SIEMENS
Ingenuity for life

Siemens Digital Industries Software

Creating a unified platform for automotive embedded application development

Executive summary

Automotive embedded software complexity is ballooning as software applications play a critical role in delivering the features and functionality demanded in modern vehicles. The growing importance of software is also increasing the complexity of the interactions between hardware, software, and physical systems. Current software development methodologies are incapable of managing the numerous and intricate cyber-physical interfaces in the cars of today. To compete, companies must evolve their software development processes to establish a common digital thread connecting software and physical systems development.

Piyush Karkare
Director Global – Automotive Industry Solutions

Contents

The modern car is a complex system of powerful computing units, sensors, actuators, and mechanical and mechatronic systems. Today, more and more vehicle functions and features are being achieved electronically. Users expect a seamless and intuitive experience while using and enjoying the many exciting features offered on modern vehicles (figure 1). This trend is making the software that manages these vehicle systems of much greater consequence. In addition, software is crucial for companies hoping to capitalize on the promise of connected, electrified, and autonomous vehicles. The result is a marked increase in the sophistication and complexity of in-vehicle software, and thus the challenge of developing, verifying, and validating such software.

Product engineering and software engineering follow inherently different development lifecycles. Engineers manage software development separately and typically only validate the software-hardware interface at predefined checkpoints. However, the rising intelligence and connectivity of vehicles are making the interactions between software and physical systems more complex, exposing the deficiencies of current processes, tools and methods. Software development work completed independent of the holistic vehicle system context increases the occurrence of problems with hardware compatibility, software quality, and program accountability.

To compete in the technological race for the future of mobility, companies must evolve their software development processes today. A common digital thread connecting software and physical systems together is the only way to take control of the increasing complexity of smart and connected products. This will enable a closed-loop behavioral representation of a vehicle's software and hardware systems for continuous validation throughout the product lifecycle. A robust digital thread will help engineers ensure that software features are fully compatible with the vehicle in which they are deployed and complete with evidence showing that all related tasks and deliverables are available on time. The digital thread will also track accountability for changes regarding not just who, but how and why.

Figure 1: Modern consumers expect a seamless user experience in vehicles, self-driving or not.

Challenges and the feature-centric approach

In-vehicle software development is a complex environment involving multiple domains across several organizations all working to implement a common set of vehicle features (figure 2). These software features evolve from requirements defined by vehicle or platform-level interactions. They are then refined with the needed electrical connectivity and network communication.

When designing and building applications, it is easy to forget that each application is part of a bigger system. Applications call on individual vehicle functions to perform some task that is then used to enable one or many vehicle features. Just as an application may help enable multiple vehicle features, any single vehicle feature may call on several applications across different ECUs.

In this case, the lane-keep-assist software feature must interact with sensors, multiple processors, steering actuators, and inform the driver about the actions being taken. The requirements for each embedded software application ultimately are derived from the vehicle-level system design and development process. This is the system context for the embedded software application. It is essential that the system context is considered across entire embedded software application development process.

Original equipment manufacturers (OEMs) typically deal with the entire vehicle-level software feature development. Much of the detailed control algorithm and embedded software application development is done at the supplier-level, although OEMs are beginning to bring

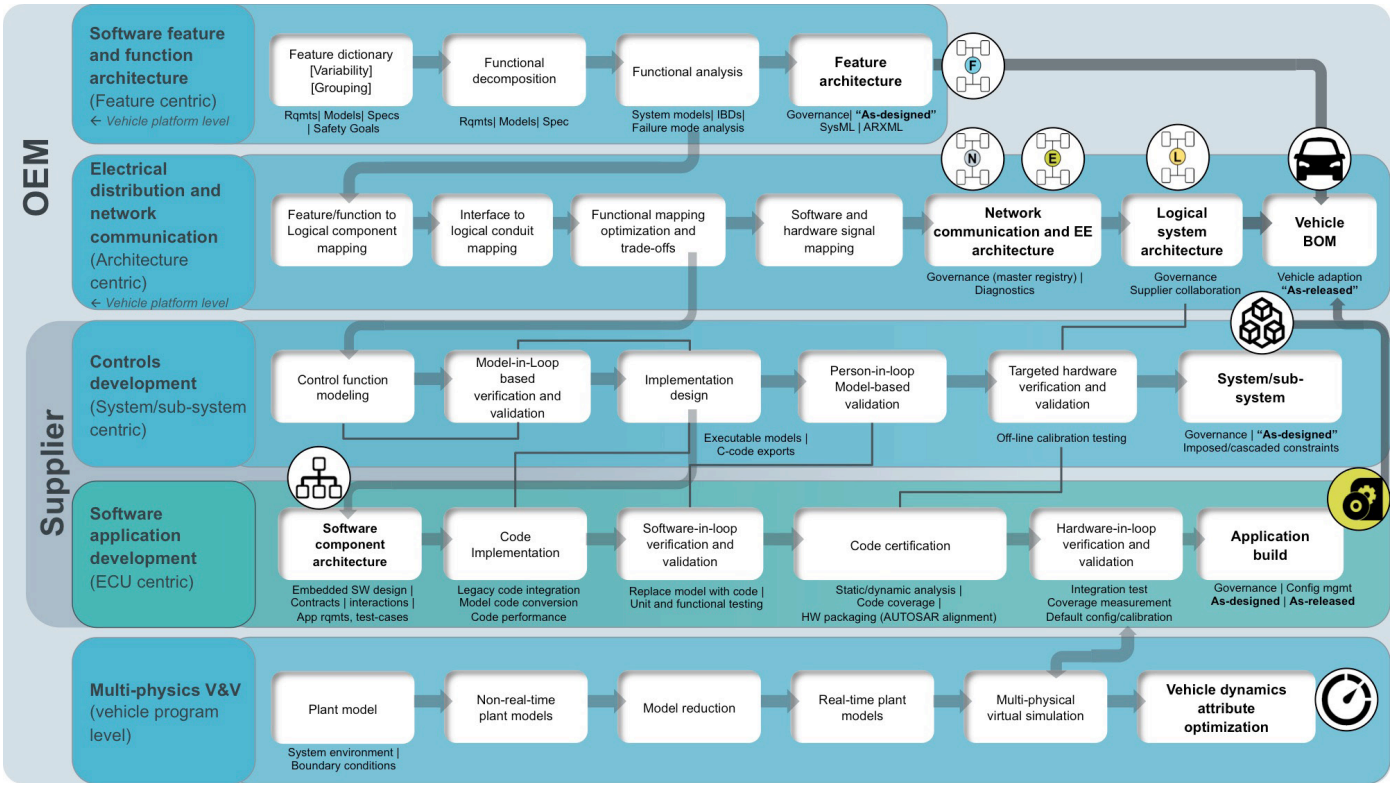

Figure 2: Automotive embedded software development is a highly complex task that involves many organizations across varying engineering domains.

critical feature development, such as infotainment, ADAS, and powertrain features, in house due to extreme competition in these areas.

Control algorithms are needed for these software features and functions to verify and validate the desired functionality. The control algorithms boil down to specific functional behaviors and interactions across software components that are implemented in electronic control units (ECUs). These components are compiled into a software application that contributes to the cyber-physical system that implements the vehicle feature. Typically, software applications are compatible across all vehicles that share a common platform.

Complicating the implementation and operation, each supplier is responsible only for their own content, not the overall feature or features their ECU content would potentially support. Engineering organizations at both suppliers and OEMS tend to take an ECU (hardware) centric approach – focusing less on the feature that is being implemented. In such approaches, ensuring the software feature can be verified and validated continuously as various software and hardware components reach maturity is a very challenging and time-consuming process. Engineers often uncover feature-level bugs late in the process due to delaying verification and validation. This causes costly late-stage changes, reduced testing time, and delayed product or feature release. As the complexity and number of distributed features increases, the probability of not finding issues, or finding them too late, increases tremendously.

A feature-centric software definition approach can help organize the chaos of automotive embedded software definition. A feature-centric and architecture-driven approach allows systems engineers to focus on defining and managing the functional decomposition of software features (feature architecture) and allocating or re-using software components to logical electrical components like ECUs, switches, actuators, and more (logical architecture). Electrical connectivity, power-distribution, grounding, and fusing can now be optimized based on the software feature needs (electrical architecture). Network communications can also be optimized and secured (network architecture).

As the vehicle programs ramp up, existing hardware-software relationships between software functions and logical electrical components will identify each of the electrical components that are necessary for each vehicle feature. Logical components can then be assigned to physical part-numbers tracked within the particular vehicle program, providing end-to-end traceability of how the software features are engineered and used by each vehicle program (physical architecture). Keeping track of all the activities at OEM or suppliers becomes a challenging and relentless mission. Enabling effective collaboration across these activities will help engineers deliver compatible, complete, and accountable software features to the vehicles.

A unified platform for automotive embedded software development

Effectively implementing a feature-centric approach requires a robust, secure, and widely accessible methodology to design, create, track and improve the complex software features that are distributed across a multitude of in-vehicle ECUs and often sourced across the globe from many suppliers.

Achieving such a methodology requires a unified platform for embedded software development to coordinate all the activities across a diverse tool-set to deliver a fully verified and validated build under hardware and system configuration constraints (figure 3). With such a platform, OEMs and suppliers can consolidate data-flows across the tool-chain eco-system and synergize to optimize process, methods, and tools integrations.

A unified platform that orchestrates all activities across the embedded software application definition, planning, development and delivery lifecycle, while also connecting with varied toolsets, will facilitate organic collaboration among many engineers, ensure traceability, and promote the re-use of available data.

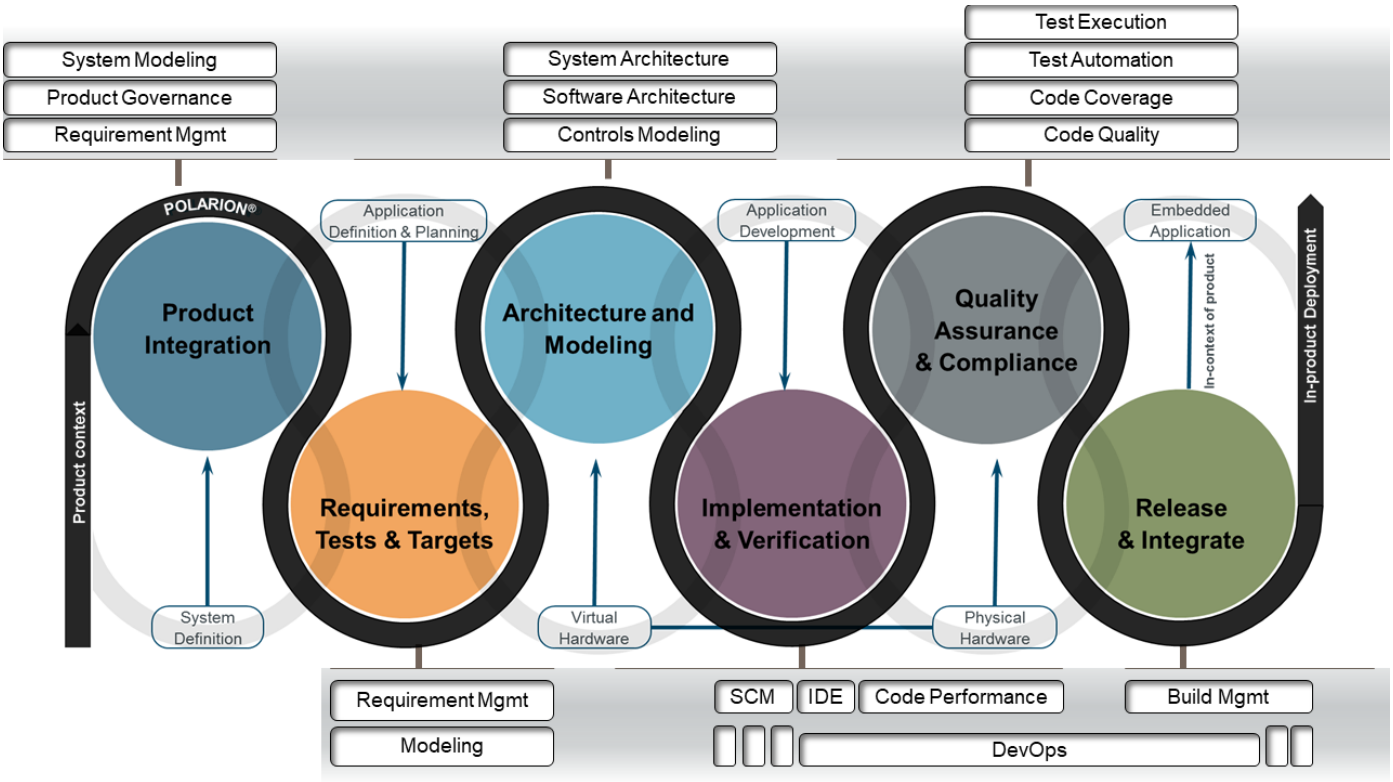

Figure 3: A unified platform for automotive embedded software development is needed to deliver verified and validated application builds based on hardware and system constraints.

The three processes of embedded software development

The orchestration of automotive embedded software application development can be divided into three segments (figure 4):

- Application definition and planning
- Application development and quality assurance
- Application delivery and monitoring

The key is to continuously ensure overall consistency and compatibility between the work that various stakeholders are performing across many platforms and organizations. A unified software development platform helps companies to orchestrate each of these processes, and the activities therein, with a single digital thread. Then, individual organizations can use AGILE or other methodologies to deliver robust embedded software applications on budget and on time.

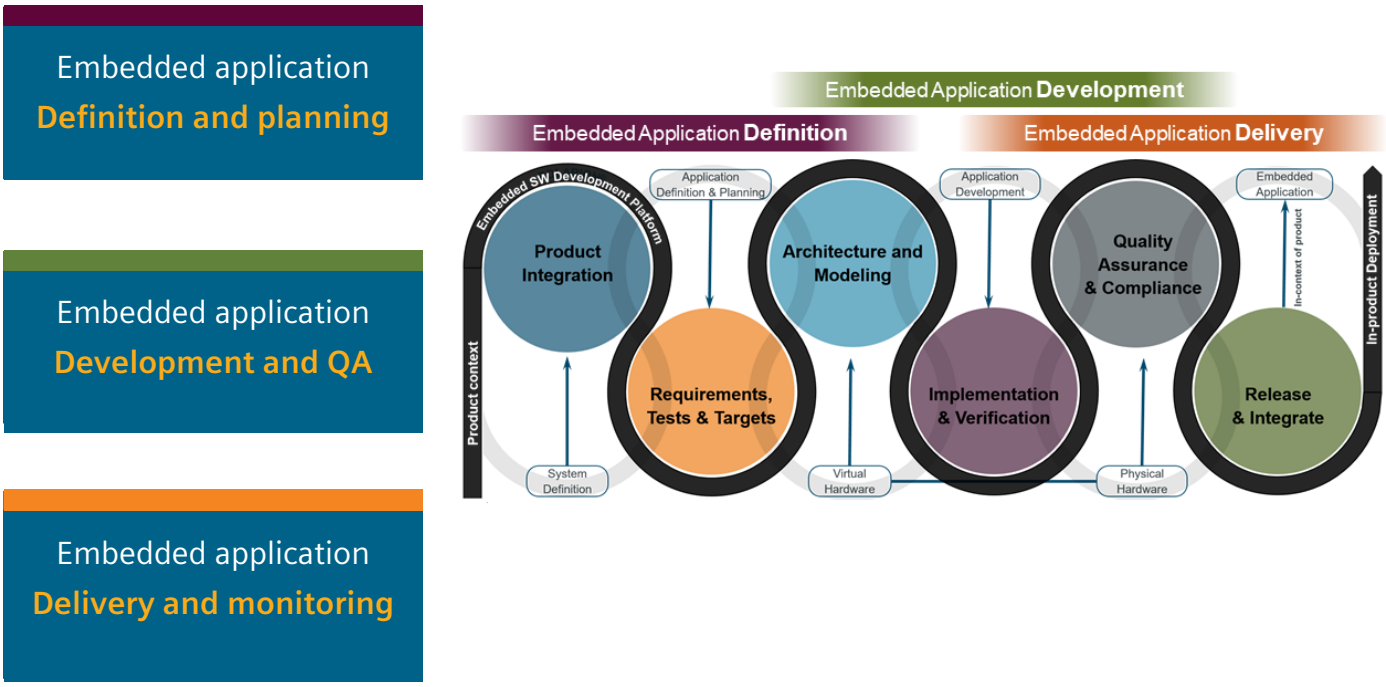

Figure 4: The three segments of feature-centric automotive embedded application development

Application definition and panning

Polarion, as the advanced software development coordination tool, can consume and track the system-level product definition to create a direct link between system-level changes and application development, ensuring the project and the overall system stay in sync. The system, in this case, is a generalization of the vehicle-level feature abstraction.

The system-level definitions and hardware constraints are codified into system-level requirements and specifications. Software engineers can then decompose the needed mix of hardware and software requirements, noise factors, failure modes and effects analyses (FMEAs), test-cases, goals and begin embedded software application-level definitions and planning. During planning, software teams can assign tasks for modeling, coding, test-execution, build production, and more across the needed toolsets. At any given time, software engineers can peek into the system-level definitions and constraints and collaborate with other system users, or be notified if there are system-level changes to evaluate software-level impacts.

The creation of, and any subsequent changes to, detailed software requirements will trigger software architecture and modeling changes. With the requirements, software engineers can update these models to align with control algorithms. Software engineers now execute detailed model-in-the-loop (MiL) tests to verify and validate that desired outcomes are achieved at the application and system-level before any code-level changes are triggered. This allows engineers to discover critical risks, issues and defects while staying aligned with system-level directives early in the process.

Application development and quality assurance

The software component architecture and modelling tasks verify and validate that component interactions achieve the desired functionality. As models get more robust and complete with verification and validation, code changes and updates can be completed and tested with software-in-the-loop (SiL) and further down with hardware-in-the-loop (HiL) testing. Engineers can then perform vehicle modeling-level updates and test again to ensure consistency, compatibility and overall accountability with needed reports and audit trails at the vehicle feature-level.

This model-based approach not only speeds up the process but can also instill methods such as SOTIF (Safety Of The Intended Functionality), ensuring that the software is working as intended, and hazards and unintended functionality are prevented by-design. Incorporating SOTIF methods complements standard functional safety and systems theoretic process analysis (STPA) approaches that mitigate risks by employing safety goals that assume faults will occur. This combination produces exceptionally robust automotive embedded software applications. This also allows the engineers to link standard process descriptions and required work-products in the unified software environment. This enables the engineers to focus on engineering tasks while the tool manages the process adherence, traceability, consistency, collaboration, and lifecycle of the data created.

Application delivery and monitoring

Finally, the embedded application must be delivered to the software assembly that will be included in the final vehicle bill of materials (BoM). But first, engineers must prepare the application for delivery and establish infrastructure to monitor the application after it is delivered. This includes constant monitoring of how and where the application is being used in the vehicle architecture.

Throughout the application development, delivery, and monitoring, the software development platform solution can connect with various tools to provide code performance, test coverage data, and to ensure alignment with

methods and coding standards such as MISRA-C. Further, the unified software development environment supports the collaboration and coordination needed to apply appropriate updates to the code and facilitate the necessary quality assurance processes.

Eventually the application must be verified and validated with virtual and physical hardware, and other peripherals to ensure the desired functionality is achieved and that the system will perform safely. This ensures the embedded software application is complete, compatible and meets the vehicle system-level requirements and needs.

The software engineers must ensure that each software build is delivered to the correct vehicle variant structure. Vehicle platforms spawn dozens of discreet vehicle variants with a mix of shared and unique features, components, and embedded hardware. The software domain has to configure builds to match each of these variants, which quickly becomes very complex. Software engineers must ensure that the software being delivered to each vehicle is fully compatible with the hardware in that vehicle variant. This is typically accomplished by packaging the software configuration, calibration, and bootloaders that coincide with each ECU in the vehicle structure as a software assembly for delivery. This is especially important when updating customer vehicles in the field (figure 5).

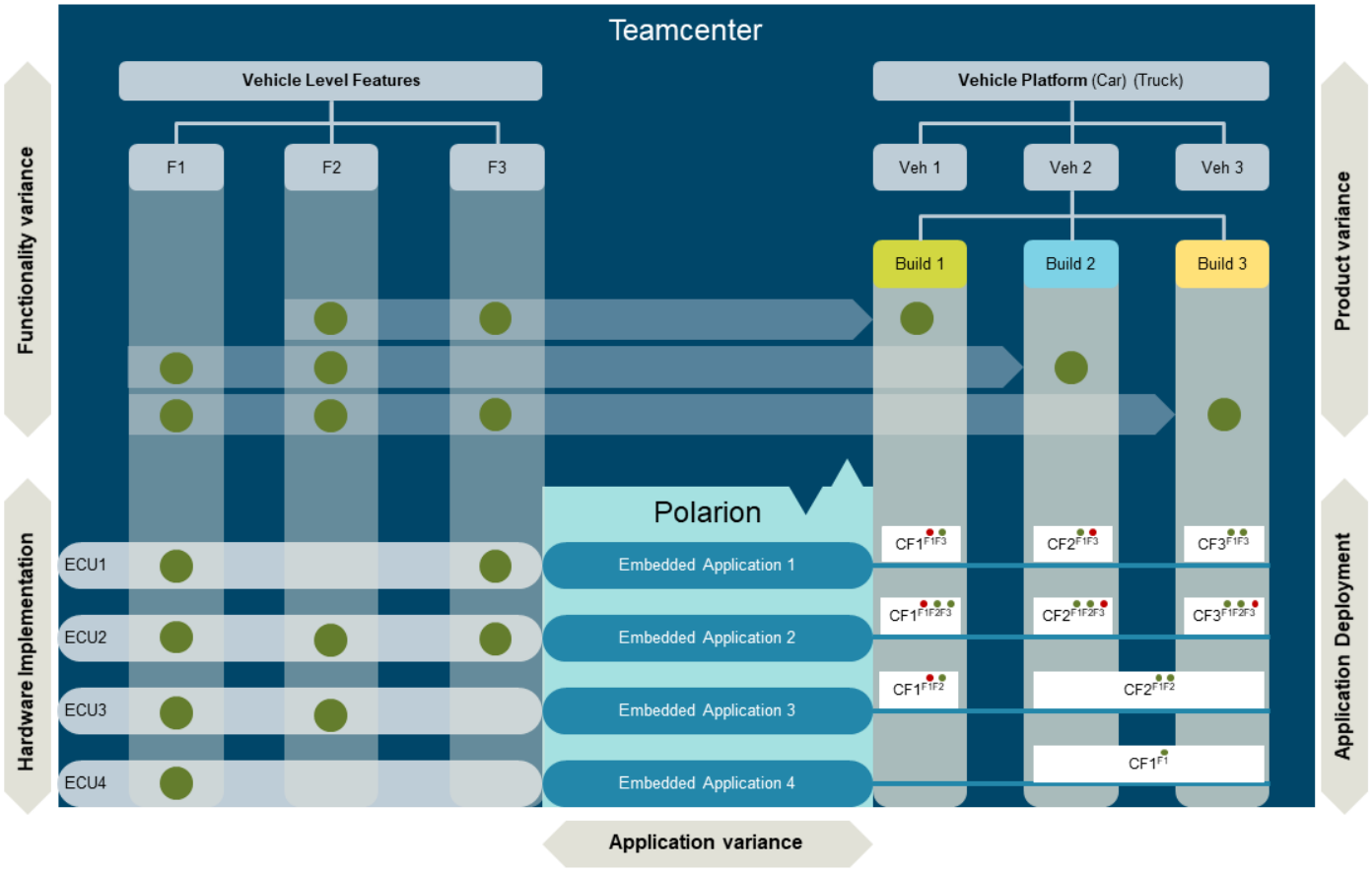

Figure 5: Vehicle features, vehicle build variance and management of application builds with configurations.

Another key element to automotive embedded software is application configuration management. Software application design is becoming less dependent on the hardware on which the application will run as companies begin to develop embedded operating systems. This is analogous to smart phone applications and operating systems. Application compatibility is based primarily on the operating system, rather than the hardware. As a result, it is becoming very challenging to manage the compatibility between automotive embedded software applications and the embedded system-level software (real-time operating system, base software, and etc.), underlying hardware variants, and vehicle variants (figure 6).

A robust combination of application lifecycle management (ALM) and product lifecycle management (PLM) is critical to manage this complexity. A unified hardware-software platform allows the OEM to build and support the basic processes and infrastructure for "As-Designed" (for development), "As-Released" (for engineering), "As-Built" (for vehicle assembly plants), and "As-Serviced" (for over-the-air and dealership updates) software builds.

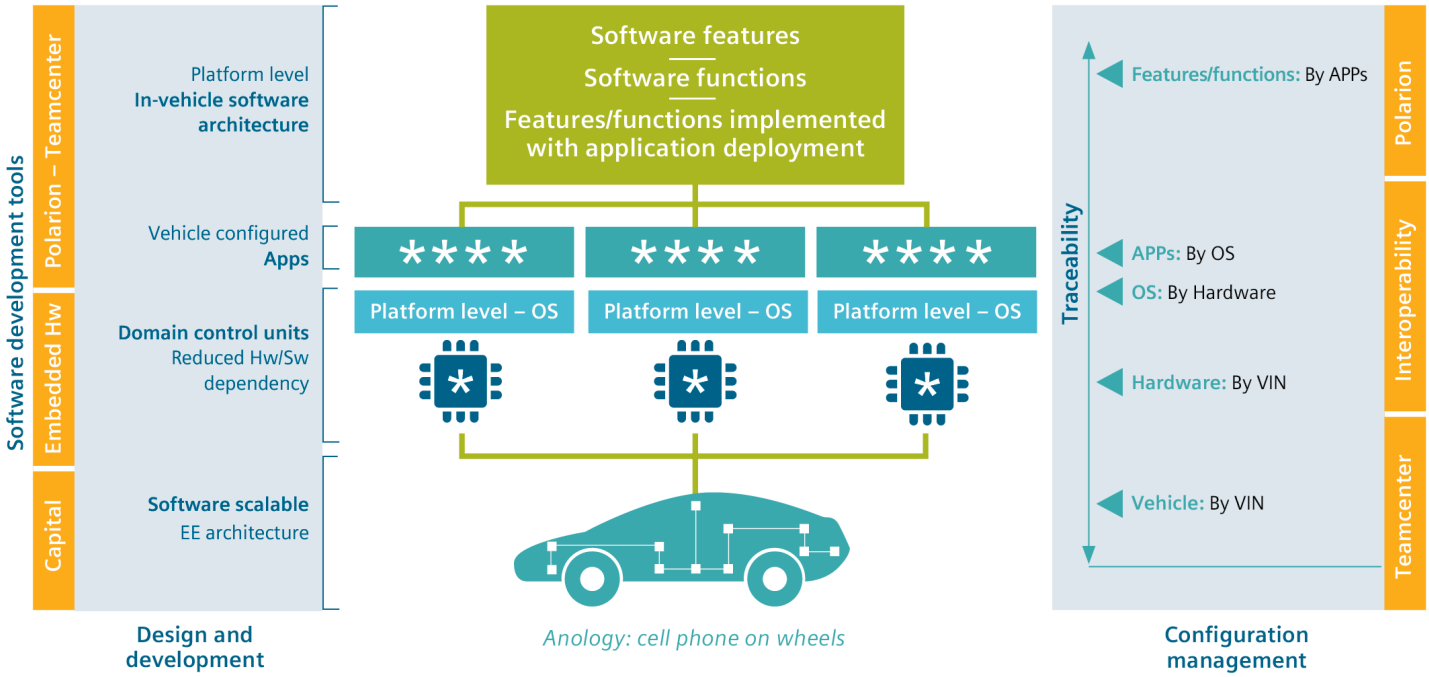

Figure 6: Automotive embedded application configuration management becomes more challenging as software becomes less dependent on hardware, similar to the smartphone application ecosystem.

Example application development flow

This example walks through the development of a new application for a specific ECU, in this case the adaptive cruise control module (ACCM), using a Polarion-based platform for software development (figure 7). A system-level change of combining two features – automatic emergency braking (AEB) and adaptive cruise control (ACC) – has been developed at the system-level abstraction. First, these new system requirements and specification must be cascaded to the ACCM and all affected sub-systems.

This is going to require significant software changes to the ACCM, in addition to non-software changes outside of the ACCM as peripherals will also be impacted. The changes to the ACCM may include new system requirements, hardware specs, FMEAs, safety requirements and more. ACCM software engineers must stay continuously in touch with the system context as they make the needed changes and updates to the ACCM software. Connecting the software development orchestration tool with product lifecycle management (PLM) solutions facilitates this continuous connection between teams and abstractions, especially with the software platform controlling change management and PLM managing activities at the feature-level.

As part of a sprint, based on the system-level changes, ACCM engineers can implement the necessary changes and updates to software-specific requirements, test cases, test methods, and more. Work items that might be impacted by these changes, based on the existing data artifacts in the software management tool, can be flagged for further review before changes are committed. Most of the work items would have been flagged already for this

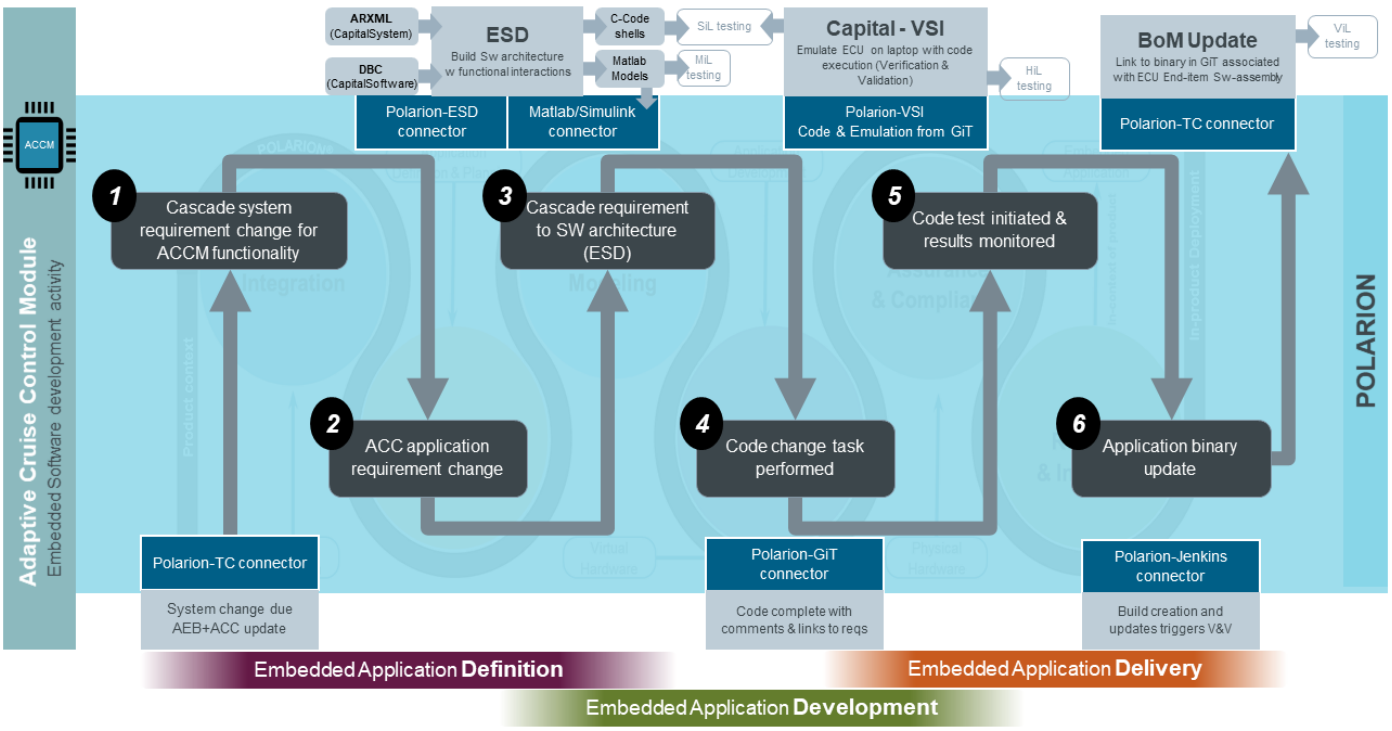

Figure 7: An example application development flow demonstrates the utility of a unified software development platform.

change as part of the engineering change request (ECR) for the AEB+ACC update. This way, the PLM and software development coordination solutions will maintain consistent data for on-going change management.

AGILE processes can proceed with the system-level changes and ensuing updates to software requirements will trigger potential updates to the software component architecture and models in Embedded Software Designer or MATLAB/Simulink. The unified software development platform can orchestrate and flag the suspected models for changes as these changes cascade and decompose down from the system-level. As these software component architecture-level and model-level changes are executed (along with electrical systems and CAN bus network communication changes imported from the electrical systems tool), ESD can export models for Matlab and C languages for model-in-the-loop (MiL) test executions. Depending on the level and amount of changes, preliminary SiL testing can also be initiated to validate the functional interfaces based on the updated architecture before any changes are applied to the code.

Once the functional interfaces are verified and validated, software engineers can begin sprints for implementing code changes. These code changes can be tied back to the source requirements, from step two, and architectural changes. This ensures that changes are accurate and have full traceability to the system-level requirements, or even further back. For example, a marketing-level directive or a warrantee based change coming in from the field to fix a re-occurring issue on customer vehicles. This bottom up traceability is extremely beneficial as it can assess wider cross-vehicle program changes and fix or update functionality with one update, instead of many, saving considerable cost.

As part of the AGILE sprints, updated code can then be tested through SiL on virtual hardware, or via hardware-in-loop with physical hardware. This testing triggered with AGILE methods facilitate continuous integration with software builds and hardware or system-level constraints. The software development platform can also manage and execute hardware-software system-level tests for which these code changes are being made available. Then, overall test results and project dashboards can help engineers check that a robust, verified and validated software application is now available for deployment.

Then, the final software assembly for the “As-Released” BoM can be updated using the engineering change notification (ECN) that was triggered from the PLM tool. The ECN targets the specific ECU part number in the configured vehicle structure. This ensures that an up-to-date software assembly is ready for configuration to be included in the final vehicle assembly. This is then sent to vehicle assembly plants for dealerships for ECU flashing.

Unifying automotive embedded software development

Integrating the software and product development ecosystems by connecting the relevant architecture definition, modeling, and testing tools creates a unified platform for automotive embedded software engineering. This platform acts as a collaborative environment that provides traceability and incentivizes IP reuse. This reduces development time and cost by detecting incompatibilities early in the process. As a result, engineers can build a robust, safe and secure product without excessive iterations.

A unified automotive embedded software platform is fundamental to the coordination of complex software application development because it supports real-time and on-demand access to design data, requirements, test results, and much more directly in the engineering view. Such a platform also contributes to a strong digital thread and makes collaboration between disparate teams and organizations organic and intuitive, boosting the productivity of each stakeholder.

Furthermore, solutions such as Polarion facilitate rapid application deployments by offering templates for Enterprise AGILE, SAFe and other large-scale implementations along with templates for standard compliance such as ISO 26262, A-SPICE, CMMI, and more. These templates are adaptable to customer-specific processes, easing user adoption and accelerating return on investment (figure 8).

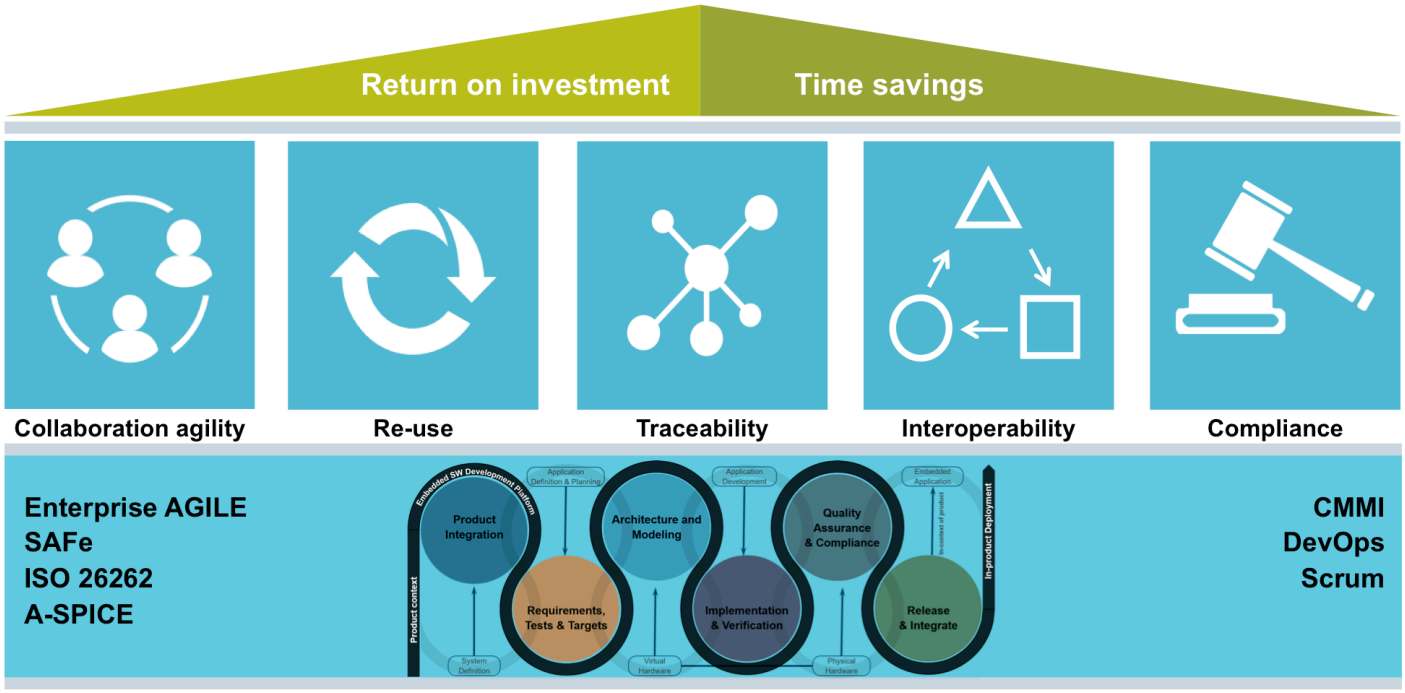

Figure 8: A unified embedded software development platform enables collaboration and reuse while ensuring traceability, interoperability, and compliance. Standard templates help deliver accelerated return on investment and development time.

OEMs and suppliers alike demand for rapid return on investment from these complex engineering environments where they can reduce engineering costs and increase efficiency. A unified software development orchestration platform provides collaboration agility with a configurable environment, making it easier and more cost effective to change agile workflows. The environment promotes reuse as companies leverage proven components to produce variations faster, enabling more differentiated products, and higher returns. Furthermore, these tools provide much-needed traceability for audits, research, and reviews, while saving resources, time, and related costs. Engineers are then able to focus on engineering tasks while the tools track work-products.

In-vehicle software is becoming more and more important to the competitive strength of vehicles in a cluttered and contentious market. Exciting and useful electronic features, such as ADAS and advanced infotainment systems, use increasingly sophisticated software to deliver the functionality and features that matter to modern consumers. To beat market competition, manufacturers will push to meet ever-tightening timelines.

Simultaneously, the expectation for software quality and reliability will surge. System lag, glitches, and poor user interface design will be judged much more harshly as they become more central to the users' interactions with the vehicle. In this new environment, a unified and collaborative environment for SW development that builds-in traceability and IP reuse will prove invaluable.

Siemens Digital Industries Software

Headquarters
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

Americas
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

Europe
Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific
Unit 901-902, 9/F
Tower B, Manulife Financial Centre
223-231 Wai Yip Street, Kwun Tong
Kowloon, Hong Kong
+852 2230 3333

About Siemens Digital Industries Software

Siemens Digital Industries Software, a business unit of Siemens Digital Industries, is a leading global provider of software solutions to drive the digital transformation of industry, creating new opportunities for manufacturers to realize innovation. With headquarters in Plano, Texas, and over 140,000 customers worldwide, we work with companies of all sizes to transform the way ideas come to life, the way products are realized, and the way products and assets in operation are used and understood. For more information on our products and services, visit siemens.com/plm.

siemens.com/plm

© 2019 Siemens. A list of relevant Siemens trademarks can be found here. Other trademarks belong to their respective owners.
78394-C6 8/19 A