WHITE PAPER

Key Considerations for Redefining Automotive Software Development

Recommendations from field experience

MathWorks®

Introduction

The vehicles of tomorrow that promise to achieve the goal of shared carbon-neutral mobility are going to be autonomous, electric, connected - and software-defined. Organizations are investing in digital capabilities to make the shift from designing the best piece of machinery to providing state-of-the-art, end-to-end mobility and digital services. The vision is a user experience, which will largely be defined through software. To enable this vision, the industry is reshaping itself around software as a future value driver.

Traditionally, most automotive software has been delivered together with hardware in embedded systems. Today, the components and systems groups see that software will help them extract maximum capability and efficiency from the engineered system. However, digital groups view automotive software more expansively, not only to improve vehicle functionality, but also to meet mobility objectives that generate new value for the organization. It is paramount that decision makers have a clear strategy for how their organization will develop the software-defined vehicle to enable this value creation.

Development Challenges

Our observation – compiled while working with every major automotive organization around the globe – is that organizations have established capabilities and maturity to address several key development challenges such as:

- Late discovery of defects
- Missing cross-domain knowledge and system design expertise
- Hardware prototyping slow and expensive
- Compliance to standards

These key challenges, however, have exacerbated in recent times by technology advances like AI and software-defined systems, resulting in their continued relevance. This exacerbation has also led to rise of certain **disruptive challenges** plaguing the transformation teams:

Complex software

Vehicle software architecture is changing from many function-specific ECUs to a mixed approach involving function-specific ECUs, domain ECUs, zonal ECUs, and vehicle controllers. This architecture also needs to account for future use of cloud infrastructure and relies on technologies that are relatively nascent for automotive applications like Linux (or other vehicle operating systems and architectures like Service Oriented Architecture (SOA)), edge enablers, automotive ethernet, and service-oriented software interaction.

Data management

Vehicle fleets generating petabytes of data per day are creating a data management problem. Engineering teams, who used to work with limited test data, now must access and use both stored and live data from the fleet. Fleet data needs to be aggregated and segregated according to the business case for each data set, whether maintenance, future development, or services.

Difficulty in harnessing AI expertise

Engineering teams are challenged to find AI expertise and to leverage it in product development. Although complex AI systems can be designed and in some cases are being used as shadow systems to learn from real-world situations, they are still a long way from becoming primary systems.

Clash of two worlds: “Digital” and “Engineering”

Automotive organizations need to build digital capabilities to open new revenue streams. They are increasingly hiring not just pure software engineers but also IT specialists, who bring with them processes, methods and tools. At the onset these might not provide the rigor required for developing automotive grade software and compliance to automotive standards like ISO 26262 and ASPICE. This is causing process and methodology conflict, compounded by differences in culture and technical know-how.

In response to these extended and new disruptive challenges, automotive companies are expanding their development processes by:

- Introducing **new approaches** for early discovery of defects and building cross-domain knowhow
- Reorganizing the engineering teams around software **methodologies**
- Standing up **new** infrastructure and **automated processes** to front-load through virtualization
- Providing engineering teams access to **fleet data for data-driven** AI and software development and to gain insight into how features are being used in the hands of end users

Some of these methods are starting to be adopted by automotive OEMs and suppliers. However, to consider the physical nature of the vehicle, to meet safety and audit requirements, and to harness the experience of vehicle engineering teams, these organizations should take a fresh look at how these methods are employed to address these challenges.

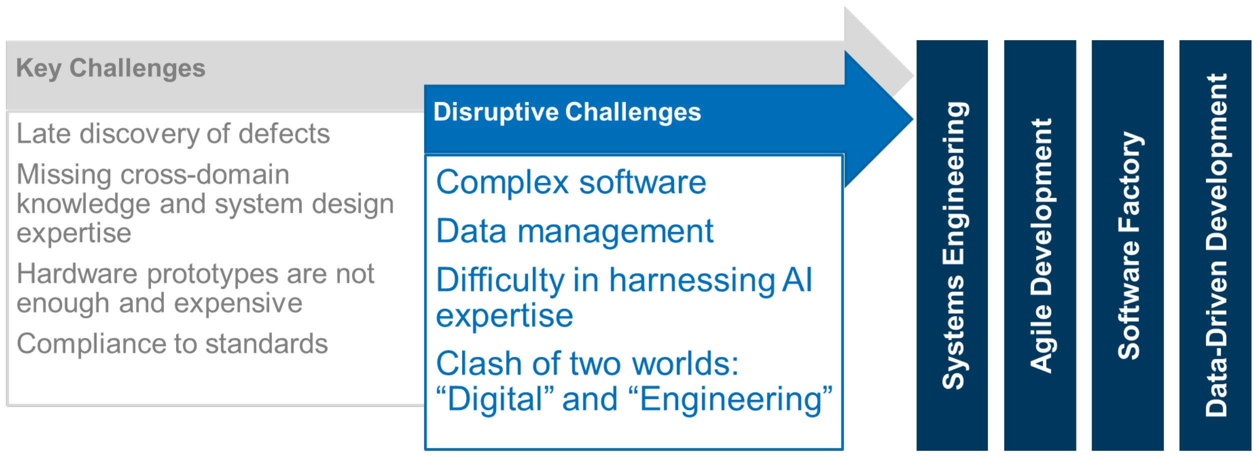
Key Challenges
Late discovery of defects
Missing cross-domain knowledge and system design expertise
Hardware prototypes are not enough and expensive
Compliance to standards
Disruptive Challenges
Complex software
Data management
Difficulty in harnessing AI expertise
Clash of two worlds: “Digital” and “Engineering”
Systems Engineering
Agile Development
Software Factory
Data-Driven Development

Figure 1: Summary of key challenges and new disruptive challenges with strategies to address them.

Systems Engineering

Automotive companies have been introducing systems engineering approaches to make complex multidomain systems and compliance with high safety standards a reality. Following the method, models act as living representation of the system specifications, designs, and implementations, and simulating those models provides a way to test and gain early feedback. However, how can systems engineering enable the rapid delivery of software, rather than gating the software pace to the slower rate of change for electronics and mechanical parts?

Agile Development

Agile development focuses on value creation and guards against a situation where a huge amount of time and effort is spent documenting requirements that turn out to be wrong, incomplete, or changed. Agile organizations have already established test-driven continuous delivery pipelines to reduce time to market for generated artifacts. The artifacts delivered might not just be software components/code but can also be rapid *vehicle-level* tests designed to test interdependencies of components. So, how can software be delivered rapidly and incrementally (a hallmark of Agile), while ensuring that rework due to incomplete or changed requirements is reduced, automotive quality standards are met, and the software is compliant to industry standards and regulations for all product line variants?

Software Factory

A software factory automates the development and maintenance of variants of an archetypical product by adapting, assembling, and configuring framework-based components. However, significant amount of architecture and function development takes place during the exploratory phase of development, when test procedures aren't fully defined. What's the right method and the right time to take advantage of the software factory and its automation capability? How can software be continuously delivered and deployed to vehicle fleet?

Data-Driven Development

Fleet data fuels the development of machine and deep learning driven features (or AI powered features). Data also allows these features to improve over time with feedback from more corner cases. This data, often existing in data lakes, should be accessible quickly and easily, and there should be a well-defined way for retrieving relevant data slices. How can organizations integrate data-oriented capabilities with simulation infrastructure and the larger vehicle development process? How can organizations ensure validation of these algorithms regardless of application context?

Recommendations for Decision Makers

The previous sections highlighted a few challenges that still need to be addressed while adopting systems engineering, Agile development, software factory, and data-driven workflows. We recommend that decision makers implement the following recommendations as they continue to find solutions. (See Figure 2)

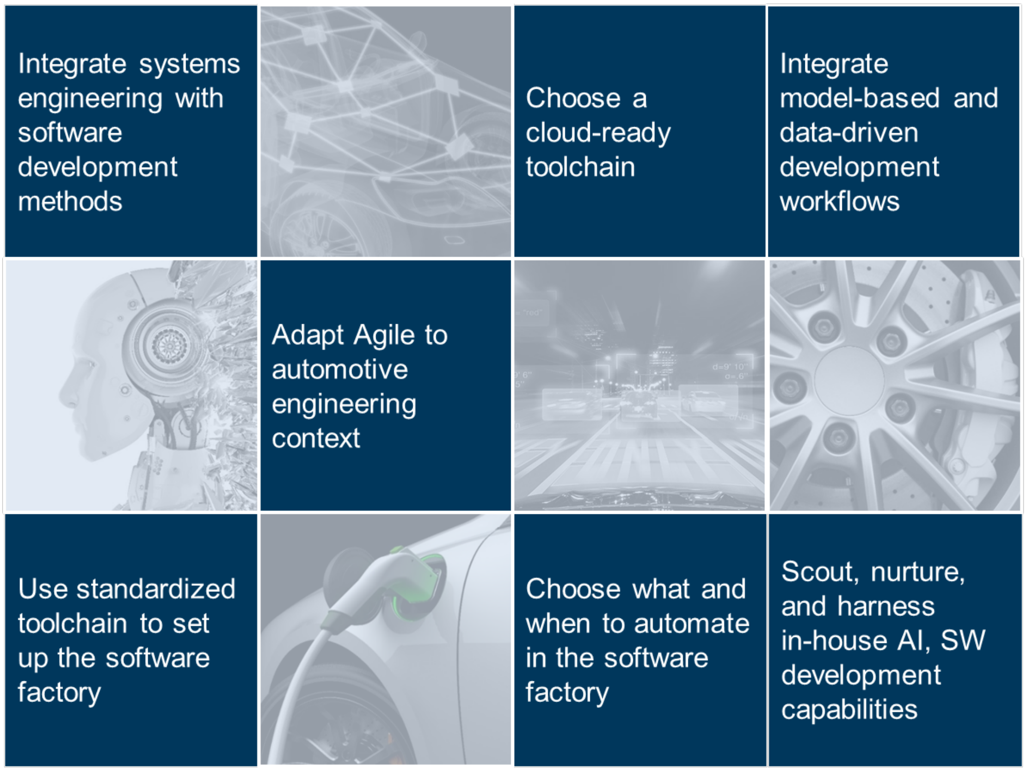
Integrate systems engineering with software development methods
Choose a cloud-ready toolchain
Integrate model-based and data-driven development workflows
Adapt Agile to automotive engineering context
Use standardized toolchain to set up the software factory
Choose what and when to automate in the software factory
Scout, nurture, and harness in-house AI, SW development capabilities

Figure 2: Recommendations for decision makers.

1. Integrate systems engineering with software development methods

To enable rapid delivery of vehicle software, systems engineering should be evolved and integrated with established software development methods, which often tend to be a mixture of model-based and code-oriented approaches. Executable specifications in systems engineering can:

- Help to visualize system architecture and component boundaries
- Tackle system integration challenges early and decouple hardware development and deployment from system/software development
- Help to inculcate system-level know-how in cross-functional teams while helping to differentiate safety-critical automotive grade software from the rest
- Enable front-loading of verification and validation of requirements
- Lead to early discovery of hardware constraints imposed on software

2. Adapt Agile to automotive engineering context

Agile must be adapted to automotive-grade vehicle software development. To adapt Agile in an automotive organization:

- Use shorter sprints to clarify customer requirements and while evaluating if a technology is suitable for the task at hand. This helps to cover both market and technology risks. In such a set-up, maturity of vision/goal decides what purpose each sprint will serve –

analysis/design/test. The output of such short sprints is a system simulation demo, and not a software component.

- Use longer sprints to develop a new product for which basic risks have already been reduced, or to develop/improve a product that is already in use. The output of such a sprint is a system deployment demo with the hardware-in-loop (HIL) or a virtual HIL including the software component. This helps in getting continuous feedback within a sprint to allow for incremental changes. Vehicle-level tests are virtualized and simulated at large scale (e.g., on HPC or cloud) so they can be incorporated in sprints.

3. Choose a cloud-ready toolchain

Software build & test and simulation are shifting to the cloud. This trend is very likely to accelerate in the coming years both for interactive development in the cloud and for scaling use cases. Furthermore, the move to cloud also promises to enable team collaboration and break down silos within organizations. To capitalize on this value-add, organizations should carefully develop their cloud strategy and ensure that expected risks have not just been worked out, but also mitigated and managed. A cloud-ready toolchain must also support DevOps for primarily three use cases:

- Core Development and fleet testing,
- Continuous system development and field updates through OTA, and
- Mobility products and services.

4. Use standardized toolchain to set up the software factory

Continuous delivery (CD) is a key capability which organizations need to master to support perpetually upgradable vehicle platforms through over-the-air (OTA) or Feature-Over-the-air (FOTA) updates. CD helps implement DevOps by connecting development and operations. This will require different organizational groups to work together and alignment of model-driven and data-driven development pipelines.

To iron out the issues related to interoperability and interdependency, standardize your toolchain in the software factory to integrate the development pipelines and to focus on high-quality deliverables. A standardized toolchain lets engineering teams focus on value creation and helps streamline processes for complying to workflow standards like ASPICE and safety standards like ISO 26262, SOTIF, ISO 21434 etc. It is advisable to engage closely with your key vendors and share requirements with them to get early access to technology and influence their technology roadmap. Leaders in the industry also leverage their vendor ecosystem to ramp-up their teams on the needed technology, especially when changing software development methods.

5. Choose what and when to automate in the software factory

To continuously deliver value – in the form of models, software, or artifacts – routine and highly frequent tasks like regression tests, metrics generation, and report generation should be automated.

The first step towards automation is to setup a system simulation setup to find and plug gaps in the system specifications and avoid rework by *not* automating tasks too early. Once the specifications are implemented through models and validated through simulation, there is increased confidence in the intended functionality and that is the *right time* to automate. It is paramount to choose tools and methodologies that give you early feedback during interactive development and enable automation at the right time. Recommended tasks are Model-in-Loop (MIL) and early Software-in-loop (SIL) using code and model integration and simulation. This is a key concept which also helps adapt agile for automotive engineering and developing automotive grade software.

6. Scout, nurture, and harness in-house AI and software development capabilities

Find out what you already know. Capitalize on existing AI and software development experience in the organization, which will include model-based and code-based approaches. Assess the current maturity of teams on both these approaches and identify leading and lagging metrics. Track those metrics to know the health of your development organization. Maintain the trajectory on leading metrics by identifying expertise and applying it to innovate and to pursue continuous improvement. Build centers of excellence to improve the lagging metrics. The payoffs here depend on the area/domain where these capabilities are applied, such as improving understanding of system physics, using code generation to comply with organizational guidelines, or using simulation across domains both at the system and component level.

7. Integrate model-based and data-driven development workflows

Model-based and data-driven approaches may be portrayed as incompatible, but automotive teams across the board already have strong experience integrating these two approaches. For instance, drive cycle tests capture real-world driving data, which can be used to improve battery efficiency by improving system design models, and to validate the updated designs against the validation data set. When data lifecycle and model lifecycle work together, they support each other to not only confirm software design but also electromechanical/sensor design, or any combination of them. This leads to a consistent system and software development process. This combination of data-driven and model-based approaches should not be limited to driving assistance systems and should also be applied to hybrid or electric powertrain, motor, and battery development.

Conclusion

Automotive companies are transforming their development processes with methods such as system engineering, Agile and continuous integration, and data-driven workflows. Some of these originate from digital groups while others are championed by system/component groups. We believe integrating these new development methods into a consistent software process is critical to overcoming the typical divide between system/component groups and digital groups.