Putting Real Production Software in the Loop, Methodologies Enabling SW Co-Development Between OEMs and Tier 1s

David Bailey, Guillaume Francois and Gregory Nice

ETAS GmbH Borsigstrasse 14, 70469, Stuttgart, Germany
David.bailey@etas.com, Guillaume.Francois@etas.com

Abstract. With software gaining importance as the main contributor both to functionality and differentiation in the automotive market place and its relevance to quality, safety and customer satisfaction, its place in the development process and the methods available to ensure short development cycles and a simultaneously high level of quality are coming under strain. Both model-based and abstracted code – not specific to the final production target, are in use in the earlier phases but these often do not provide code which is testable in a meaningful way for the final product. In this paper we will explore methodologies which allow target independent code to be produced and managed as a product within the development process – establishing clear linkage between development code and the final product and accountability and traceability throughout the process. We will leverage the increasing implementation of Autosar and proliferation of model based sw development techniques in the process.

1 Introduction

The past ten years has seen an exponential growth in the amount of software going into vehicles both to perform and improve core functionality, such as fueling, combustion control, valve actuation and breaking as well as to extend the capability of the vehicle with advanced driver assistance, connections to roadside, service bay and ubiquitous wireless infrastructure bringing the Internet in ever-closer connection with the Automotive control system. At the same time the era of "Mass Customisation" has bred a generation of consumers who increasingly look, not only to purchase the latest and best technologies but who also expect to be able to configure products in the way that suits them and their life-style. This has driven an exponential growth in product variants and as a consequence software variants that require managing throughout the product life-cycle. One would think that this would be sufficient to present challenges to an industry which has adopted electronic control as recently as the 1980's but on top of these challenges customers expect ever higher levels of safety and concern for the environment. This has also lead governments to increasingly legislate in the areas of safety, emissions and sustainability in developed markets adding to the cost-burden on Western OEMS to get their products into markets. Whilst there is no doubt that high and ultra-high end technology is coming to the market in the shape of Teslas, Bugattis, and other high-end products, there is also a diametrically opposed trend occurring in parallel for simplicity and functionality both with the requirement to address the needs of the first-time vehicle purchaser in India or China and with a new generation of consumers in the West where the vehicle no-longer represents a status symbol but rather is regarded as a method for getting from A-B with as little fuss as possible, preferably with access to Facebook and Twitter.

To address all of these demands simultaneously is the reason why the Automotive industry must embrace a paradigm shift in its thinking, costing and planning for product development. The differentiators in the market are increasingly software based as are the potential risks of damage to brand and future viability for firms if Software is not managed with the same regard for safety and quality as any other component in the vehicle. In addition the OEM is increasingly playing the roll of systems integrator not only on the vehicle level but also at the level of individual ECU. Integrating software developed, in house, by a tier 1 and by 3rd party specialized suppliers in one mechatronic system. This fact also drives a demand for a consistency of architecture, interfaces and lifecycle management to ensure quality and traceability in the entire process

2 From V-Cycle to Model Based to Agile.

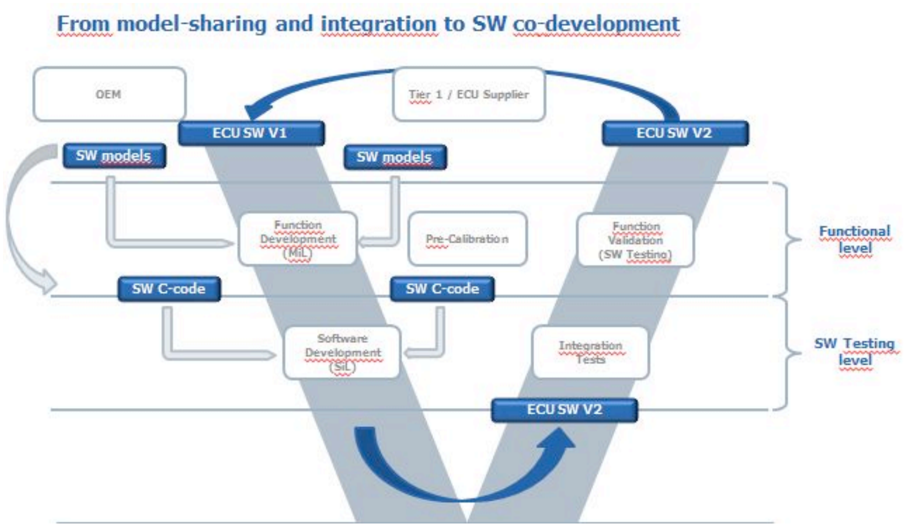


Fig. 1. From the V-Cycle to “Agile”

Considering both the driver for time to market and the requirement to manage, integrate and test software components to the required quality levels has necessitated both Tier 1s and OEMs to restructure their development processes in significant ways. From a high level requirement a feature or function within the system is typically modelled. Since there is no universal modelling tool – this typically involves a range of modelling tools in complete system, ranging from specialized environments such as Ricardo WAVE or GT-Power from Gama Technologies, to Simulink to C-code. There then follows an iterative development process around the function where the model function is refined until such time as the functional developer is satisfied that the desired functionality is reliably performed. At this stage it is not a given that the Functional model is perfect – rather that it appears perfect in the environment under which it is tested. Therefore it is essential to manage within this phase; model variants, configurations, tool versions and data-sets so that the finalized functional model can be reproducibly brought back into a further iteration step if an issue is found during the further development and validation stages.

Here we are already at a level where most OEMs and Tier 1s widely collaborate in the software development process. Namely sharing IP and functional specification on a model level. A desired function is developed by an OEM. It is tested in a Model-in-the-loop-environment with the associated plant, behavioural and environmental models. The Functional Model is then provided to the Tier 1 for further integration testing and is either converted into a further, more suitable form for model-based code generation directly or hand-coded in C before integration within the software build process for a specific ECU and target processor.

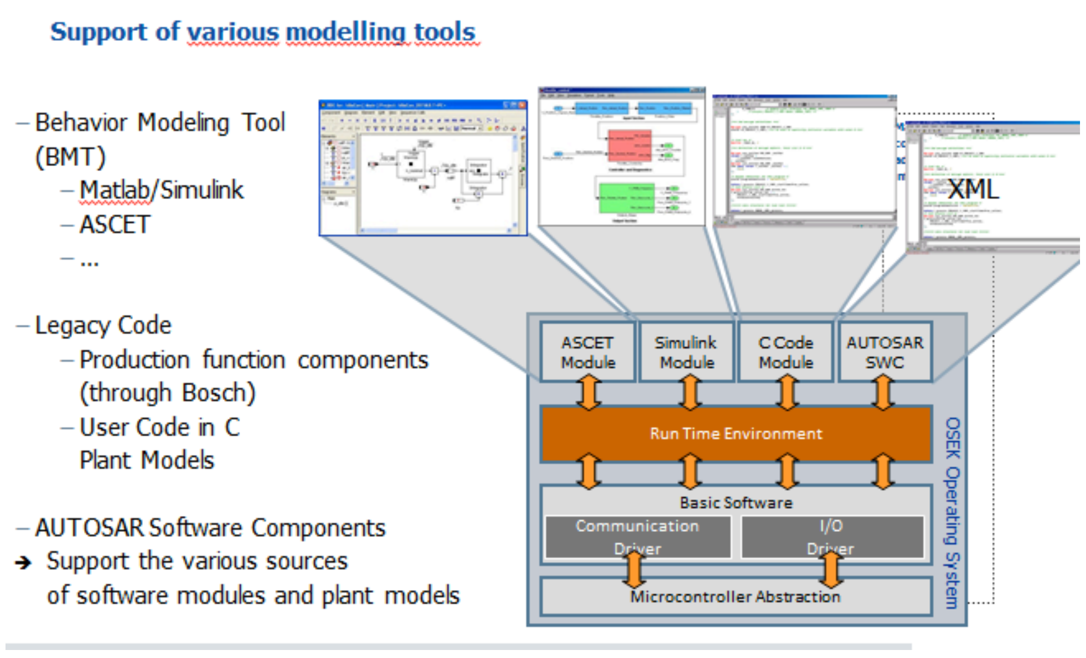
The challenges here for process integrity and traceability are significant.

Firstly the range and variety of models, tools, versions, configurations and associated data-sets is both broad and numerous.

Secondly the tested function model – which acts as a specification for the Tier 1 omits information and configuration steps with the rest of the software system and potentially also the mechatronic system that may have an effect on the specified feature "in-situ" within the overall system. Of these, OS behavior, interaction with other SW components within the system, for example diagnostic modules, execution order and quantisation effects are frequently areas where discrepancies between the "perfect" behaviour in the function is followed by implementations of the series-intent code which end up behaving differently in the target.

This fact has driven both the drive for common ECU SW architectures such as Genivi and more importantly for control systems, AUTOSAR and a range of tooling that enables pre-Autosar architecture to be tested in a common environment that can be shared between both OEM and Tier 1 that takes account as many of the areas for potential divergence in functional behavior between the target and the pure model-level environment.

3 The Virtual and Rapid Prototyping Environments



Support of various modelling tools

- Behavior Modeling Tool (BMT)
  - Matlab/Simulink
  - ASCET
  - ...
- Legacy Code
  - Production function components (through Bosch)
  - User Code in C
    Plant Models
- AUTOSAR Software Components

➔ Support the various sources of software modules and plant models

Fig. 2. A Virtual and Rapid Prototyping Environment

The requirements that therefore arise out of 1) The desire to develop in an agile way to improve quality and time to market and 2) The requirement to share and validate implementations of the functional specification in a meaningful way, lead to the requirement and productive use of Virtual and Rapid Prototyping environments at both Tier 1 & OEM. For these a number of essential features apply.

- Functional, behavioural, plant and environmental models need to be integrated in a common environment.
- This environment needs to provide features which allow the testing of changes to configurations in the OS configuration and other sw modules within the system in a manner which is as close as possible to the behavior which will be seen in the final target.
- A seamless transition between the prototype code and a real control system is also highly desirable to enable a functional test within a Real-Time active control system and potentially a base target ECU where the ECU code itself is not complete or not yet modified for the intended use case. In this case the environment should also support compiling to code for computing modules which process the function in parallel to the existing code

running on the ECU by means of software hooks and bypasses which allow a first – low effort integration of the new sw function into the control systems without going through a complete ECU software build.

Virtual Prototyping	**Rapid Prototyping**
Non real-time: Runs as fast as possible or with time scale. No connection to the real world. No I/O No communication buses	Meets hard real-time conditions
Stimuli or plant model required	Interacts with the real world. Comprehensive support for peripherals, analog and digital I/O & communication buses
Used for early validation and pre-calibration on the Windows® PC on the developer's desk	Validation and calibration on the test bench or on the road

Fig. 3 Virtual vs Rapid Prototyping

4. Typical Process Steps in SW Sharing Projects between a Tier 1 and an OEM

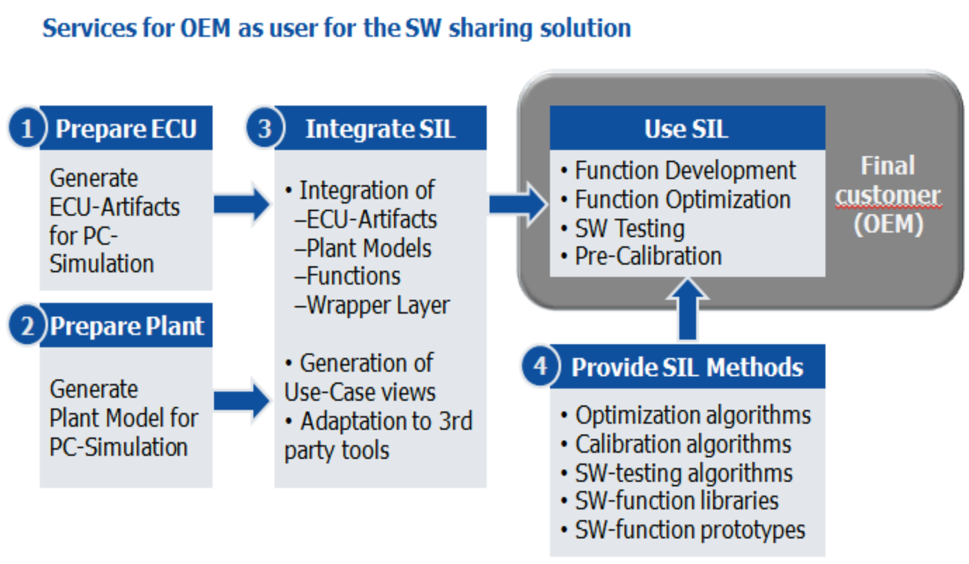

Services for OEM as user for the SW sharing solution
1 Prepare ECU
Generate ECU-Artifacts for PC-Simulation
2 Prepare Plant
Generate Plant Model for PC-Simulation
3 Integrate SIL
• Integration of –ECU-Artifacts –Plant Models –Functions –Wrapper Layer
• Generation of Use-Case views
• Adaptation to 3rd party tools
Use SIL
• Function Development
• Function Optimization
• SW Testing
• Pre-Calibration
Final customer (OEM)
4 Provide SIL Methods
• Optimization algorithms
• Calibration algorithms
• SW-testing algorithms
• SW-function libraries
• SW-function prototypes

Fig. 4 Typical Process Steps in a Project employing software sharing between an OEM & Tier 1

Step 1 in the process usually involves the Tier 1. In most cases the Tier 1 will have existing tooling and methodologies for allowing their Platform software and Application software to run on a PC. In this particular case the Tier 1 uses the same OSEK OS or Autosar OS and RTE and compiles it for the PC target. If the aim is to test primarily on a functional level then software modules or groups of software modules (Functional Components or "FC"s) are compiled as .dlls capable of being integrated within the virtual prototyping environment via an Autosar RTE or an "RTE like" runtime environment that also supports legacy APIs and API calls. This is usually the role of the Tier 1 or the party which has the most responsibility in the project for SW development and integration. Again the ECU artifacts and the PC runnables need to be mapped and managed to ensure, traceability, repeatability and accountability. Frequently software components need to be stubbed to provide and interface to models or other stimuli instead of an interface to genuine HW.

Step 2. Is to prepare the plant. This involves deciding and selecting the required models and stimuli required within the environment to be able to test the SuT with the required test coverage and depth. The required data-sets, parameterization, models and other stimuli also need to be managed with the same regard as the software components mentioned above. This task is usually carried out by the party responsible for the validation environment, frequently a 3rd party with a close working relationship with both Tier 1 and OEM. However both Tier 1 and OEM can frequently assume this role entirely or partly according to where the domain knowledge for the systems being modelled in the environment lie.

Step 3. Is to integrate the environment. This entails combining the simulation and the system under test within the test environment – mapping models and other stimuli to software component interfaces and integrating other tooling required for the test procedure – for example, calibration tooling, version and configuration management tooling, bus-analysis tooling and test automation tooling. This usually lies within the competency of the party responsible for the environment, in most cases a 3rd party.

Step 4 Is to define and provide the required test strategies and methodologies. These are usually provided by a 3rd party with input as required from both tier 1 and OEM

Step 5. The operational use of the SIL system is usually and OEM activity – nevertheless as with other test activities it can frequently be completely outsourced to a 3rd Party with the OEM or Tier 1 customer providing simply a list of requirements to be tested and the Pass/fail criteria. The system itself may also be employed at different departments within one OEM with various aims and work-splits. For example for pre-calibration or use within the OEMs own functional development process where the SIL environment provides a high-fidelity replication of the system as it will operate in the final series target.

5 Bringing Autosar and Agile Together

So far we have only considered the possibilities widely in use today based on the predominant sw architectures in series production at OEMs. Today and especially in the realms of Powertrain, ADAS, Chassis and Drivetrain are predominantly based on a software architecture proprietary to the Tier 1 or to the OEM. Whilst many Tier 1s offer an “Autosar Sandbox” for their OEM customers which allow easy integration with their legacy software – the vast majority of the code in these areas is legacy. However the move to Autosar is happening and happening rapidly even in these areas and at the very minimum a very good interface is required between the legacy architecture and Autosar in order to allow both migration of functions between architectures and reliable operation in the series target where a mixed architecture is employed. As far as the SIL environment is concerned this step towards Autosar offers a range of new possibilities that were only achievable before with a level of effort prohibitory to implementation in the past.

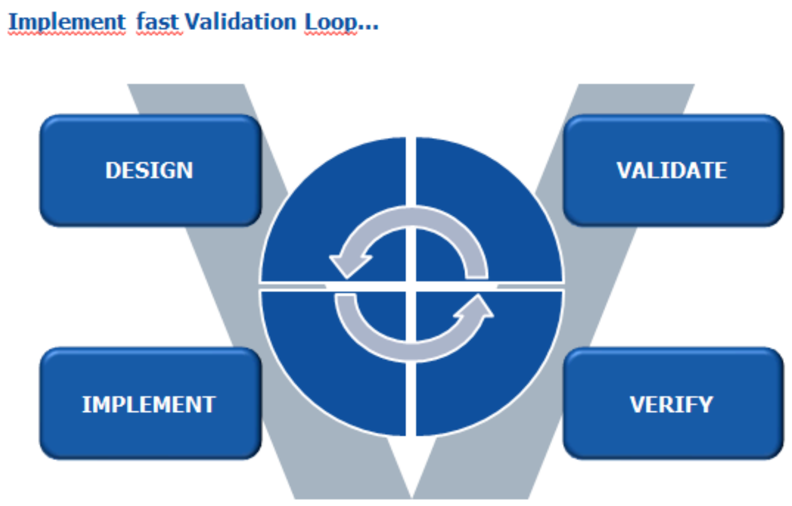


Fig. 5 Agile in Automotive is enabled by Autosar

The first major benefit brought by the shift to Autosar is that most Automotive Silicon today is delivered with a HW abstraction layer that supports the Autosar standard (ie an Autosar MCAL). As this has been so for a number of years a large amount of the code, Autosar or not, is already using a target independent API for communication with the microprocessor. This has significantly reduced the amount of stubbing required to enable code to be portable to the PC and also increased the portability of projects between processor platforms.

The RTE is non-target specific anyway, so any software in the system using the RTE for communication can be compiled 1 to 1 either for a real target or for the PC. Assuming there is a port of the OS for the PC available which allows for the same degree of configuration as on the target and an API or other communication mechanism by which other (non Autosar) Software components and elements within the environment (models, stimuli, test tools) then it is already significantly less, if not low effort to provide a build chain which is capable of taking the ECU artifacts intended for the series target and compiling these 1 to 1 for a PC based virtual validations platform. This, to the extent that it is not only possible off-the-shelf with full Autosar architectures but also extremely cost effective even with significant legacy content due to the fact that the system can be run and tested and developed against at very high fidelity, in Real Time or faster than real time. This has the corollary that many of the test activties that previously required real ECU HW, real engines and real vehicles to carry out can now be contemplated on the desktop. Thus offering a truly huge potential to cut time, effort and corresponding costs from the entire development process.

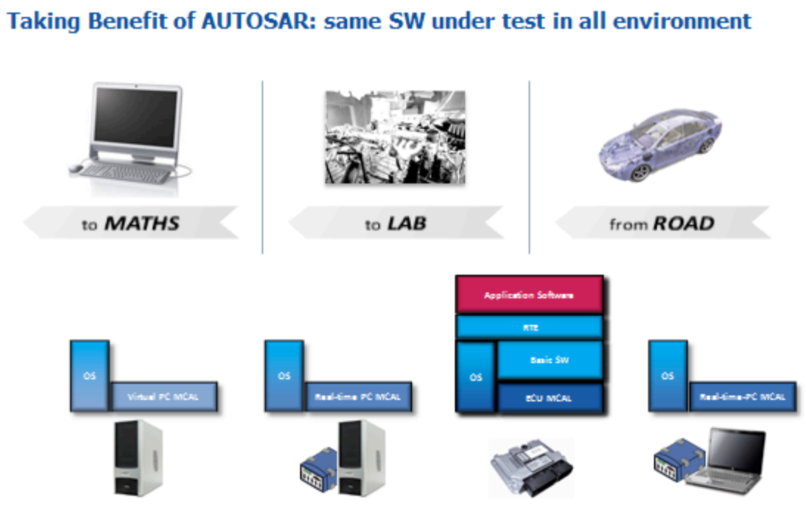


Fig. 6 The same SW can be compiled for any target at any stage of development

Aside from the obvious benefits of being able to front-load the development process, a number of other advantages emerge, some esoteric and some banal but all bringing a degree of much needed rationalisation to the exponentially growing demands within the industry. On the banal side – testing and development resources are spread throughout the world – fortunately or unfortunately there is no truly global market established so customs officials, especially in some of the lower-wage economies where development and testing is typically carried out, still have an interest to arrest, inspect and delay development and prototype ECUs and components. On average delivery time between a development centre is Germany and India ranges from 1-3 months. On the more esoteric side, once the control system is available on the PC steps to integrate this with Android, Genivi or other OS in a multicore envirnoment to address connected vehicle development becomes much more simple. Most crucially writing, testing and proving that code is safe has become key, since the systems we drive our children around in are controlled by the software that we produce.

If we consider some of the issues arising our of Philip Koopman and Michael Barrs analysis of Toyota's software presented in the case "Bookout & Schwarz v Toyota", most of the issues raised concern fundamental flaws in design and architecture that can only be address at the deeply embedded level. Therefore there is an increasing demand that the virtual validation methods we use are as close to reality as possible. In todays model-based world, which is largely regarded as state of the art there is increasing testing on that level but without the ability to dive down into the behaviour at the lower levels and here there are plenty of banana-skins lurking.

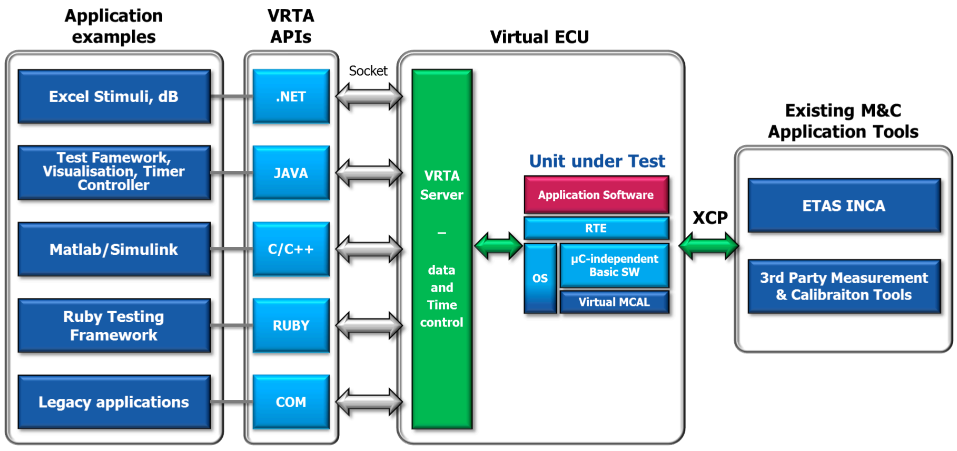


Fig. 7 APIs offering access to all levels of the sw architecture from external test tooling and simulation enviroments

6 Conclusion

The number, complexity and scope of embedded control systems in the vehicle is increasing exponentially. These systems now perform the key functions of the vehicle so software has to perform on all levels – primarily of course with regard to safety, but also with respect to fulfilling customer requirements with regard to features, differentiation and reliability. To put this in context a 737 Airliner contains 6,5 Million lines of code. The latest S-class from Daimler contains over 20 million. In terms of driving or flying hours the average platform racks up 1000x more flying hours than the entire fleet of Boeing 737s has flown since 1968. Whilst it must be conceded that not all of these systems are safety critical – the importance of introducing rapid iteration cycles and traceability throughout the development process cannot be underestimated. This can only be achieved with a paradigm shift in the industry with regard to the way software is regarded, developed and managed – with the same importance or more than any other part of the vehicle. If the automotive industry does not step up to the challenge then increasingly it is looking as if others will do so in its stead.

References

1. http://www.safetyresearch.net/blog/articles/toyota-unintended-acceleration-and-big-bowl-%E2%80%9Cspaghetti%E2%80%9D-code

2. http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf

3. Pavey & Winsborrow, "Demonstrating Equivalence of Source Code and PROM Contents", Computer Journal Vol 36, No 7, 1993

4. Charette, "This car runs on code", IEEE Spectrum, Feb 2009