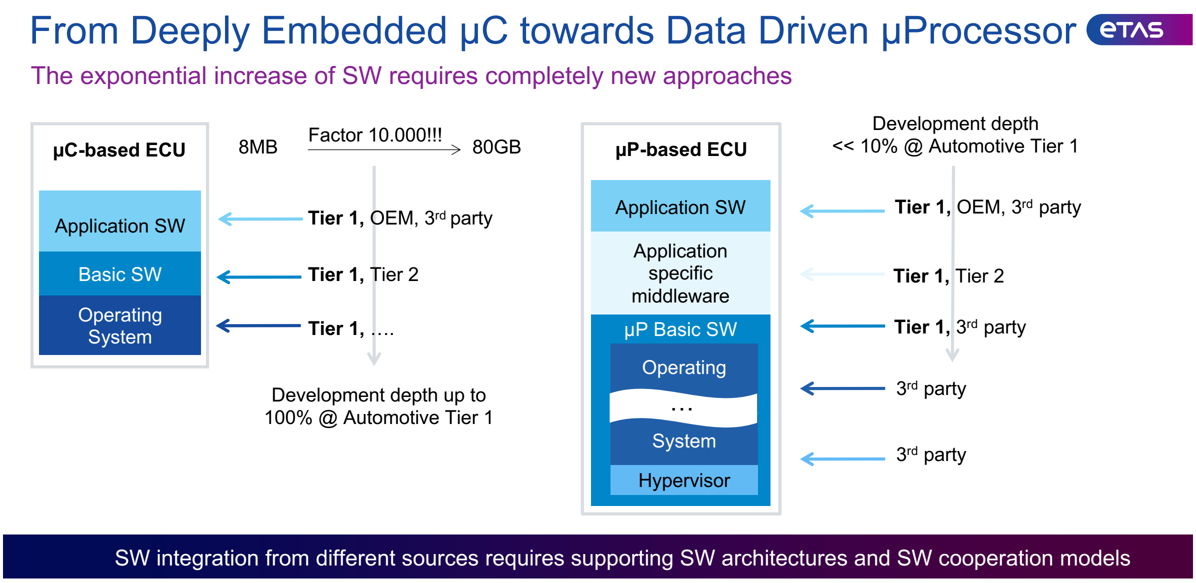

Fig. 5: Comparison between µC-based versus µP-based SW stacks.

The topmost layer hosts application software. Due to decoupling from the underlying hardware, algorithms, libraries, or even complete application packages can be shifted much easier to the next hardware generations compared to the deeply embedded domain. Software development becomes open for many software suppliers for dedicated ECUs.
Consequently, the development depth of Tiers for µ-Processor based software becomes less. On the other hand, the amount of software to be implemented and integrated increases disproportionately.
So, there is not a lack of software to be developed, but software and its integration into vehicle computers becomes a stand-alone business. Software and services become products, that are not provided on cost-bases anymore, but value-based pricing.

Conway's Law hits the Industry

With high-performance vehicle computers software features from different application domains can be shifted to centralized vehicle computers. Since the domain-specific applications are not necessarily allocated to a domain-specific ECU anymore, there is an impact on engineering organizations in the automotive industry. Earlier all OEM and Tier were organized based on application domains, i.e. powertrain, chassis, body electronics and infotainment. The organizational setup worked over two decades very well. Due to the decoupling of software from hardware and newly established centralized E/E architectures, the earlier organizational structures do not serve development needs anymore. According to Conway's Law [1, 2], saying that, organizations can only develop solutions which fits to its communication / organizational structure, and centralized E/E architecture breaks the well- established application domain companies' organizations. Consequently, automotive companies undergo disruptive organizational changes, leading to high efforts for transformation programs.

Collaboration models and Open Source

Another disruptive aspect comes with the emancipation of the software from hardware. Modern software product development comes with a completely different leadership approach compared to the classical automotive leadership approach. Not a single company can own the entire software stack anymore. Therefore, partnerships and other collaboration models need to be established.
Open source becomes a key aspect in the automotive domain. The development of open source significantly differs from classical automotive software. Instead of specification-driven development with given milestones and deliveries, open-source projects follow a "code-first" approach. That means, that open-source projects focus on implementation to solve a recent problem. Developers decide what is getting implemented within the wide range of existing open-source projects. There is no management that controls and monitors the project's progress. Of course, companies can influence implementations of open-source projects by contributions. This can be achieved by spending engineering efforts or even contracting engineering companies to contribute based on requests of the ordering company.
For the automotive industry, open source offers a wide range of chances, but also some challenges. Depending on the type of software (compare Section 3) one must decide, which software category should utilize open-source and which not. E.g. in the deeply embedded ECU software with high functional safety requirements, a deny strategy is common practice. It is important to note, that even in this case a solid process of open-source scanning needs to be in place, to ensure that no open-source (also no open-source snippets from developer forums) are included in the product.
In µ-Processor based software open-source plays an increasing role. Especially the underlying POSIX operating systems come with a huge amount of open-source inside.
In a next step, automotive companies use open- source com-

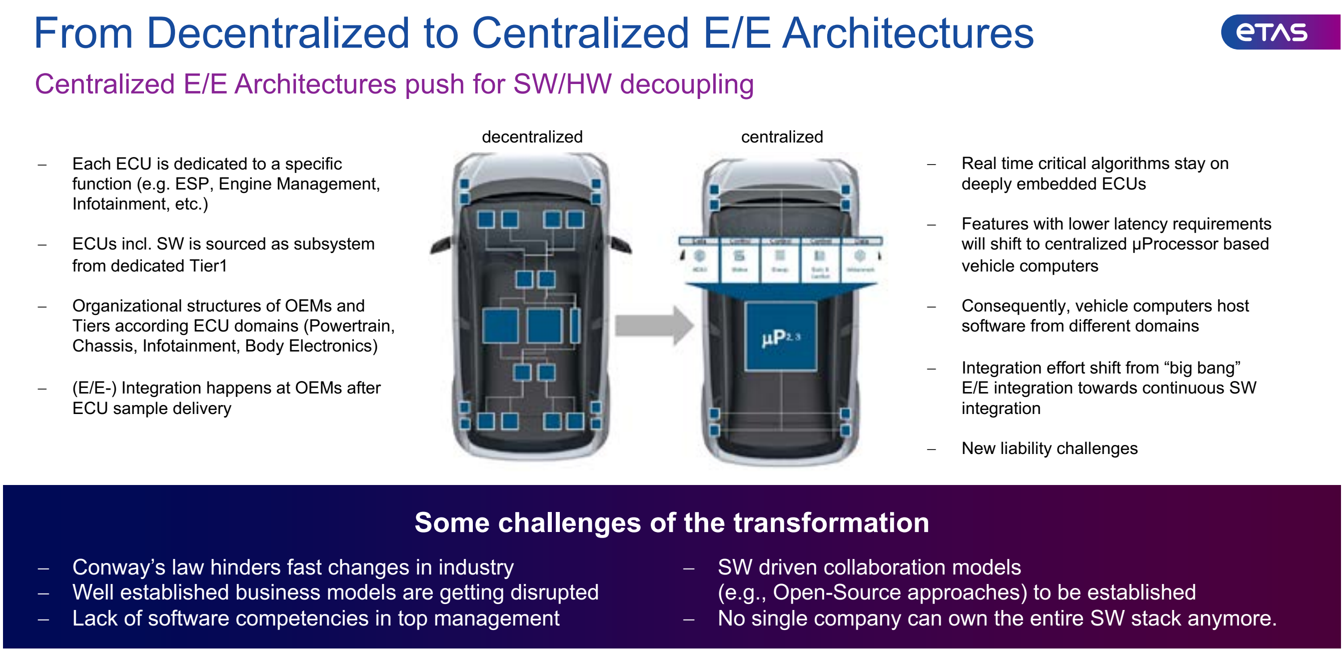

Fig. 6: Centralization of E/E architectures leads towards SW-decoupling from HW.

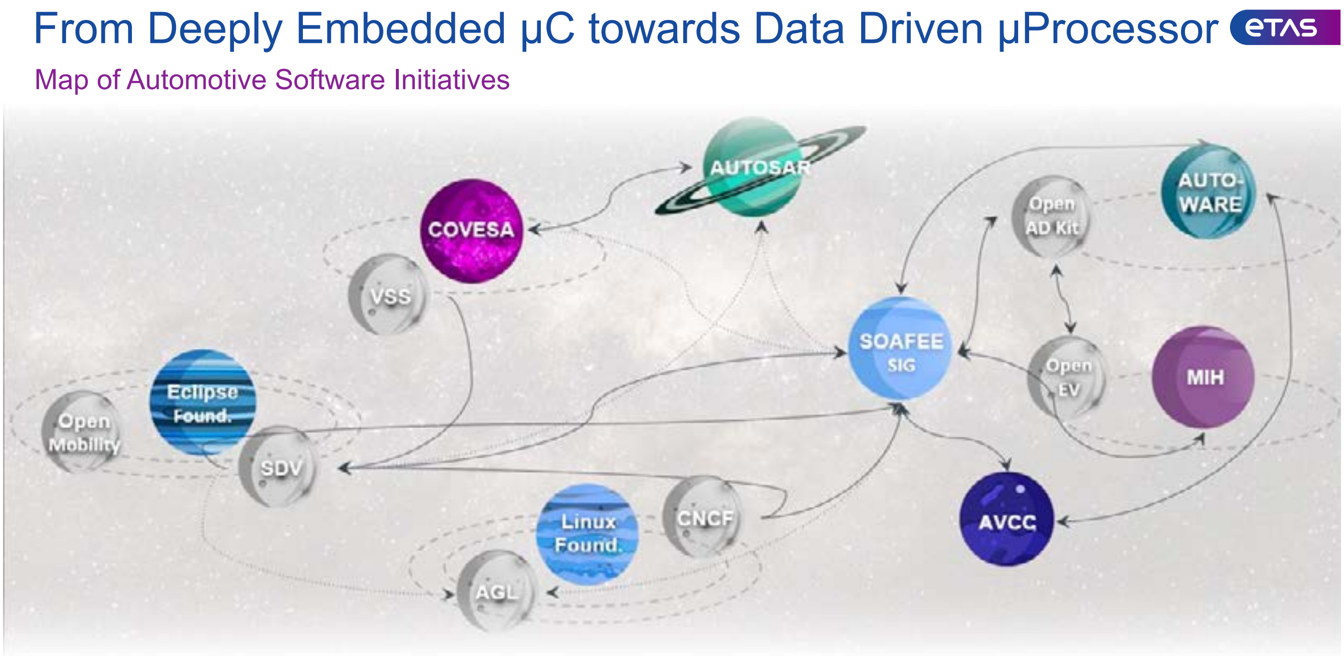

Fig. 7: The "Universe" of different open-source activities (and beyond).

ponents within their own development. This open-source "use approach" is mainly driven by engineers to speed up the development and avoid re-inventing the wheel. In this situation, the organization already needs higher open-source skills. With the usage of open-source contractual aspects become relevant. With the use of an open-source component, the corresponding license terms need to be fulfilled. There are many different versions of open-source licenses and the consequences for the own product need to be understood. Typically, open- source license terms need to be delivered with your software solution. E.g. licenses with "Copyleft" clauses come with a high risk. An open-source component with a Copyleft license enforces, that the software containing the component, needs to be placed under the same open-source license, which means, that your own component needs to be put under open-source, which might not be in favor of your business. Besides this obvious risk, there are some other aspects to be considered. There might be incompatible open-source licenses. Assume you want to include two open-source components coming with different open-source licenses. If your development uses exactly one of both, your product is fine. But if you are using both open-source components the incompatibility of both licenses might lead to an illegal product on your side.

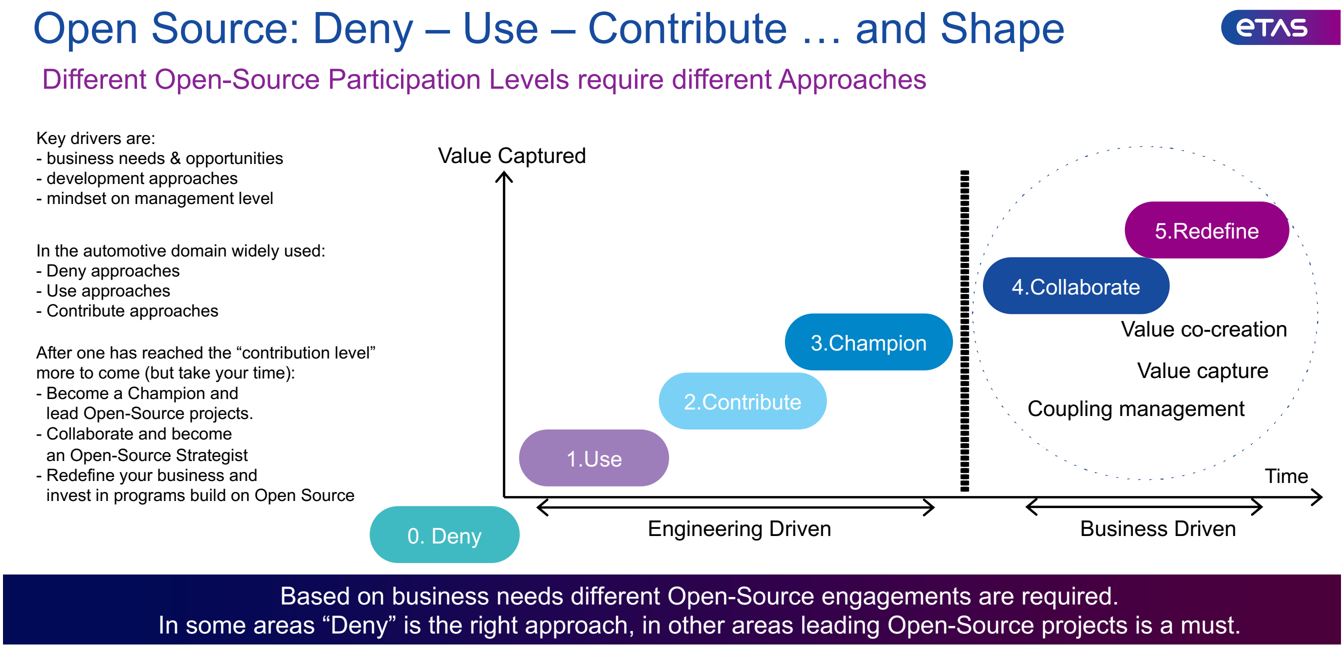

Fig. 8: The open-source engagement model.

Fig. 9: With connectivity, cyber security becomes an increasing challenge.

Therefore, using open-source requires dedicated open-source skills and proper open-source scanning to extract the list of used open-source licenses, to ensure the fulfillment of license terms, to identify risky licenses and to avoid conflicting licenses within your own product. If this process is well managed, open-source contributes to the business. Development costs can be reduced, but as already mentioned, open-source does not come for free. Bugfixes or feature enrichment of the open-source component might be required. This can be done outside the open-source project within your own development. But this comes with a big disadvantage – with each new version of the open-source component, the bugfixes and additional features need to be locally reimplemented. In this situation, contributions are the right approach. Instead of doing local modifications again and again, it is much more efficient to contribute bugfixes and feature extensions directly back to the open-source project as well. Afterwards, the modifications are included in all upcoming versions of the open-source component. It is important to note, that contributions to open-source projects follow a specific process. Contributions need to be accepted by the community and maintainers of open-source projects. Consequently, contributors need to earn reputations within open-source projects, which can only be achieved by spending significant effort within open-source communities. Often this might be perceived as unnecessary development costs, but appropriately established finally saves development costs.

Beyond what has been said so far, the utilization of open-

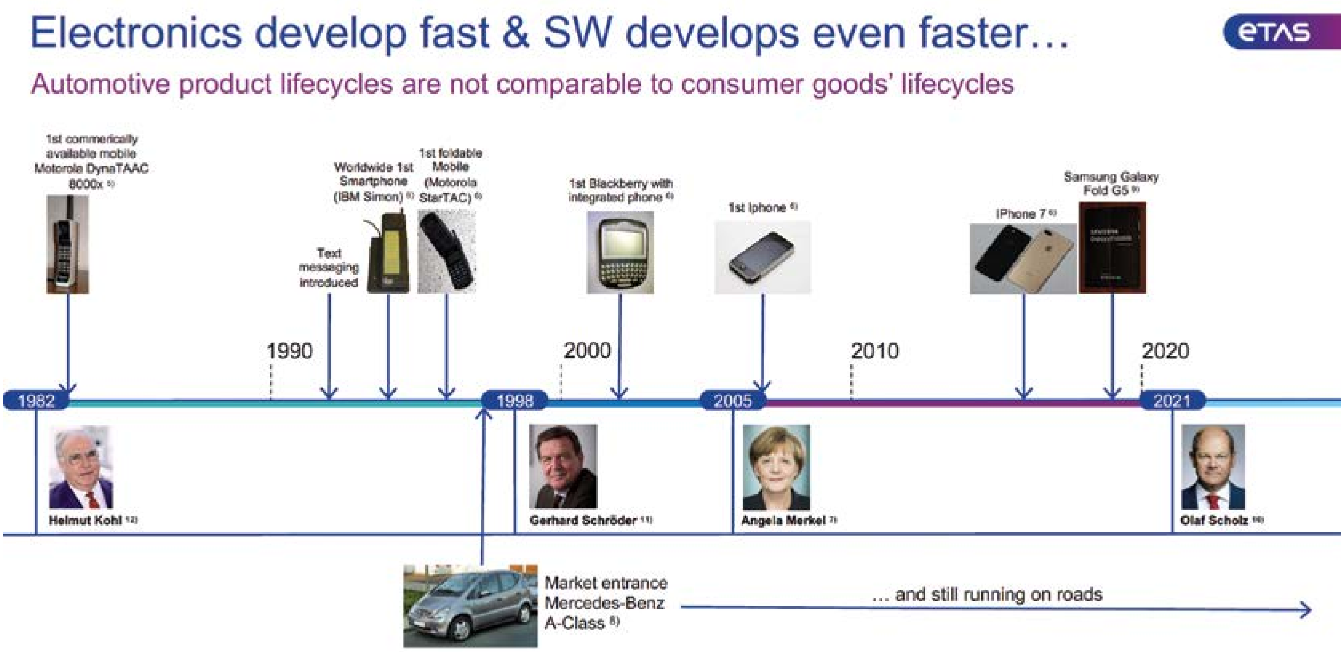

Fig. 10: Product life cycles compared between mobile industry, automotive (and politics).

source for business reasons can be increased further. Instead of contributing only, companies might start to shape things in open- source, i.e. defacto-standards. Becoming an open-source champion by leading entire open- source projects, can retain recently and enable new businesses. One key factor for success is, that the open-source project is addressing problems, which are relevant for a wider range of companies. Only then, the project will become a vibrant open-source project. Since automotive software development faces different requirements compared to e.g., consumer electronics (especially the mobile industry), an automotive open-source eco-system will look different. One big challenge are long product life cycles (up to or even beyond 20 years). Due to continuously increasing cyber security challenges, automotive software needs to be maintained over the entire product life cycle. In consumer electronics (esp. mobile business) product life cycles of some few years are common. In case a mobile device is used beyond available software maintenance, the user is not facing functional safety-related risk, even if his device has been attacked by a hacker. For vehicles on the road, the situation looks different.

Maintaining automotive software over such long product life cycles will become more and more difficult with proprietary software stack. For proprietary software stack components companies need to reserve entire software maintenance teams over product life cycles. These teams won't be available for the development of new innovative features and will put a cost burden on the company. Therefore, companies utilizing open-source approaches will gain a significant cost benefit by reduced maintenance efforts.

As already earlier mentioned, open-source projects cannot be controlled like former automotive software projects. Contributors and maintainers of open-source projects influence and control their open-source projects. If automotive companies would like to influence open-source projects significant contributions are required. To overcome the initial barriers towards a cross-industry, open-source-based automotive software stack, the European Commission as well as national authorities initiated public funded projects in this domain. Since 2024 several SDV public funded projects have started, most of them in open-source, ensuring that there are no anti- trust risks with the collaboration across the industry. Prominent examples of cross-industry projects are:

- FEDERATE [3]: A Coordinating and Support Activity (CSA)) for EU-funded projects for the Software-defined Vehicle.
- HAL4SDV [4]: Hardware Abstraction Layer for SDV
- SHIFT2SDV [5]: Addressing a higher layer in the software stack, just below differentiating application software.
- Eclipse SDV [6]: SDV related projects: A set of automotive related open-source projects under the umbrella of the Eclipse Foundation
  - Eclipse Score [7]: An open-source stack focusing on a pre-integrated software stack that shall become a basis for commercially available automotive stack through distributors.

The open-source approach with public funded projects gives another benefit: All project results can immediately be used within series projects. Of course, there is still a problem to be solved. Especially, the project Eclipse Score addresses a crucial requirement within the automotive domain: How can open-source projects fulfill process-related requirements considering ISO 26262 (Functional Safety)? In this context, discussions are ongoing on how to ensure automotive-grade within open-source projects. On the other hand, discussions are also ongoing with the AUTOSAR consortium, how the AUTOSAR Adaptive standard can become more open for people or organizations that are not members of AUTOSAR.

Conclusion

This paper has provided an overview of different software categories in the automotive industry. The disruptive transformation based on the advent of the software-defined vehicle has been described. One key approach to address the challenges with the increasing amount of automotive software is the utilization and shaping of open-source ●

References

[1] Conway, Melvin E. (1968): How do com- mittees invent? http://www.melconway.com/Home/Conw ays_Law.html

[2] Zerfowski, Detlef and S K, Niranjan: "(R)Evolution der Automotive-Software- Architekturen. Wie neue Software- Technologien die Automobilindustrie verändern." Proceedings Embedded Software Engineering Congress 2017, Sindelfingen, 4th-8th December 2017, pages 411-424.

[3] FEDERATE: European Software-Defined Vehicle of the Future (SDVoF) Initiative https://federate-sdv.eu

[4] HAL4SDV: Hardware Abstraction Layer for a European Software Defined Vehicle Approach https://www.hal4sdv.eu

[5] SHIFT2SDV: Webpage yet to com.

[6] Eclipse SDV: https://sdv.eclipse.org

[7] Eclipse SCORE: Eclipse Safe open Vehicle Core https://eclipse-score.github.io

[8] AUTOSAR Adaptive Platform: https://www.autosar.org/standards/adaptive-platform

[9] EU Chips JU - https://www.chips-ju.europa.eu/Projects

Resources

Fig. 1: https://thebulletin.org/doomsday-clock/timeline

Fig. 2:
1) By Luitold-Eigenes Werk, CC BY-SA3.0, https:/commons.wikimedia.org/w/index.php?curid= 19429338
2) https://www.bosch-presse.de/pressportal/de/de/40-jahre-antiblockiersystem-abs-von-bosch-169700.html
3) By Cschirp, CC BY-SA 3.0 de, https://commons.wikimedia.org/w/index.php?curid= 16709605
4) By Granada - Eigenes Werk, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid= 70384434

Fig. 10:
5) Motorola DynaTAC - Wikipedia https://commons.wikimedia.org/w/index.php?curid= 6421950
6) Wikipedia
7) https://assets.weforum.org/sf_account/image/-Iz2VwsxPVkx3GQrc5m3oWD1mI2d4yD2liRq60jnV 04.jpg
8) https://commons.wikimedia.org/wiki/File:Mercedes_A_160_Elegance_(W168) front_20090926.jpg
9) Samsung Galaxy Fold – Wikipedia
10) BMF/Thomas Koehler/photothek.net
11) https://www.gspeakers.com/wp-content/uploads/2021/01/schroeder_gerhard.jpg
12) https://i.pinimg.com/originals/14/aa/2a/14aa2a0bb4aa1b713f7bfd9b0bc02ea1.jpg

Fig. 8: Based on "Open-Source Engagement Model", © 2007 Peter Carbone; https://timreview.ca/article/93

Glossary

SDV: Software defined Vehicle

POSIX: Portable Operating System Interface: A set of IEEE standards defining Application Programming Interface (API) for POSIX compatible operating systems.