Siemens Digital Industries Software

Capital Software Designer

Applying an architecture-driven approach to onboard software design

Executive summary

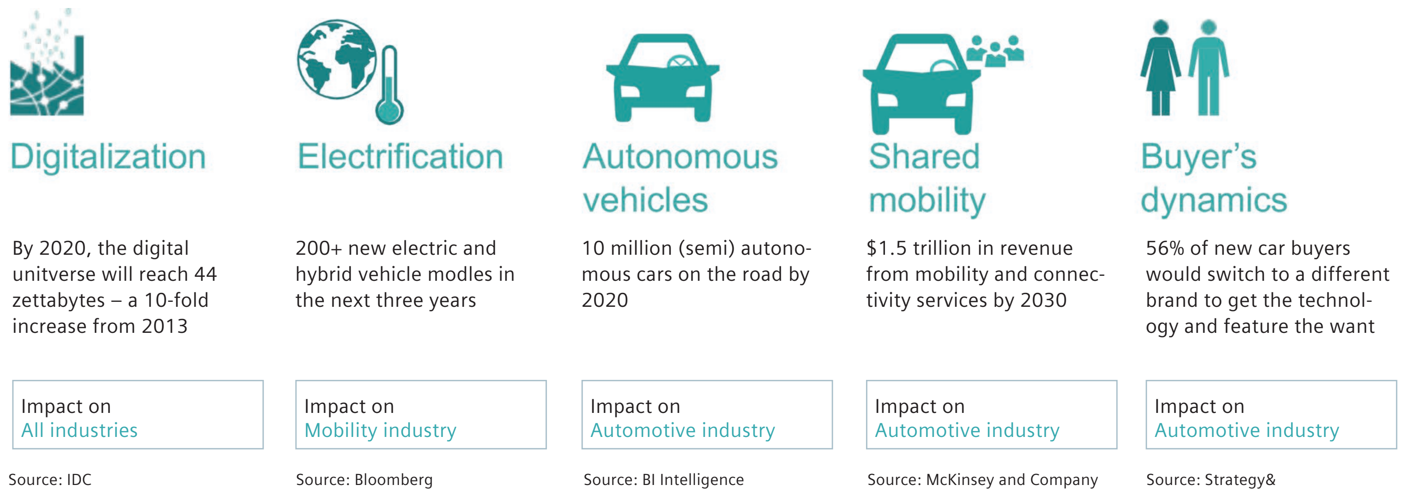
Are you struggling with increasing onboard software volume and quality, a large number of variants and time-to-market pressure?

This white paper reviews market trends that are transforming embedded software development in the automotive industry from an activity owned mostly by suppliers to a shared responsibility between original equipment manufacturers (OEMs) and suppliers.

This digital transformation requires different processes and dedicated process support tooling. This white paper describes the digital transformation challenge and suggests an architecture-driven approach for onboard software design based on the functionality of Capital Software Designer.

siemens.com/swdesigner

Abstract



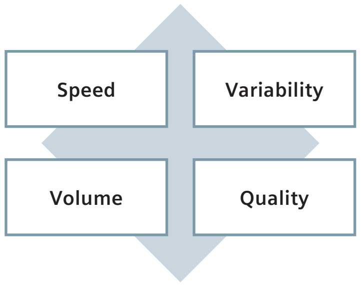
Digitalization	Electrification	Autonomous vehicles	Shared mobility	Buyer's dynamics
By 2020, the digital unitverse will reach 44 zettabytes – a 10-fold increase from 2013	200+ new electric and hybrid vehicle modles in the next three years	10 million (semi) autonomous cars on the road by 2020	$1.5 trillion in revenue from mobility and connectivity services by 2030	56% of new car buyers would switch to a different brand to get the technology and feature the want
Impact on All industries	Impact on Mobility industry	Impact on Automotive industry	Impact on Automotive industry	Impact on Automotive industry
Source: IDC	Source: Bloomberg	Source: BI Intelligence	Source: McKinsey and Company	Source: Strategy&

Figure 1: Megatrends transforming the automotive industry.

The market trends in the automotive industry point towards connected, increasingly autonomous, highly customized, electric and networked vehicles that are perceived by younger generations more as "tablets on wheels" than as traditional vehicles. These vehicles are expected to be extensible over their life through app purchases and installations, and offer passengers added-value services that are based on networks (see figure 1). In addition, time-to-market pressure keeps building, and product complexity frequently leads to defects that aren't detected until late in the process when they are more expensive to fix, thus diminishing the company's profit.

Although software was used to run subordinate, low-level embedded control and entertainment functions in the past, today it is used to: perceive and categorize its environment, coordinate the driving process in advanced driver assistance functions, provide telemetry data to its manufacturer, receive over-the-air updates, and obtain high levels of authority over route planning, engine, gears, brakes and steering. In other words, manufacturers are using software to take more responsibility for the driving process. Classically clear-cut boundaries between infotainment and vehicle operating are blurring.

Consequently, a collection of completely new functions increases the volume of onboard software by orders of magnitude. A large portion of this software is critical for safety. Connecting vehicles to the internet opens it up to security threats, compromising vehicle integrity and passenger confidentiality and safety. Due to constraints regarding weight, power, heat transfer and cabin space, it is not possible to continue adding functions by adding more electronic control units (ECUs). Thus, the industry will see a consolidation of software functions for large, multi-core ECUs. As specified by AUTOSAR, dynamic updates and extensions require a departure from static ECU images. Instead, manufacturers will need to move toward a dynamic architecture more resembling general-purpose information technology (IT) systems, as specified by AUTOSAR Adaptive[1].



Challenges in embedded automotive software.

Black-box paradigm

This trend contradicts the traditional approach in which most embedded onboard software has been developed and integrated by suppliers into components or subsystems. Acquiring and integrating engines, gearboxes, heating, ventilation and air conditioning (HVAC) systems, seats and lighting systems are examples of such black-box systems. Integration happens simultaneously in multiple domains, such as mechanics, thermal, hydraulics, electrical, electromagnetic interference and buses. In this paradigm, embedded software is shipped as a nearly invisible part of the supplied subsystem, and it is not supposed to be frequently updated. Onboard software in this paradigm interacts with other onboard software using well-defined, controlled bus signals. The black-box paradigm has allowed OEMs to focus on defining bus segments, packages and signals, but has also led to a proliferation of ECUs in vehicles, creating problems in terms of weight, consumed cabinet volume, heat and power consumption.

Gray-box paradigm

A gearbox designed by a supplier will deliver its software not as a dedicated embedded control unit, but as a binary executable the OEMs will integrate into a large-scale ECU. This change in technology has a strong impact on the process organization between OEMs and suppliers. OEMs are suddenly required to specify, order, integrate and validate software they had previously not seen. This a gray-box paradigm, as the OEM will typically not develop the implementations. This process change is an example of digital transformation, as opposed to digitalization; see the inset box on the right.

Migrating towards the gray-box approach requires OEMs to take ownership of software requirements and define software architectures, timing and memory needs; provide unambiguous implementation specifications and acceptance criteria to suppliers; select and configure basic software layers, and schedule, integrate, build, verify, validate and qualify delivered source code on a large scale. Since supporting the interaction with suppliers becomes tighter in the grey-box acquisition paradigm, the aerospace industry refers to it as the extended enterprise[2]. Adoption of this paradigm in the automotive industry is well documented in the contemporary literature[3].

Developing onboard software in the extended enterprise requires adequate tool support for connecting to typical application lifecycle management (ALM) systems such as Polarion ALM™ software. The same is true for:

- Connecting software variability to the product lines
- Specifying systems and software requirements, functionality and quality of service
- Describing software architecture and its needed functional and timing properties
- Creating supplier design specifications and integrating delivered software artifacts
- Verifying software compliance with specified properties
- Validating the software in its operational context
- Qualifying software for production use
- Reporting all results to the connected ALM system

Finally, large legacy code bases containing a company's valuable experience needs to be considered, as few projects are start from scratch.

Digital transformation

Adopting digitalization has enabled companies to migrate existing and well-established processes from paper-based artifacts to digital artifacts running on IT infrastructure without changing processes. Digitalization has brought added value since with digital artifacts it is easier to identify versions, re-vise, archive and search compared to paper artifacts. To a large extent, digitalization has preserved the relationships between product designers, manufacturers and their suppliers. Digital transformation is about realizing all the opportunities digital technologies have to offer, including processes and business models[4].

Architecture-driven design

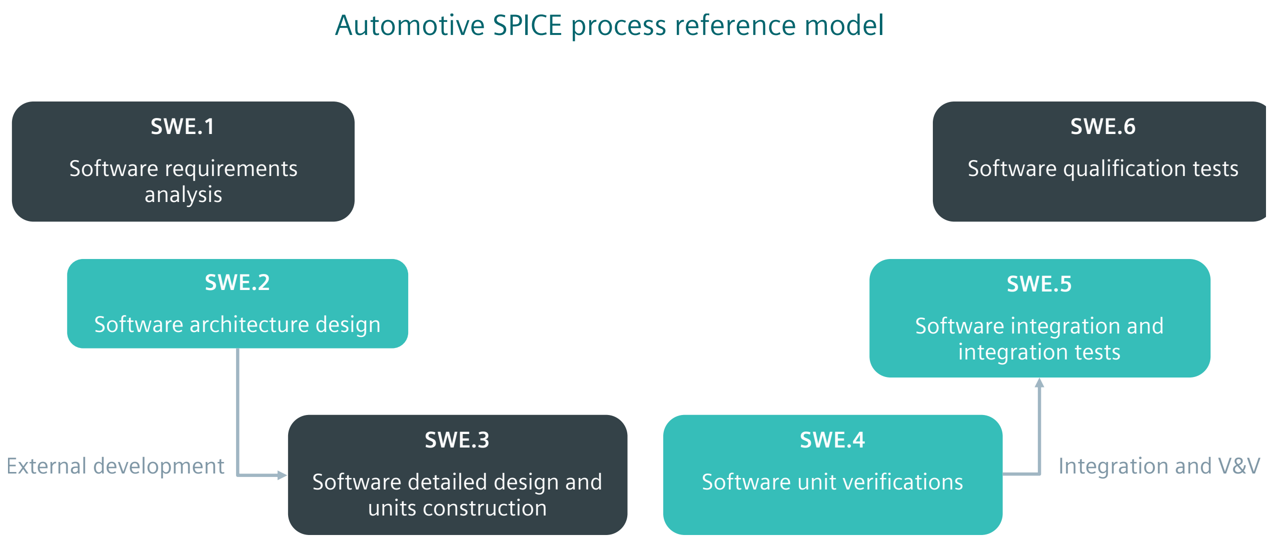

Automotive SPICE process reference model
SWE.1
Software requirements analysis
SWE.2
Software architecture design
SWE.3
Software detailed design and units construction
External development
SWE.4
Software unit verifications
SWE.5
Software integration and integration tests
Integration and V&V
SWE.6
Software qualification tests

Figure 2: Key software engineering process groups of the Automotive SPICE model.

Using Capital Software Designer enables you to defuse the potentially negative effects of increased software volume, faster time-to-market, variant richness and demand for higher quality by taking a uniquely comprehensive, architecture-driven, model-based approach to embedded software design. It is tailored to the needs of the extended enterprise, facilitating collaboration and giving developers complete freedom of choice in their development environment.

Enabling the extended enterprise

Capital Software Designer supports onboard software design in the extended enterprise, from software specification and qualification to tightly integrated services targeting software engineering process groups of the Automotive Software Process Improvement and Capability Determination (A-SPICE) model[5], (see figure 2). A rich architecture model is the core and single source of truth for all these activities. OEMs must master at least the higher-level software engineering (SWE) process groups SWE.1, SWE.2, SWE.5, and SWE.6. Suppliers need to master SWE.3 and SWE.4, but their responsibility may include the higher levels as well, depending on the scope of their supplied units.

Capital Software Designer is focused on processes SWE.2 for software architecture design, SWE.4 for software unit verifications and SWE.5 for software integration. It integrates with SWE.1 for software requirements analysis and SWE.6 for software qualification tests through an integration with Polarion ALM, and with software qualification.

Frontloading defect detection and removal

Architecture becomes useful if it is rich enough to be analyzed before the first line of implementation code has been written to detect inconsistencies in the design. To this end, Capital Software Designer provides data types enriched by physical units, data-flow architecture enriched by formal contracts, and block annotations expressing timing needs. Additionally, all relevant artifacts are variant-aware. An adequately enriched architecture model becomes analyzable for contract consistency and how well it can be scheduled before the first line of code is written. Formal methods based on the Z3 model checker (open source software developed by Microsoft) are employed to provide mathematically sound and dependable analysis results.

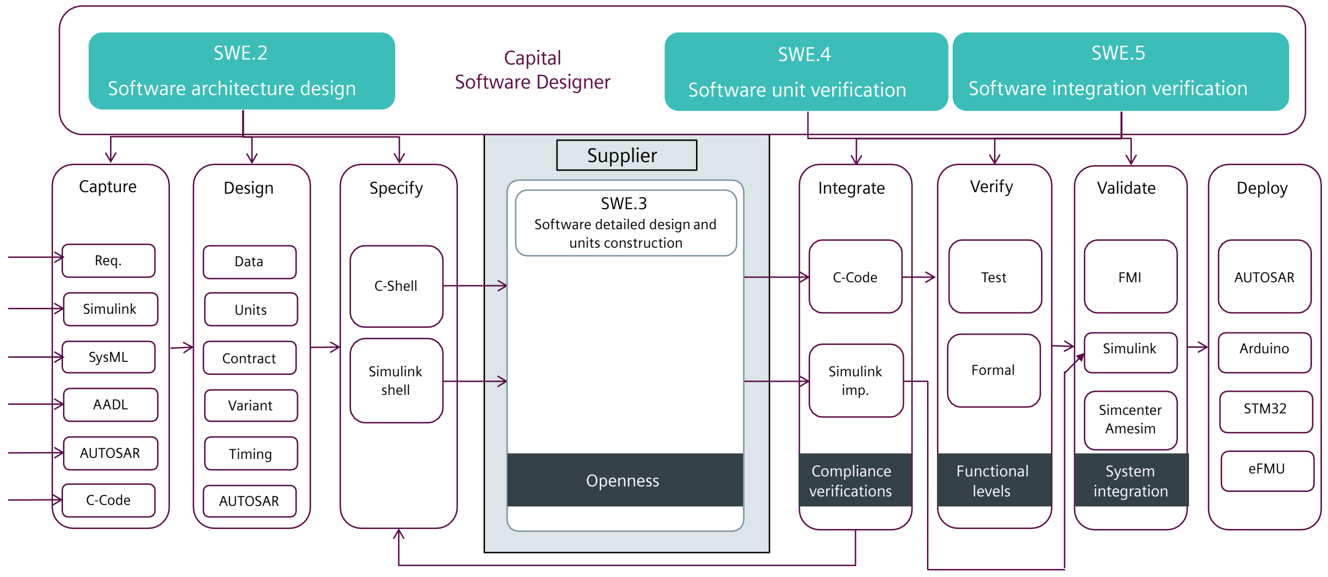


Figure 3: Services needed for onboard software development in the extended enterprise.

With these capabilities, specified software is more likely to be right the first time and need fewer iteration loops between specification and implementation teams.

Delivering software quality

To address OEM and supplier needs, it is necessary to capture legacy code, typically written in the C programming language, as well as existing descriptions of architectural information, which are expressed, for instance, in Systems Modeling Language (SysML), AADL or the Simulink® environment. These legacies need to be integrated with new code driven by model-based development, and verification, validation and qualification steps. Capital Software Designer supports all these steps, regardless how the source code was created. Apart from unit testing mechanisms and integrating with best-in-class simulation tools, Capital Software Designer enables user to leverage formal methods provided by the C bounded model checker[6] (CBMC) to reliably check whether delivered implementation code complies with the block contracts. If defects occur, Capital Software Designer is used to create human-readable examples demonstrating how the defect can manifest itself. This information is ultimately useful for developers to remove the defect.

Capital Software Designer services

Figure 3 describes key services that address the pains expressed in figure 1 and enable efficient onboard software engineering in the extended enterprise. The services for imports, architecture enrichments and executable specification export pertain to clients who design and integrate onboard software into their products. The same pertains to integration, verification, validation and target deployment services. The code realization is executed by suppliers, who may belong to the same organization as the client, or to an external

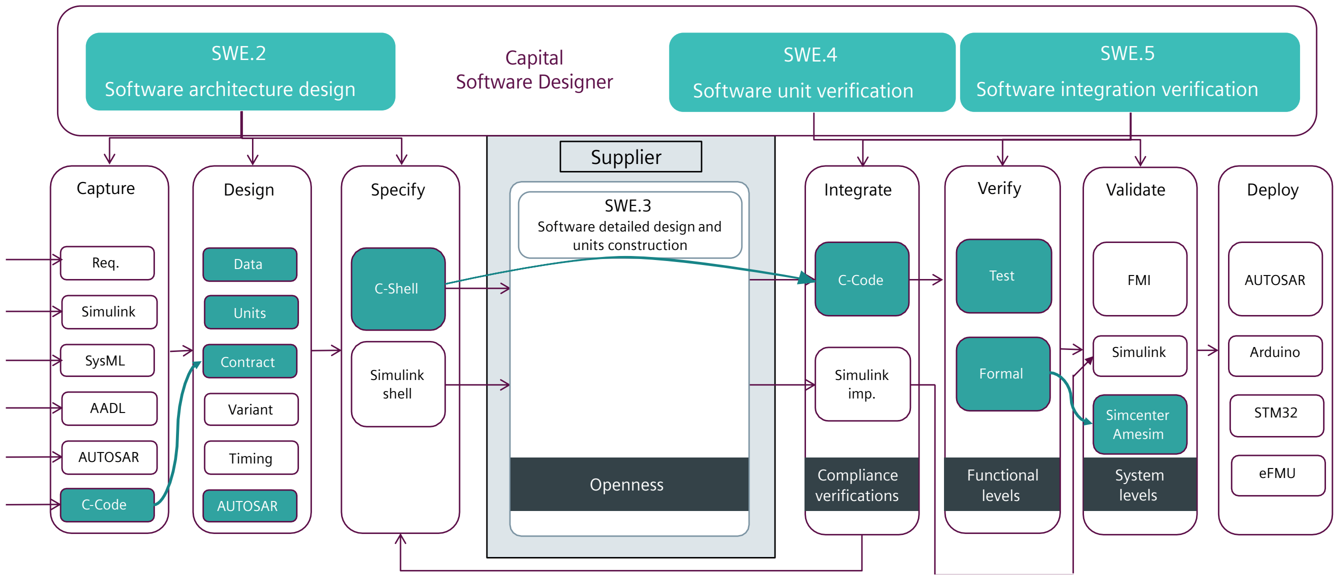


Figure 4: Using Capital Software Designer for a manual C programming-based scenario for onboard software design.

organization. Note the words original equipment manufacturer and supplier are not used here as the client/supplier pair is more generically applicable to supply chains of arbitrary length.

Typical design scenarios for onboard software

This section explains two typical usage scenarios for onboard software design. One starts from C legacy projects and adds new software functions that are handwritten in the C programming language. The other captures legacy architecture models that are improvised using Simulink and brings in new functionality to be developed in model-based fashion.

Of course, mixed scenarios involving C legacy code and new functions in model-based paradigm, which contain mixed implementation paradigms for new functions, also appear in practice and are supported by Capital Software Designer.

The details of capabilities in each service group are explained in subsequent sections.

C legacy projects with new functions in C

Most automotive OEMs and suppliers have large quantities of C legacy code that work and should be re-used, while the organization is moving gradually toward a model-based software development paradigm for reasons explained in section 1. The first step in the migration is to import, analyze, understand and refactor C legacy code, as highlighted as a first step in figure 4.

In the second step, the data-flow architecture is to be derived in a semi-manual way from the legacy code, so architecture enrichments from physical units, contracts and timing needs as well as associated upfront analyses, can be used when they become available. The original legacy code is retained and linked to the architecture.

New software starts as an architecture model, and its implementation by internal development teams or external suppliers is federated with executable specifications such as rich C shell templates (figure 4).

Returning the templates with filled implementation, integration checks are executed on the side of the OEM, followed by unit tests, formal verifications and closed-loop validations.

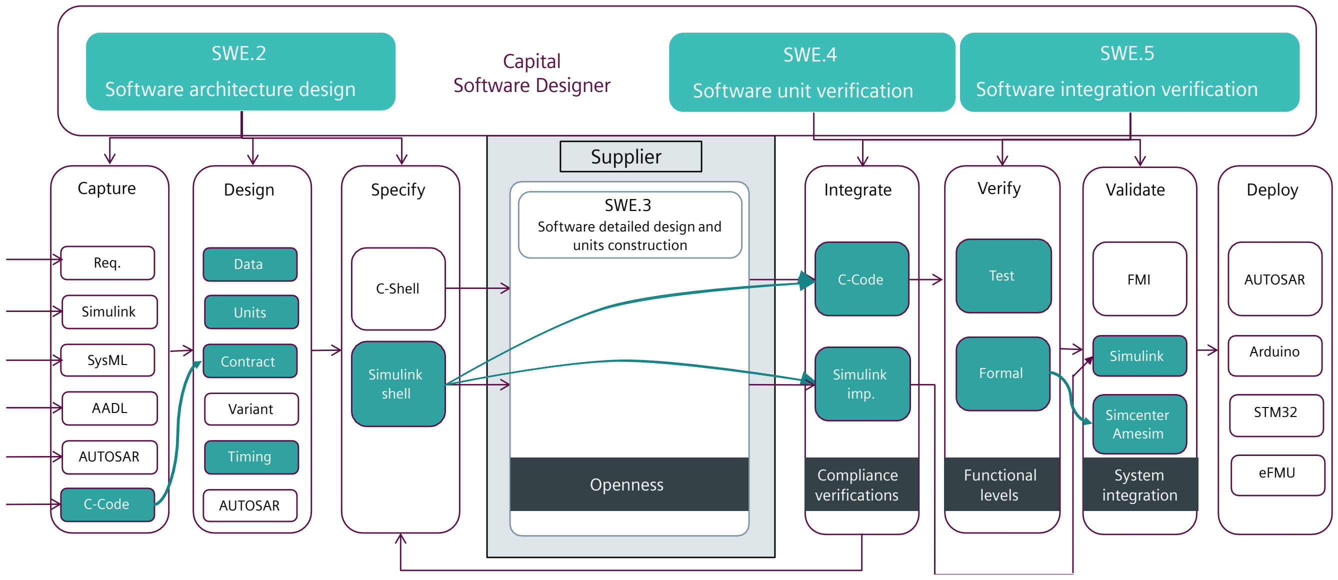


Figure 5: Using Capital Software Designer for a model-based development scenario for onboard software.

Simulink based projects

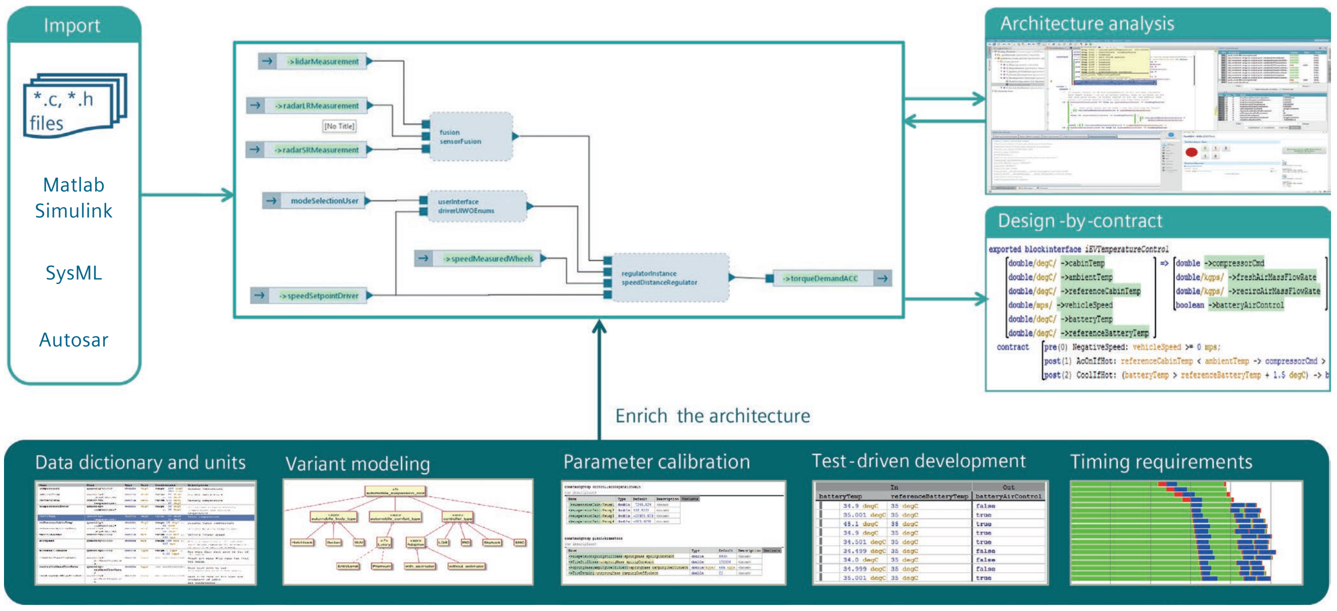
Some organizations have been capturing architecture information in Simulink using empty subsystems to indicate data-flow structure. Such descriptions can be used to populate Capital Software Designer data-flow diagrams, which are then available for further enrichment as described in figure 5.

New software starts as an architecture model. Its implementation by internal development teams or external suppliers is federated with executable specifications as rich Simulink templates (figure 5). These templates are essentially empty systems that contain only ports with the correct name and data type.

Implementation is then conducted externally in Simulink. The production code is obtained from the Simulink models using off-the-shelf code generators such as Embedded Coder or dSPACE TargetLink.

The generated C code is then subjected to integration checks, followed by unit tests, formal verifications and closed-loop validations.

Enriched architecture frontloads error detection and drives implementation



```
exported blockinterface iEVTemperatureControl
    [double/degC/ ->cabinTemp               ] => [double ->compressorCmd                ]
    [double/degC/ ->ambientTemp             ]    [double/kgps/ ->freshAirMassFlowRate   ]
    [double/degC/ ->referenceCabinTemp      ]    [double/kgps/ ->recircAirMassFlowRate  ]
    [double/mps/ ->vehicleSpeed             ]    [boolean ->batteryAirControl           ]
    [double/degC/ ->batteryTemp             ]
    [double/degC/ ->referenceBatteryTemp    ]
  contract  [pre(0) NegativeSpeed: vehicleSpeed >= 0 mps;
            post(1) AcOnIfHot: referenceCabinTemp < ambientTemp -> compressorCmd >
            post(2) CoolIfHot: (batteryTemp > referenceBatteryTemp + 1.5 degC) -> b
```

In		Out
batteryTemp	referenceBatteryTemp	batteryAirControl
34.9 degC	35 degC	false
35.001 degC	35 degC	true
45.1 degC	35 degC	true
34.9 degC	35 degC	true
34.501 degC	35 degC	true
34.499 degC	35 degC	false
34.0 degC	35 degC	false
34.999 degC	35 degC	false
35.001 degC	35 degC	true

Figure 6: Capital Software Designer provides rich onboard software architecture modeling capabilities.

Rich architecture is the key to frontloading detection of design errors. Capital Software Designer enrichment covers block interface-type systems, physical units, contracts and timing needs. Furthermore, the architecture drives open implementation using the paradigm and language of the user’s choice. Figure 6 summarizes these capabilities, which are further explained in this section.

Import capabilities

Moving C legacy code to a model-based paradigm is a challenge frequently encountered in the industry (see top left of figure 6). Capital Software Designer derives the call graph from C source projects and displays the call graph to the user in a navigable way.

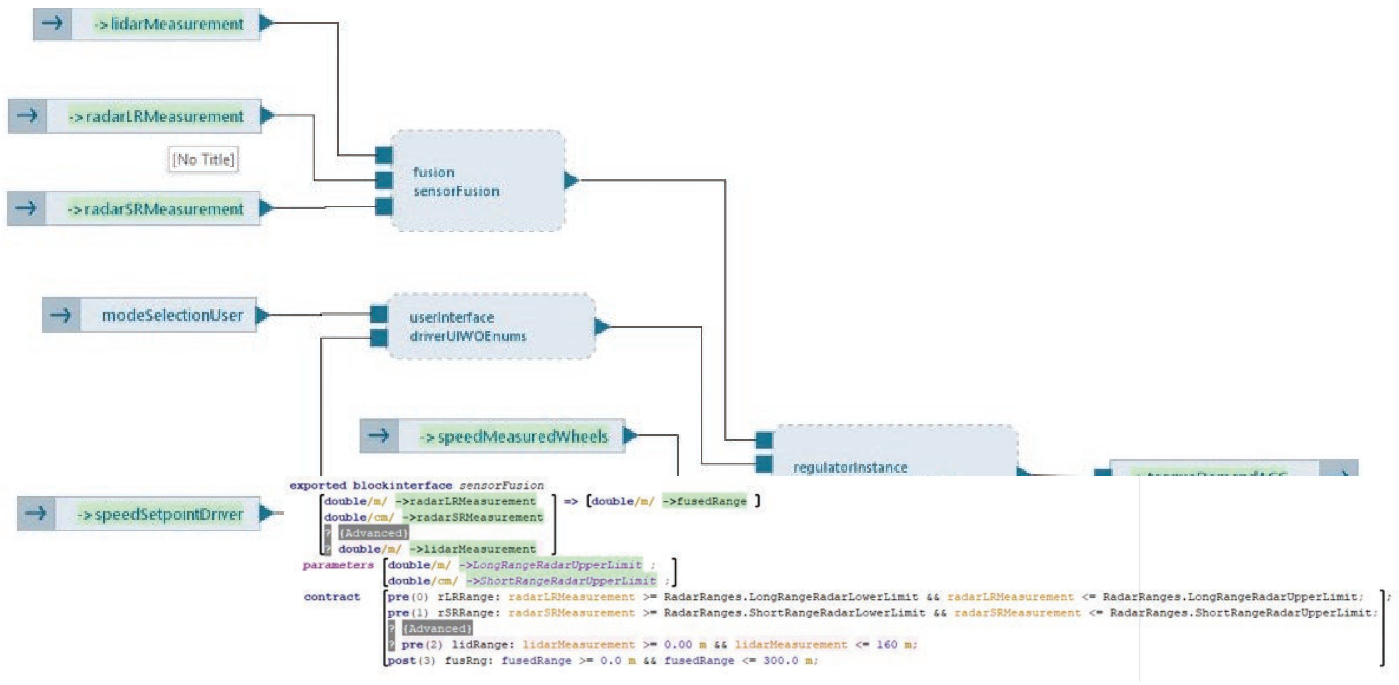
Apart from C legacy, users may have legacy architectural information available in different sources such as Simulink models, UML/SysML models, AADL and AUTOSAR. Capital Software Designer offers importers for all these sources and can merge the imports with existing architecture models.

Architecture enrichment and analysis

Architecture modeling is often seen as a documentation activity only, which is a wasted opportunity. Architecture can drive implementation, verification and validation if the architecture model is sufficiently rich.

Data-flow language is the core of the embedded software architecture model (see figure 7). In the data-flow paradigm, software functions are allocated to blocks. Capital Software Designer supports re-use and composition with block interfaces and abstract blocks, and hierarchical composition with composite blocks.

Blocks exchange data through ports, which have rich type and physical unit information associated with them in Capital Software Designer.



```
exported blockinterface sensorFusion
   [double/m/ ->radarLRMeasurement    ] => [double/m/ ->fusedRange ]
   [double/cm/ ->radarSRMeasurement   ]
   [? {Advanced}                      ]
   [? double/m/ ->lidarMeasurement    ]
 parameters [double/m/ ->LongRangeRadarUpperLimit ;  ]
            [double/cm/ ->ShortRangeRadarUpperLimit ;]
 contract   [pre(0) rLRRange: radarLRMeasurement >= RadarRanges.LongRangeRadarLowerLimit && radarLRMeasurement <= RadarRanges.LongRangeRadarUpperLimit;  ];
            [pre(1) rSRRange: radarSRMeasurement >= RadarRanges.ShortRangeRadarLowerLimit && radarSRMeasurement <= RadarRanges.ShortRangeRadarUpperLimit;]
            [? {Advanced}                                                                                                                              ]
            [? pre(2) lidRange: lidarMeasurement >= 0.00 m && lidarMeasurement <= 160 m;                                                               ]
            [post(3) fusRng: fusedRange >= 0.0 m && fusedRange <= 300.0 m;                                                                              ]
```

Figure 7: Block architecture with block contract shown for the sum block.

Contracts are the formal requirements associated with a software block and are expressed in terms of pre- and post-conditions. Contracts are added to the block interfaces. Analyzing the consistency of contracts between all blocks on an architecture model frontloads the detection of design errors to the early stages, even prior to the implementation of the first application function. Note that at the level of block interface modeling, input and output units can be made generic, which means their concretization comes during the instantiation of blocks following an interface.

Timing needs of software functions is another architecture enrichment area supported by Capital Software Designer. From the period, offset and deadline properties, as well as statistics of function execution length, the software can be used to schedule all functions on a diagram, which can be assessed prior to the target scheduling.

External or legacy architecture models can be imported into Capital Software Designer, which supports various tools and languages, including SysML. These models can then be moved to rich architecture, thus enabling you to edit, refactor, enrich and analyze them. In addition to the data flow, alternative SysML-like block definition diagram (BDD) views on the architecture are also provided, making it easier for system engineers to transition from system architecture to a detailed software architecture.

Test definition is another activity that belongs in the early stages of architecture definition.

Openness for implementation

Onboard software architects cannot simply hand over their work products to developers. They need to communicate the rationale behind their architecture, remain approachable to change requests through their company's channels and processes, and assess returning implementation work products. All this holds true regardless of where the development team is located, which may be another department in their company or at a supplier.

To preserve the openness of the chosen implementation paradigm, architectural elements are exported as rich implementation templates to be shared with suppliers. A meaningful solution honors the fact the appropriate implementation paradigm varies depending on content.

For example, model-based design with automatic code generation is frequently appropriate to implement signal processing and control algorithms, whereas hardware drivers, ECU state management and diagnostic functions are often best when they are handwritten.

Capital Software Designer supports the simultaneous use of all these paradigms in the same project. The architecture exporters currently support the following programming and modeling languages:

- C
- Simulink

In case the user selects C language, code and header files with all necessary information such as statements, macros and function interfaces are generated. Rich comments trace the code to Capital Software Designer blocks, explain the available data types; their units and meaning, as well as the block contracts the developers must obey. The function bodies contain special comments that protect handwritten implementations when re-exporting the templates due to interface changes.

When Simulink is the implementation tool of choice, the template comes in the form of a Simulink model representing the blocks, ports and connections. In both cases, templates are enriched and linked to architecture elements to allow change management and efficient integration.

In addition, Capital Software Designer makes it easy to package implementation templates with associated dependencies with internal or external suppliers.

Integrate, verify, validate, qualify: sanitizing supplied code

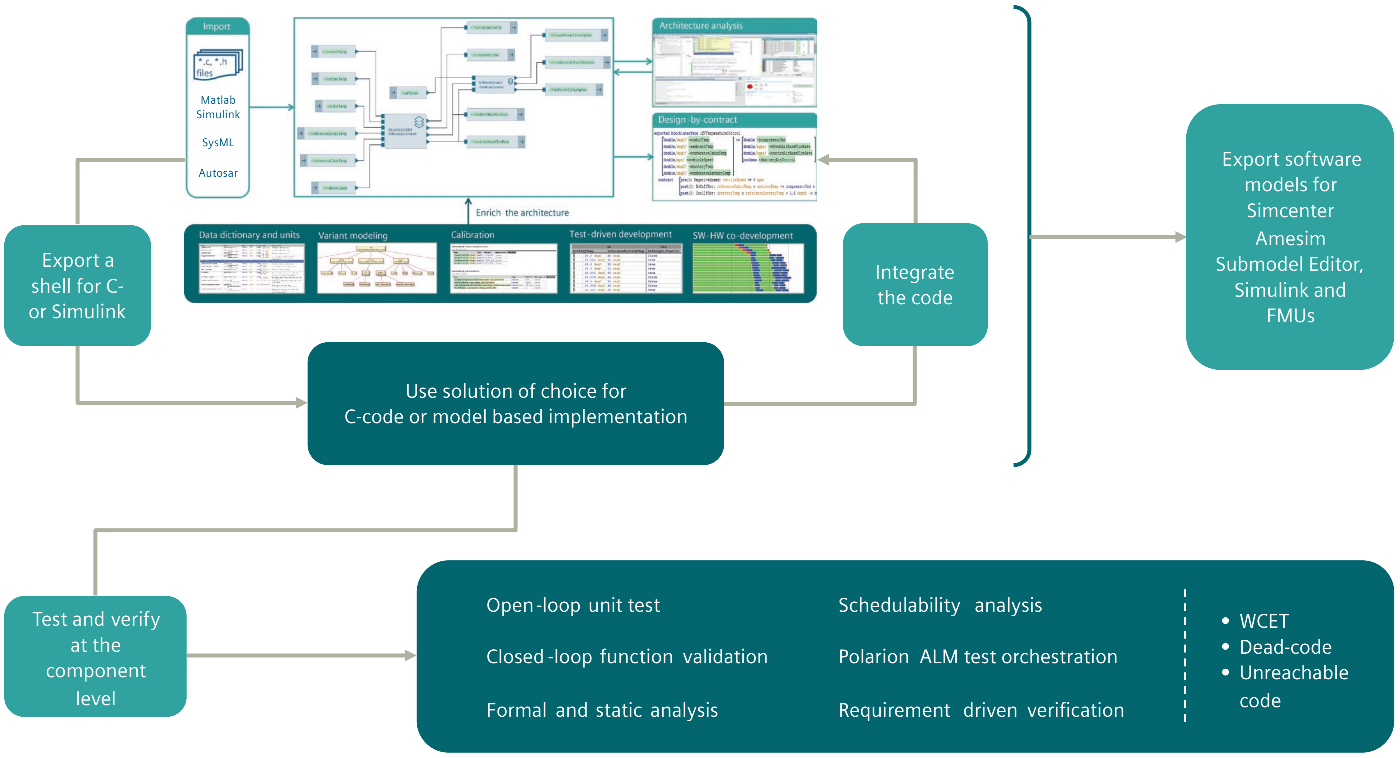

Import
*.c, *.h files
Matlab Simulink
SysML
Autosar
Architecture analysis
Design-by-contract
Enrich the architecture
Data dictionary and units
Variant modeling
Calibration
Test-driven development
SW-HW co-development
Export a shell for C- or Simulink
Integrate the code
Export software models for Simcenter Amesim Submodel Editor, Simulink and FMUs
Use solution of choice for C-code or model based implementation
Test and verify at the component level
Open-loop unit test
Closed-loop function validation
Formal and static analysis
Schedulability analysis
Polarion ALM test orchestration
Requirement driven verification
WCET
Dead-code
Unreachable code

Figure 8: Integration, verification, validation and qualification capabilities of Capital Software Designer.

Rich architecture is leveraged to drive the integration, verification, validation and qualification stages of onboard software development. Capital Software Designer capabilities include sanity checking of delivered source code based on the architecture model, unit testing, formal verifications, static analyses, timing analyses, integration with Polarion ALM for test management and requirement-driven verification, and system validation using closed-loop simulations as summarized in figure 8.

Integration capabilities

Shipped code may or may not adhere to the interface specifications provided in the implementation templates. The first integration step is therefore an automated check to determine whether the supplied code adheres to the interfaces. Capital Software Designer identifies integration issues arising from broken interfaces.

Test-based verification capabilities

After checking the interface accuracy of shipped implementations, functional correctness is the next thing to check. Testing is the dominant method in functional verification in practice today. Capital Software Designer supports execution of block unit tests and reporting of test results to Polarion ALM. Furthermore, Capital Software Designer supports automatic generation of test vectors to achieve full test coverage according to popular coverage metrics, such as branch coverage and modified condition/decision condition (MC/DC) coverage. Thereby, Capital Software Designer leverages the enriched architecture in the test and verification stage.

Formal verification capabilities

Although testing is popular and well understood, having data-flow contracts in a software architecture opens more powerful possibilities based on formal analysis methods.

Capital Software Designer makes the use of formal methods easy; for example, to prove the compliance of a block implementation with its contracts. If violations are possible, they are reported as counter examples. These sample executions of the program are helpful for the developer to understand and repair the underlying implementation errors. Furthermore, it is possible to analyze for robustness of floating-point arithmetic and unreachable code with static analyses.

The formal contract checking capability is available for software with architecture that is part of Capital Software Designer. The static analyses are applicable both for code, which has architecture that is part of Capital Software Designer, and for external code, such as legacies that have not been imported, libraries and firmware stacks.

Closed-loop validation capabilities

Capital Software Designer supports closed-loop simulations of its onboard software according to the software-in-the-loop (SiL) paradigm by integrating with two simulation platforms:

- Simcenter Amesim™ software
- Simulink

These external tools control the simulation process. The Capital Software Designer model closed-loop harnesses and test cases generate the external closed-loop simulations and presents the simulation results.

Code qualification capabilities

Prior to production use, it is also important to know whether source code meets internal coding styles, follows coding best practices, contains known security issues, and complies with functional safety standards. Capital Software Designer integrates with best-in-class code scanners for that purpose.

Conclusion

This white paper summarizes the digital transformation challenges facing the onboard software sector in the automotive and other industries. Driving supplier-based implementation, integration, verification and validation from semantically rich architecture enables manufacturers to rapidly deliver high-volume, variant-rich and high-quality onboard software.

We have discussed two typical development scenarios covering both C legacy-intense situations and model-based development situations.

Capital Software Designer supports the digital transformation of onboard software design organizations with its software architecture backbone and integration, verification and validation capabilities, leveraging the functionality of Simcenter system simulation solutions as well as Polarion ALM.

References

1. Jakobs, Christine, and Tröger, Peter. Quo vadis, AUTOSAR 2017. http://dblp.uni-trier.de/db/conf/gi/gi2017.html (accessed 8 15, 2018).
2. Pardessus, Thierry. "Concurrent engineering development and practices for aircraft design at Airbus." Proceedings of the 24th ICAS Conf. Yokohama, Japan, 2004.
3. Guetta, Olivier, Ishigami, Kazuhiro and Coutenceau, Emmanuel. "Renault Nissan new Software Strategy." Proceedings of ERTSS 2017. Toulouse, France, 2017.
4. Matt, Christian, Hess, Thomas and Benlian, Alexander. "Digital Transformation Strategies." Business & Information Systems Engineering 57, no. 5 (2015): 339-343.
5. Mueller, Markus, Klaus Hoermann, Dittmann, Lars and Zimmer, Joerg. Automotive SPICE in Practice: Surviving Implementation and Assessment. Rocky Nook, 2008.
6. University of Oxford. CBMC Model Checker. 07 22, 2018.

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 www.sw.siemens.com.

www.sw.siemens.com

76422-C5 8/19 C