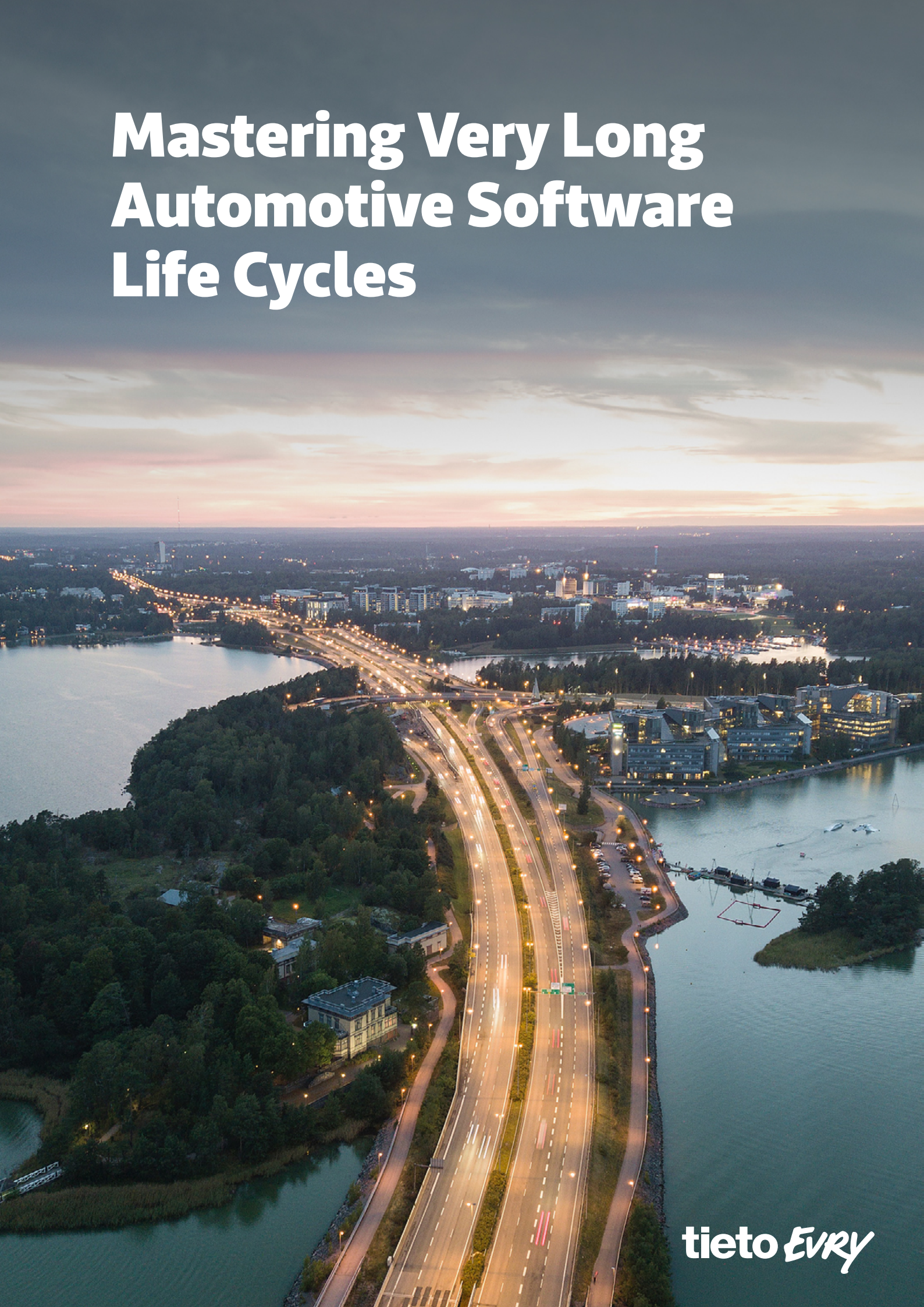
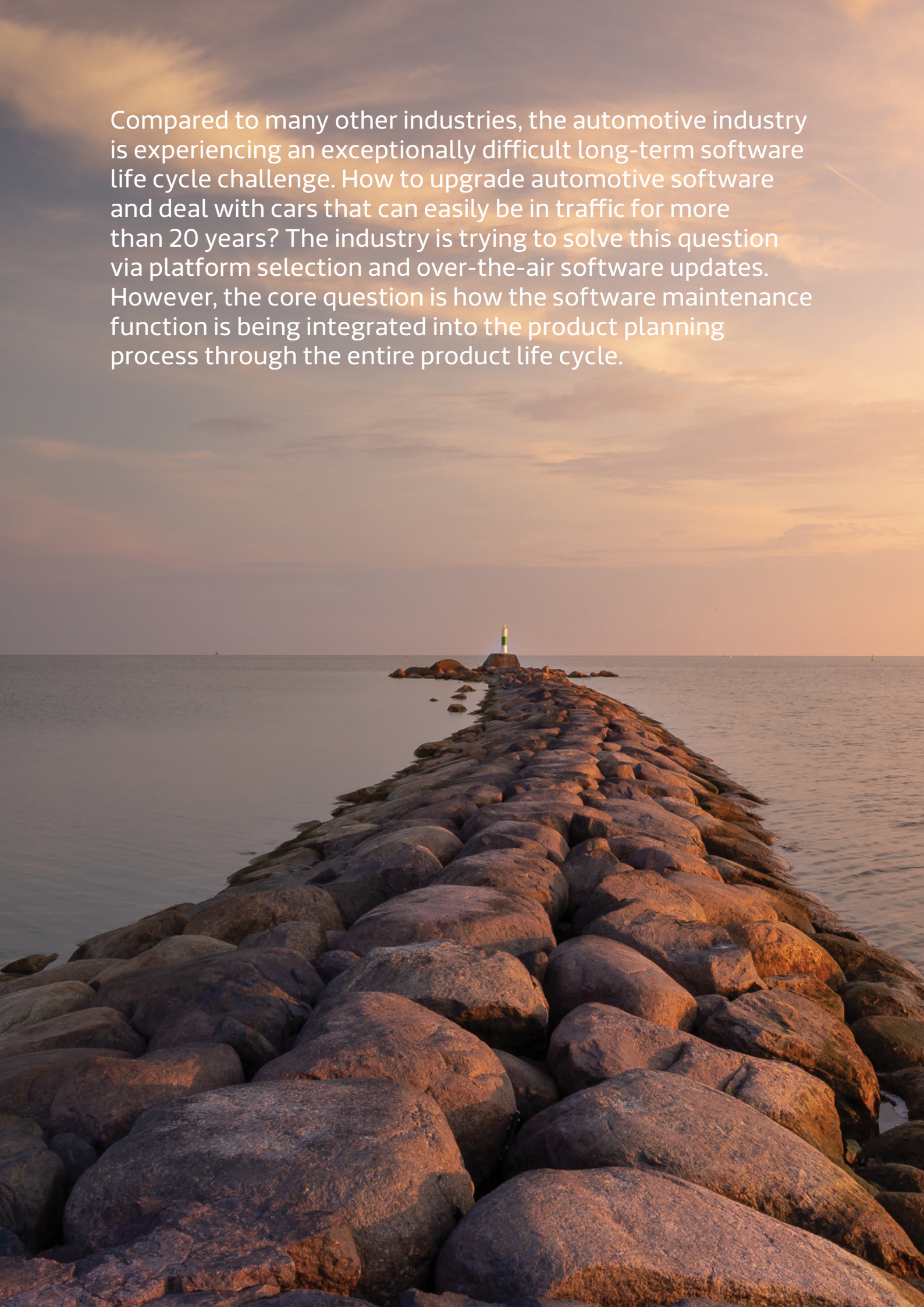
Mastering Very Long Automotive Software Life Cycles



tieto EVRY

Compared to many other industries, the automotive industry is experiencing an exceptionally difficult long-term software life cycle challenge. How to upgrade automotive software and deal with cars that can easily be in traffic for more than 20 years? The industry is trying to solve this question via platform selection and over-the-air software updates. However, the core question is how the software maintenance function is being integrated into the product planning process through the entire product life cycle.



Introduction

This whitepaper covers various aspects of the modern automotive software licecycle management and also provides five main factors to consider as a resolution.

This paper is especially useful for Automotive OEMs and their suppliers developing the next generation's increasingly feature-rich and connected automotive software systems while carrying the responsibility for keeping the vehicles safe for their users.

Everyone planning the software product development and life cycle management, selecting development practices and tools, designing the software architecture or managing the procurement of external software components (or support software for HW components), for example, is affected by the challenges of very long-term software maintenance.

About the writer

Markku Tamski, Lead Software Architect at TietoEVRY, has solid long-term experience in software and product development covering automotive as well as smart devices and connected services. His career includes working for 15+ years on leading edge mobile device and connected service software technologies, and for the past 6+ years with automotive software and product development. During this time, he has worked in varied roles, such as crafting SW development processes and architectures, and leading development projects, software builds, releases and deliveries, as well as product and manufacturing quality assurance activities.

TietoEVRY has decades of experience in acting as a strategic software R&D partner in the telecom and smartphone area. The learnings are resonating well into our automotive R&D partnerships.

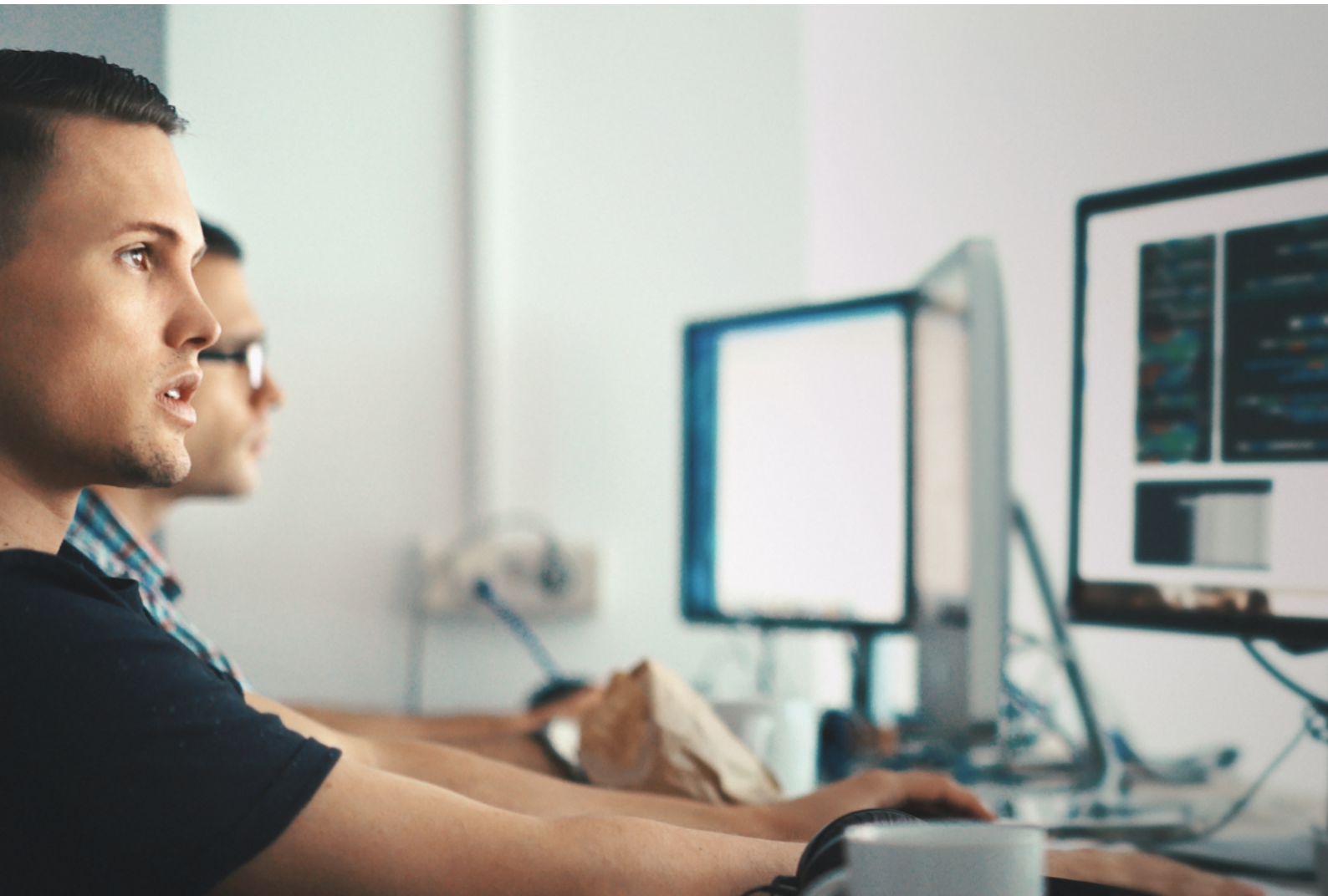
The development speed of Automotive Software should be increased urgently

Software is commonly acknowledged as a major enabler of substantial changes in the automotive industry, such as digitalization, connectivity, electrification, mobility services and ADAS/AD systems. Which leads to:

“OEMs must deal with the exponential increase in software content while attempting to reach software-development speeds typical of digital-native companies.” [1]

What are the methods “digital-native companies” have used to reach their speed of development in increasingly complex software systems?

Some of the methods have been **process-related**, such as moving towards a more agile way of working in software development, but that is only one part of the story. Other important factors have been **re-using and building on top of existing**, known technologies, components and code, as well as limiting the support time-window where updates are provided.



Why not just copy development practises from the software-driven, established industries?

There are several differences, both bigger and smaller between the automotive industry and the industries that have driven the software evolution. Some of the largest and most consequential of these differences are **the typical maintenance period** (product longevity/service life) and the **responsibilities** of product and system vendors.

Automotive Software Lifespan

From the first car being sold to the end of the service life for the last one with the same software systems, the lifespan of a car model can easily be 15-25 years. This is very different from the industries that work at much higher clock speeds.

The product/version lifespans of some of the operating systems behind the products in the "digital-native" industries highlight the problem:

- Android was not yet released 15 years ago. The oldest currently supported version is from 2016.
- Windows XP was in use 15 years ago. Even extended support ended in 2014.
- Linux kernel's longest long-term support release was 2.6.32, supported 2009 – 2016.

Even though there are some initiatives to provide very or super long-term support versions of, for example, Linux kernel for infrastructure use, it's clear that none of these commonly used platforms will solve the problem for automotive any time soon. **Automotive is at the forefront of figuring out very long-term software maintenance challenges for connected, always up-to-date software systems and general use operating systems specifically.**

Safety-critical and secure automotive software responsibility

When automotive safety is in the news for safety-related recalls, the reason is a system that doesn't function as expected in certain conditions. It doesn't matter if the root cause is hardware or software. There is no exception for software related issues when it comes to responsibility for vehicle safety.

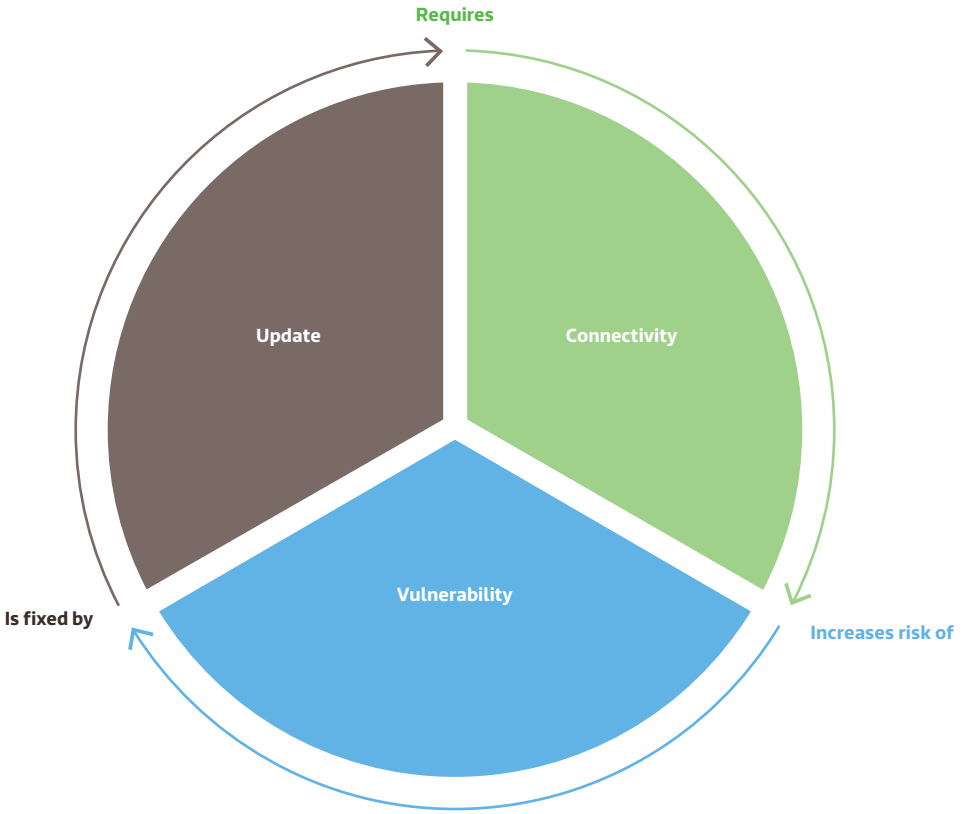
As an example, security researchers identified and published a hack [2] on a particular Jeep model that allowed them to remotely control any of those vehicles, thus making all of them unsafe. The vehicle owners started a class-action lawsuit against FCA, the manufacturers of Jeep. FCA's argument against the lawsuit wasn't that safety responsibility does not cover software, but that none of the vehicle owners were directly affected by the hack, and that the hacked systems were promptly updated with a fix before anyone was affected.

In practice, OEMs and system vendors must be able to show that they have taken reasonable steps to keep the vehicle safe also in terms of software issues..

Key factors for the shift in the software maintenance

Looking at traditional products' development life cycle in different industries, the maintenance phase is often presented as the final phase of the system development. The primary responsibility of maintenance has been to make sure that the product is safe to use and works as originally intended until the end of its lifespan. Even the latter part has been the vendor's responsibility for a limited time, and then the costs of maintenance shift to the customer. If the product changes very slowly or not at all, the number of safety-related issues identified and fixed should be declining over time. This is no different for software products with c omparably limited exposure to outside influences.

Modern complex software products, automotive included, are no longer like this. Vehicles are connected to a larger ecosystem, with features being added, removed or rendered useless by changes in the external systems required for the features to function. Updates are a fact of life, and not a major cost issue to be avoided if possible. On the other hand, connectivity brings new security challenges. Vehicles are a constant target for hacking attempts and finding vulnerabilities, as physical access to the vehicle is no longer required.



The impact of security and safety demand to the total cost of software ownership (TCO)

In this new world of connected and always up-to-date vehicle software, the need and cost for maintenance will no longer decrease over time. Priority will shift from identifying passive flaws in the system to preventing active hacking attempts, to make sure that the vehicle is secure and safe.

Older vehicles are especially interesting for finding vulnerabilities, so the model of a few years of security updates used by some of the previously mentioned “digital-native” industries won’t be enough. Any vulnerability may become the reason for the next class action lawsuit if it hasn’t been taken seriously and it leads to an actual hack and danger to the customers/owners of the vehicles. Not to mention the potential for lost sales due to the vehicle being labelled unsafe.

In principle, security vulnerabilities are just like any other critical issues. The number of vulnerabilities should decline over time if nothing else changes. However, any change, such as the need for introducing new features and changes to the connected, external systems can “reset the clock” when it comes to bugs and security issues. Additionally, new techniques are constantly being studied in the cybersecurity area, so systems that were once secure may not stay that way even when not changed. “Row hammer” is an example of such new generic technique allowing development of hacks across a wide variety of existing systems [3].

One type of vulnerability would be an external service being stopped after a number of years. Any component left to the vehicle side could present a security risk, if it wasn't removed, isolated or otherwise kept secure, even when the server-side functionality was removed. An example of an attack using "abandoned" code is "SimJacker": It's an attack against the phones caused by a vulnerability in the Sim Toolkit feature used by mobile operators. The specification of the affected component hasn't been updated since 2009, but according to the researchers, the component is still used in SIM cards of a billion people in 30 countries.

We can see from other software-driven industries that the maintenance costs can be, for example, from 60% [3] to 90% [4] [5] of the software development cost total, or that by the fifth year of maintenance the total costs can equal the development costs [6]. The maintenance costs, in general, have increased considerably over the last decades [4], with the increased complexity of the software system and other factors.

An important cost trend specifically relevant for the automotive industry and very long-term support as identified in the studies is that the yearly maintenance cost usually goes up, and the speed of change accelerates, over the product lifetime [6]. This is easy to understand as an inevitable effect caused by the use of external components, such as operating systems, components, development tools and various services. As their own development progresses, the deviation between the product in maintenance and these components grows larger. At some point, fixes are no longer available for the old versions, and either an upgrade to a newer version or taking over making the fixes for the older version is required. The longer the latter track continues, the less it benefits any new product development. At some point the changes required by the software stack may even start obsoleting the hardware, lacking support for all available HW features.

There's little to suggest that the same findings would not apply to automotive software. On the contrary, they are reinforced by the specifics of the automotive industry:

- Very long-term maintenance period leading to the even bigger delta between the old and new
- Responsibility to keep the vehicle safe all through its lifespan
- Vehicle and its software fulfilling any required certifications at all times
- Strict processes that need to be followed for each change made
- Challenges of managing a very large software product base of often privately-owned vehicles in various countries with no guaranteed network access

What should Automotive companies do to cope with the long-term software maintenance challenge?

Automotive OEMs and all the players in the ecosystem should start acting accord the long-term maintenance needs. With connected, always up-to-date software products, maintenance is no longer a single separate stage happening in isolation after development. **The maintenance is on-going product development, sharing the same requirements.**

Despite the unique additional requirements in automotive, there are good lessons to learn from consumer electronics, MedTech and telecom industries that have already made the transformation from hardware to software-driven.

Combining experience from all of these industries, having automotive insights in mind, there are five factors to remember:

- **Treat maintenance as part of the product planning from start to end**
- **Ensure that system architecture enables updates and maintenance**
- **Use codeline management practices and DevOps processes**
- **Use software suppliers and partners cleverly to manage the product life cycle**
- **Manage hardware dependencies**

Let's now take a look at each separately:

1. Treat maintenance as part of the product planning from start to end

Maintenance as the last stage of the product life cycle separated from the development stages, will not carry on very long when supporting connected, safety-critical products. Instead, maintenance should be seen as part of the development cycle. It will need to adhere to all the same standards and have access to all the same data and tools as original development.

A well-defined maintenance strategy needs to be created early in the project and kept up-to-date throughout the product support period. Software maintenance strategy for complex software components may involve, e.g. decisions on long-term support versions and avoiding forking the mainline versions. Often the maintenance strategy will need to consider multiple maintenance stages over the product support period. For example, a software component from an external provider may receive direct support for a set period, then a transition to creating fixes by backporting from future versions and, eventually, require fully internal ownership of the product software development.

Maintenance is development also in terms of the product requirements, processes and commitments. The updated product needs to be just as safe, fulfil the product-specific certifications and have a feature set, where planned external services work and any obsolete components are removed or otherwise secured. If the focus is on the immediate needs and latest development tools and methods, testing, and project documentation, rather than considering their availability in 10-15 years, it is very easy to create issues that only surface later. Even a commonly agreed upon good practice like “continuous improvement” can become the enemy if not equally considered in the very long term.

2. Ensure that system architecture enables updates and maintenance

Maintenance has a role at the product planning, but it also needs to be part of the product design in system architecture work. Products needing very long-term support benefit the most from good, solid architecture decisions and well-known and understood solutions.

Planning around common computing platforms and applications (preferably with long-term support plans) instead of extremely distributed ECU-per-functionality architecture gives product development and maintenance a solid foundation. They’ve been there before and many solutions have stood the test of time, unlike in-house solutions still learning to fly.

As the complex software system needs to be inherently updatable, modularity is a benefit. Modularity can take many forms, such as containers, stable known-good network technologies and well-managed APIs. The priority needs to be with the capability to modify, test, update and even remove specific components with as little impact to other parts of the systems. Specifically, removal of features from a product in use is one specific area that hasn’t been at the forefront of architecture work before vulnerabilities have been identified in consumer electronics with the proliferation of always-connected, complex software ecosystems. Again, modularization with known security methods can help manage the impact of such changes to the system.

Architecture should also define minimal systems for critical core features, such as OTA updates, to minimize the potential for issues in these areas. Less critical components, such as infotainment applications can be updated quite often, but the more critical the components are for the product’s security and continuing functionality, the less need there should be for updating them. The most critical components should be the most hardened and well-insulated from the other systems.

Yet another critical, over-arching part of the architecture work for such a system is vulnerability management. Vulnerabilities need to be tracked, fixed and the product software updated continuously throughout the product support period to keep the product secure and safe. This is the minimum level for showing that the manufacturer has taken safety issues seriously. Found, reported but not fixed vulnerabilities are a major risk if a safety-related hack using that vulnerability is devised. A pre-emptive method for preventing any such vulnerability from becoming a hack and a safety concern goes, again, back to solid, reasonable architecture decisions. Known good security solutions should be used to limit the effects of any vulnerability. For example, modularization via IP networking and use of IP firewalls.

3. Use codeline management practices and DevOps processes

As a maintenance period with updates actually extends the development and releasing period, the source code management and other DevOps processes need to be extended to cover the whole product lifetime. The more agile the processes are during the initial product development, with continuous integration and delivery practices, the easier it becomes to manage the increased number of releases and branches over the years of support.

Version control with branching strategy, and configuration and release management are found at the core of good, long-term practices for staying on top of the releases. Traceability in both directions, both from product software versions to code and on to requirements, and from requirements to code and on to product software version, is mandatory for finding the root causes of faults and following development and releasing quality processes.

Product code ownership for all code used, including dependencies all the way to the end of the dependency chains, must be managed, so that making new, fixed versions are not dependent on any external party. This is surprisingly difficult, especially when it comes to various tools used, and any dependencies they may have.

The emergence of cloud-based development and work management tools brings in yet another issue to manage: how to continue development according to existing processes and available data, once the cloud-based tools are no longer available or compatible? It's best to have a local solution available at that time, which actually means "from the start of the project". No one can guarantee a transition period in case, for example, of a bankruptcy.

4. Use software suppliers and partners cleverly to manage the product life cycle

The importance of knowing what to do yourself and where to use suppliers and partners instead is one of the core aspects of delivering and maintaining complex software products. Software evaluation and procurement should always consider the options for very long-term maintenance. This is obvious when software components and platforms are attained as commercial products, but often there are options also for Open Source software maintenance. For example, procuring Linux kernel and defined platform/distribution creation and maintenance responsibility as a service from a supplier with suitable Linux expertise may free internal resources to core product development from generic platform maintenance.

Another area where clever use of partners and suppliers may lower the barrier and number of issues faced significantly is the processes and work practices related to maintenance. Experience in backporting fixes, upgrading to newer versions and other such tasks combined with a good understanding of product software version, branch, configuration and release management help significantly and they don't need to be created from scratch. However, using existing knowledge to kickstart one's own maintenance activity does require a willingness to adapt the existing ways of working as well.

5. Manage hardware dependencies

At some point in the support period and depending on the changes required in software and chosen maintenance strategy, the software may also start obsoleting the product hardware. This may be a result of critical issues that cannot be fixed without changes in the hardware, or just the removal of software support for any hardware features of the product. Replacing hardware is expensive, and rarely the first option, but in some cases, it might still be less expensive than the options, such as continuing development of the obsolete software versions with hardware support, if they are commercial binary components, for example, or buying back all the vehicles with the safety issue.

There are a few guidelines that can reduce the likelihood of hardware being obsoleted by software. Very long-term component availability is a common requirement for hardware designs, but this needs to be extended to cover also the availability of the required support software for that hardware. In practice, sourcing any hardware components that require driver software must take into account not just the current but also future compatibility with all relevant parts of the system, for example Linux kernel versions, over the whole planned maintenance period of the product.

Other areas that can benefit hardware maintenance in the same vein as software, are modularity and interface management and known good security solutions. Security solutions help to limit the effects of any identified vulnerabilities, making the need for HW changes less probable. Modularity in design may ease replacing as small of a part of the hardware system as possible and subsequently require as little software modifications as possible. However, it may also create unneeded barriers to managing and updating the system with software, and open up extra attack surfaces in the vehicle, so it's not always beneficial.

Conclusions

Software systems in the automotive industry are evolving to be more complex, connected and constantly updated members of a larger ecosystem. At the same time, development speed increases are being pursued with the use of modern software development practices test-driven in other industries. Combining these with the product liability requirements over the long product lifetimes inevitably leads the automotive industry to a new frontier without a readily available map.

The very long-term support challenges in software presented here have not yet been solved by other industries to the full extent needed by the automotive industry. However, the lessons in good practices for software system development planning and execution do provide a solid foundation to start solving these issues. The longer the maintenance needs, the more benefit adopting known good practices provide. Conversely, bad practices in early development may make piecing required data together, when needed, impossible.

The first guideline to start solving this problem is simple: Very long-term support activity – and security as one specific area for that – must be included in the product planning from start until the end and in each activity and design decision. Everyone involved must understand the chosen maintenance strategy, it's implications for their work and vice versa, and how the maintenance strategy may change over time, as the unexpected occurs. We can't expect to get everything right, as we really don't know for certain what "right" even is 10-15 years from now. It's still very easy to pull the rug from under a lot of good work by the simplest of omissions, which could have been prevented by making sure everyone sees also the world beyond the product launch.

References

[1] McKinsey Center for Future Mobility, "Automotive software and electrical/electronic architecture: Implications for OEMs," April 2019.

[2] A. Greenberg, "Wired," 21 July 2015. [Online]. Available: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. [Accessed 7 February 2020].

[3] R. D. Banker, S. M. Datar, C. F. Kemerer and D. Zweig, "Software complexity and maintenance costs," Communications of the ACM (Vol. 36, Issue 11.), 1993.

[4] S. M. H. Dehaghani and N. Hajrahimi, "Which Factors Affect Software Projects Maintenance Cost More?," Acta Inform Medica, vol. 21, no. 1, p. 63–66, 2013.

[5] J. Koskinen, 30 April 2015. [Online]. Available: https://wiki.uef.fi/download/attachments/38669960/SMCOSTS.pdf?version=2&modificationDate=1430408196000&api=v2. [Accessed 20 November 2019].

[6] C. Jones, "Chapter 23: Maintenance and Enhancement Estimating," in Estimating Software Costs, 2nd Edition, McGraw-Hill, 2007.

TietoEVRY creates digital advantage for businesses and society. We are a leading digital services and software company with local presence and global capabilities. Our Nordic values and heritage steer our success.

Headquartered in Finland, TietoEVRY employs around 24 000 experts globally. The company serves thousands of enterprise and public sector customers in more than 90 countries. TietoEVRY's annual turnover is approximately EUR 3 billion and its shares are listed on the NASDAQ in Helsinki and Stockholm as well as on the Oslo Børs. **www.tietoevry.com**

tieto EVRY