Software Integration of a Safety-critical ECU: an Experience Report

Ieroklis Symeonidis, Niklas Angebrand, Kostas Beretis, Eric Envall
ArcCore AB
Gothenburg, Sweden
e-mail: {ieroklis.symeonidis, niklas.angebrand, kostas.beretis, eric.envall}@arccore.com

Abstract—**In this paper, we present a software integration methodology in accordance to the automotive software standard AUTOSAR. The case under examination is the active safety electronic control unit (ECU) of the recently developed platform, called Scalable Product Architecture, of the Volvo Car Corporation. Particular emphasis is given in the relationship between the supplier of the ECU and the car manufacturer. Efficient communication between these two parties has been a challenging issue. Therefore, specific workflows regarding the exchange of information and the overall way of working are presented. The need of a dedicated integration team acting as an interface between the two organizations is also highlighted. Finally, concrete guidelines enabling continuous integration throughout the development process are provided. Our approach contributed in decreasing the software development cycles. We strongly believe that the conclusions drawn from our work experience can be generalized up to a certain level, affecting the automotive industry as a whole.**

Keywords- automotive software; AUTOSAR; embedded systems; integration; ISO26262.

I. INTRODUCTION

The development of embedded software in the automotive domain is characterized by its complexity. The reason behind that is not only the increasing functional and safety requirements but also the fact that a significant number of subcontractors are involved in the development process. Traditionally car manufacturers (or OEMs) had the role of integrating different subsystems developed by their suppliers (Tier-1s, Tier-2s, etc.) following a "black-box" approach [1]. These systems used to be limited in scope and usually resided on a single ECU. On the contrary modern automotive systems include distributed functions with strict timing and communication requirements between various ECUs. Such functions can be found in active safety and advanced driver assistance systems of premium cars.

The challenges of software engineering within the automotive industry have been well described by Broy et al in [2]. The increasing role of software as a source for innovation and the multidisciplinary nature of the domain are highlighted. Issues regarding the integration of software components in a distributed system are presented in [3]. In this paper, a general overview of the existing challenges as well as possible solutions to design and analysis issues in automotive systems are presented. AUTOSAR [4] and its implications on the development tool-chain are analyzed in [5]. In addition, [5] also shows how the concepts of the AUTOSAR methodology can be brought together in a common tool-chain leading to a higher degree of automation in the software development. Moreover a case study regarding a way of incorporating AUTOSAR in the development process of an antilock braking system (ABS) is presented in [6]. Finally, an approach for dealing with the complexity of the development process according to specific corporate needs is analyzed in [7]. Emphasis is being given on the need of constructing a tool chain with high degree of reusability and automation. A case study regarding a tool-chain model for AUTOSAR ECU design is also presented.

Software applications produced by different vendors need to be integrated into the final software for the ECU. The development of such software is an iterative process, consisting of multiple releases with parallel lifecycles. The purpose of our work is to present a well-defined software integration process and the way it should be aligned within the development process. These guidelines are derived from our experiences as a software integration partner for Volvo Car Corporation. Detailed information about roles and workflows as well as the flow of information between the OEM and the ECU supplier throughout the whole process will be provided. Our proposed workflow allows the exchange of key information between the two partners while at the same time it protects each side's intellectual property.

The remainder of the paper is organized as follows. In the following section, we provide the technical background of our paper. Section 3 presents the software development process and section 4 describes our approach towards software integration. Finally, we draw our conclusions and outline future work.

II. TECHNICAL BACKGROUND

In this section, we are going to identify the key stakeholders that were involved in the development process. Also, we are going to present the automotive standards that influence the development process. In this project, there are three interacting parties:

- **OEM:** The main job of the OEM is to develop functions (e.g., Lane Keeping Aid, Collision Warning). A *software architect* defines the structure of the part of the software that will implement these functions. The implementation is assigned to *function developers,* who use Matlab/Simulink as a development tool. The developer of each function is responsible for his distinct part of the code/functionality. The *system testers* are responsible for testing and verifying that the system conforms to a certain behavior.

- **Tier-1:** The responsibilities of the Tier-1 include both the hardware and the final software (including the OEM's advanced functionality). In particular, the Tier-1 designs the ECU hardware, performs OS configuration and implements its own functions. These functions address communication with peripherals (e.g., sensors) and basic functionality such as diagnostics. The same stakeholders (*software architect, function developers* and *system testers*) can be identified within the Tier-1.
- **Software integration team:** ArcCore's software integration team resides inside the OEM. Our team acts as an interface between the two organizations enabling effective communication between all the stakeholders on the appropriate level of abstraction. Our approach increased the efficiency of the development leading to shorter time to delivery and reduced cost. A major challenge of our software integration team was also to establish a work flow and an automated tool-chain. This tool-chain consisted of requirements management tools, code generation tools, AUTOSAR authoring tools, compiler and linking tools as well as general purpose tools like data repositories and build servers. Under this work environment it is clear that the responsibilities of the software integration team can be quite broad, requiring various competencies. The range of activities covered could span from developing glue code or gateway functionality, up to specifying to a component supplier the system functionality to which the component must conform [8].

A. AUTOSAR

AUTomotive Open System Architecture is a software architecture standard developed jointly by automotive manufacturers, OEMs and tool developers. This standard was created to satisfy the need for standardization of basic software and the interfaces to applications/bus systems [9]. The motivation behind that was to reduce system complexity and keep the development cost feasible. Some additional goals of AUTOSAR include the scalability across different vehicles and platforms, maintainability throughout the product lifecycle and the sustainable utilization of natural resources [10].

AUTOSAR follows a layered architecture, where hardware, basic software, runtime environment and application software are separated from each other [11]. The basic concepts of AUTOSAR are the Software Component (SWC), the Runtime Environment (RTE) and the Basic Software (BSW). Each SWC should be assigned to one ECU and encapsulates part of the functionality of the application [12]. The implementation of the SWC is independent of the underlying platform, following the basic design concept of separation between layers. The RTE provides a communication abstraction to the SWCs connected to it, providing the same interface and services both for inter and intra ECU communication. Since the requirements of SWCs running on RTE may vary, different ECUs may have different RTEs. The BSW is essential to run the functional part of the software. It is the standardized software layer, which provides services to the SWCs [11]. It contains both standard and ECU specific components.

Although this model based approach is a step forward in reducing the complexity of the development process, there are certain limitations. AUTOSAR methodology does not include topics like requirements management, hardware development and build management. Therefore, it does not cover the complete development process lifecycle [5]. Furthermore AUTOSAR does not standardize test procedures [5]. Finally, AUTOSAR neither defines concrete guidelines and procedures for development strategies to be followed nor separates distinctly the activities in the various development phases [13]. From the above it becomes clear, that there is not a universal way of working with the standard but only case specific implementations like the one addressed in this paper. The way of working can be subjective and highly dependent on the developer's work experience and interpretation of the standard. Therefore, a well-defined development process is of great importance.

B. ISO26262

ISO26262 [14] is the standard for functional safety management of electrical and or electronic systems within the automotive industry. It applies to all development activities of safety-related systems (electrical, electronic and software) and addresses possible hazards caused by malfunctioning behavior of such systems, including their interaction. It consists of 10 parts, each one dealing with a specific development activity. Parts 6- "Product development at the software level" and 9-"Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses" are of great importance for our work. Part 6 highlights the importance of performing safety analysis at the software level and suggests some mechanisms for error handling and detection at a generic level. However, no clear guidelines are provided leading to subjective implementation in industrial practice [15]. Part 9 provides a classification mechanism for hazards according to ASIL. The ASILs can have the following values "QM, A, B, C, D" where D requires the most attention and QM the least due to a combination of potential severity, controllability and exposure of hazards [16].

ISO26262 provides guidance to identify the level of effort required to achieve the desired level of functional safety [17]. It can also be viewed as a defense against liability claims and it is not a certification requirement [17]. Furthermore AUTOSAR only provides mechanisms to support functional safety on a software level and does not guarantee any functional safety properties of the final system [16]. The key notion that brings together the two standards is "freedom of interference". By partitioning the system into safety related and non-safety related components it has to be assured that there is no interference between the safety related ones and the rest of the software, or that it is reliably detected. Memory partitioning provides spatial freedom of interference, while other techniques like implementation of a watchdog manager provides temporal freedom of interference. Finally, a way of guaranteeing correct exchange of information is through end-to-end communication protection mechanisms. However, IS026262 does not explicitly address AUTOSAR. Therefore, the selection and implementation of any safety mechanism is a responsibility of the AUTOSAR vendor.

III. Software Development Process

The case under examination is an active-safety ECU with two parties involved in the development of the software. Since both parties need to protect their intellectual property, only the compiled version of the source code is exchanged between them (i.e., object-code). Along with the object code a definition of the software structure and its interfaces is supplied in the form of ARXML files. This highlights the importance of a dedicated software integration team, able to combine object code from both sides with a common system extract into unified software. The system extract contains the software structure and the interfaces as well as all service and integration information needed by the software integration team, as defined by the AUTOSAR standard.

An automotive software development project consists of several internal iterations/releases. In our case, each release was divided into the following phases: contract, function integration, testing and verification and short-loop phases (Figure 1).

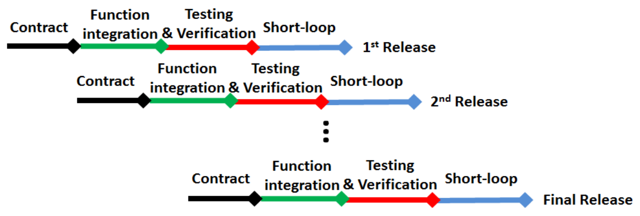

Figure 1. Software development process.

A. Contract

AUTOSAR defines a Composition SoftWare Component (C-SWC) that contains one or several Atomic SoftWare Components (A-SWCs), which we will refer to simply as SWCs. The system has a root composition, which contains one composition for the OEM SWCs and one for the Tier1 SWCs (Figure 2). Both sides need to define the interfaces between their compositions and to the external signal busses available for the ECU. This is done by exchanging contracts in form of a preliminary system extract (Figure 3). At this phase software architects on both sides need to provide an initial software structure for the SWCs containing interfaces for sending and receiving signals as well as interfaces towards the diagnostic services. In the contract phase not all details about the final SWC need to be defined. It is possible to add more information in an iterative manner. Usually in the automotive domain, external bus interfaces need to be defined early in the design process and remain unchanged (frozen) until the next release of the software. At a later stage the interfaces towards services like diagnostics need to be also frozen. In order for several parties to be able to work in parallel it is important to freeze the composition interfaces and the service interfaces at the same time.

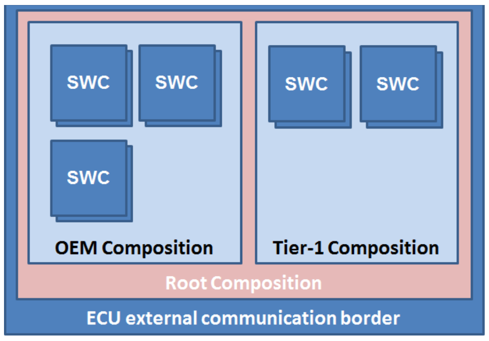

Figure 2. Software composition.

In order to validate the initial software structure, an RTE generation is performed by each party. RTE contracts are generated for each SWC during the contract phase. At this point the SWCs consist of a basic structure with no functionality, we call them SWC shells. To be able to make an RTE generation a preliminary BSW configuration is needed. The purpose is to validate and identify incompatibility issues in the initial structure. Depending on the completeness of the BSW configuration there might be errors/warnings at this phase. The cause of these errors/warnings must be identified by the software integration team and reported to the Tier-1. As an additional validation step the software containing only SWC shells is compiled. This helps to identify errors related to the source code.

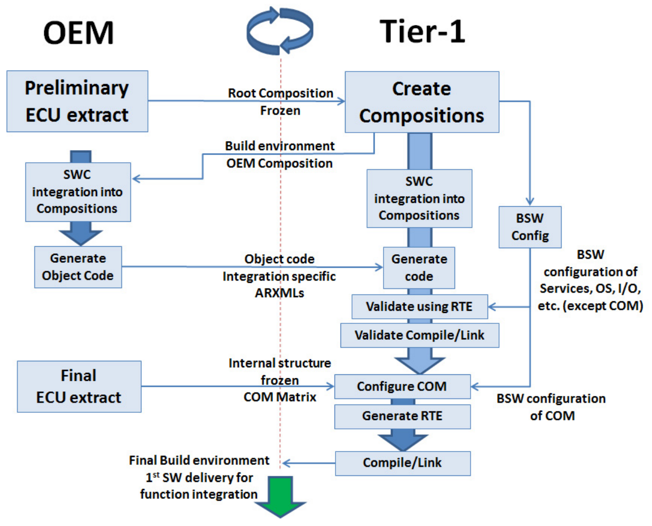
OEM
Tier-1
Preliminary ECU extract
Root Composition Frozen
Create Compositions
Build environment OEM Composition
SWC integration into Compositions
SWC integration into Compositions
BSW Config
BSW configuration of Services, OS, I/O, etc. (except COM)
Generate Object Code
Object code Integration specific ARXMLs
Generate code
Validate using RTE
Validate Compile/Link
Final ECU extract
Internal structure frozen COM Matrix
Configure COM
BSW configuration of COM
Generate RTE
Compile/Link
Final Build environment 1st SW delivery for function integration

Figure 3. Contract phase.

After the successful generation of the RTE, the SWC shells are delivered to the function development team. In order to enable continuous integration we produce a dummy function for each SWC. This marks the beginning of the function integration phase (Figure 4).

B. Function Integration

At this stage the function developers introduce their functionality in the SWC shells. Any integration issue that might occur is resolved by the software integration team and a new shell is generated. To be able to deliver object code, the integration team needs a properly configured build environment, which is the responsibility of the Tier-1. More specifically RTE generation needs to be error-free, as well as BSW modules like OS and COM need to be configured properly. The key for continuous integration is that the code always builds. This is guaranteed by the initial dummy functions, which enable the software integration team to successfully build software regardless of the development state of a specific function. Functions are integrated gradually. Successful integration of a function is indicated by a successful build of the software. Once all the functions of the specific release are integrated, the produced software is delivered for testing on target. It is of great importance to verify that both sides use the same build environment to produce code. Therefore, the Tier-1 delivers the build environment at the end of this phase, having incorporated all the possible changes introduced during function integration. An example of such a change could be the mismatch of the linker script due to changes introduced in the memory sections.

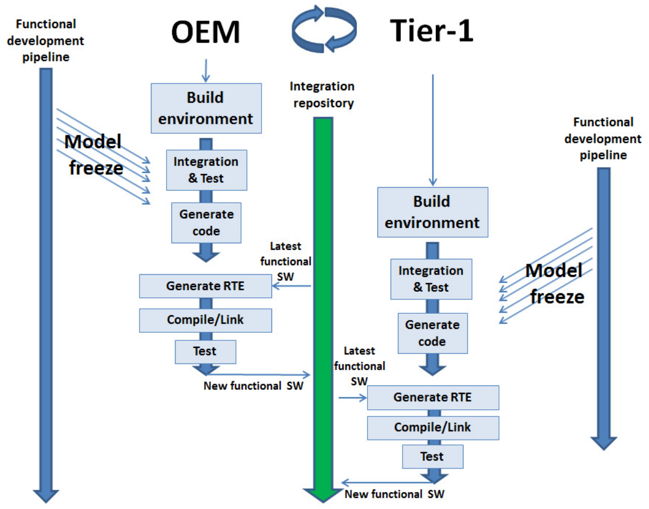

Figure 4. Function integration phase.

C. Testing and Verification

The testing and verification phase follows. This is not in the main scope of the software integration team and therefore it will not be analyzed in full detail. Dedicated teams on both sides perform testing and verification on system level, based on specific requirements. Prior to the system level tests it has to be mentioned that the function developers test their functions in simulated environments. The software integration team performs unit tests of the SWC shells and various configuration tests on the system extract. Also upon the official delivery in the form of a binary (from the Tier-1 to the OEM), a series of acceptance tests are performed. If the outcome is successful and the proper documentation is approved, then the software is available for the test vehicles.

D. Short-loop

Due to the relative long lead time from function freeze until the function is available in test vehicles there is a great need for having internal engineering releases (i.e., short-loops). This allows the OEM to speed up the function development process and detect possible bugs at an early stage. A short-loop can be performed once a build environment is setup, including the Tier-1 object code. In a short-loop build, new source code from the function team is introduced in order to build complete new software. As long as the internal structure changes and the border of the compositions remains the same software with the new functionality is produced. Any changes introduced must be compatible with the given BSW configuration.

IV. INTEGRATION APPROACH IN THE CONTEXT OF AUTOSAR

The abstraction of AUTOSAR can, with great benefits, also be extended into the function development domain. This is done by supporting the function development with SWCs that encapsulate the pure functionality into a functional library and adding AUTOSAR helper components that take care of the AUTOSAR properties (Figure 5). Depending on the implemented functionality, each SWC may require different helper components.

This workflow comes with multiple gains. The functions can be developed and verified in a different environment (for example Matlab/Simulink) without any AUTOSAR dependencies. As mentioned earlier, the SWCs can always be provided with a dummy function, which ensures that the system always builds. Another aspect is that the function developers do not need to know the AUTOSAR details and can keep their focus on function development.

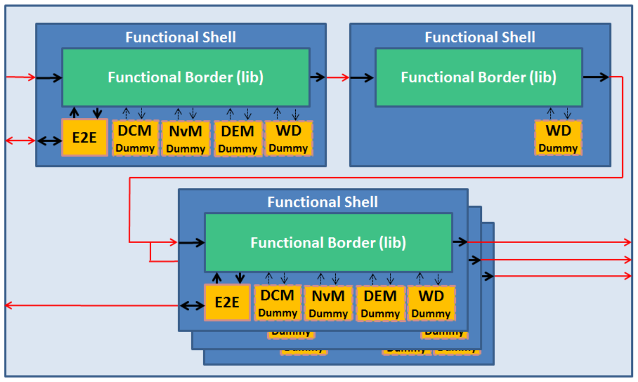

Figure 5. Functional composition under AUTOSAR context.

The usage of a dedicated database for the needs of the OEM's software architect was also introduced. All the system design related information (e.g., signal interfaces, diagnostic services) can be stored in this database. This enables the software architect to model the system in a lightweight fashion, thus providing a higher abstraction level regarding AUTOSAR. Using the information stored in this database, the software integration team can generate the AUTOSAR definition of the system in the form of ARXML files.

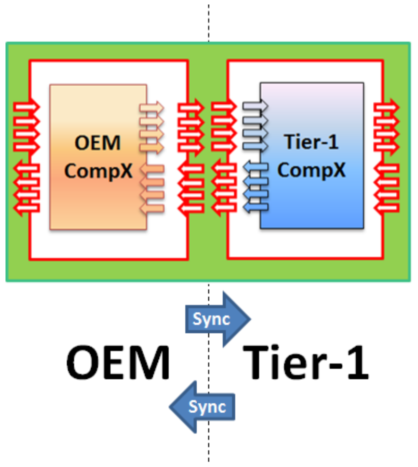

Figure 6. Exchange of information.

For the successful integration, the following rules were established for the exchange of information between the two sides. This is essential in order for both sides to have a synchronized view of the overall software structure (Figure 6). The OEM defines and owns the borders of both compositions (red boxes). Each side defines its composition and internal SWC structure including intra connections. The Tier-1 must make sure that its composition matches the defined border. This information combined with the signal database (defined by the OEM) leads to complete system extract that can be used for the development process. The supplier's border can only be changed by mutual agreement (change request).

V. CONCLUSIONS AND FUTURE WORK

Throughout this case study we illustrated our approach for reducing the complexity involved in the development process of an automotive embedded system. The main challenges in such a process are the interaction between the development partners, the variety and sometimes incompatibility of the tools involved, as well as the subjective implementation of the dominant automotive standards such as AUTOSAR and ISO26262. We presented a proven-in-practice software development framework according to the needs of the AUTOSAR standard. The interaction between the OEM and the Tier-1 becomes much more efficient and at the same time intellectual property is protected. The key element for the successful interaction is the software integration team, which has a broad variety of responsibilities as described earlier. This team may be part of the OEM or could alternatively be a third partner working for the OEM like in our case.

Furthermore concrete guidelines enabling continuous integration in the context of AUTOSAR were provided. In this way, the function development gets decoupled from any AUTOSAR constraints. This leads to shorter development cycles and consequently to a faster time-to-market for the final vehicle. According to the “Driver Support and Software Integration” manager of the Volvo Cars Corporation, the time for producing a short-loop has decreased “from several days to about an hour”. He also stated that, “the AUTOSAR interface specification time has decreased from three months to less than two hours”. Previously this process was manual, involving several engineers, while now it is fully automated.

However, there is still room for improvement. Certain adaptations of the existing tool-chain are needed, in order to deal with incompatibilities between different tools. Ideally this tool-chain should fit into any automotive development environment. Finally, we also plan to implement an AUTOSAR-compliant testing framework for function performance measurement and debugging on target, based on actual log data from test vehicles. In this way possible bugs related to actual implementation that were not detected through simulations can be recreated on a development board with the same microprocessor. With this approach we reduce the need to utilize test vehicles, which are limited in number and might not be available.

ACKNOWLEDGMENTS

We would like to express our appreciation to all people at the Active Safety department of the Volvo Car Corporation and especially to Daniel Levin, for their collaboration throughout this project. Furthermore we are also sincerely grateful to Postdoc researcher Risat Pathan from Chalmers University of Technology for his insightful advice.

REFERENCES

[1] H. Heinecke, et al., "Software Components for Reliable Automotive Systems," in Design, Automation and Test in Europe, 2008. DATE '08, 2008, pp. 549-554.

[2] M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann, "Engineering Automotive Software," Proceedings of the IEEE, vol. 95 , 2007, pp. 356-373.

[3] M. Di Natale and A. L. Sangiovanni-Vincentelli, "Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools," Proceedings of the IEEE, vol. 98, 2010, pp. 603-620.

[4] AUTOSAR development partnership, "AUTomotive Open System ARchitecture". [retrieved: March, 2015]. Available: http://www.autosar.org

[5] S. Voget, "AUTOSAR and the automotive tool chain," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010, 2010, pp. 259-262.

[6] T. Hermans, P. Ramaekers, J. Denil, P. D. Meulenaere, and J. Anthonis, "Incorporation of AUTOSAR in an Embedded Systems Development Process: A Case Study," in Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on, 2011, pp. 247-250.

[7] M. Biehl, J. El-khoury, and M. Törngren, "Automated Tailoring of Application Lifecycle Management Systems to Existing Development Processes," International Journal on Advances in Software, vol. 6, 2013, pp.104-116.

[8] P. Wallin, J. Froberg, and J. Axelsson, "Making Decisions in Integration of Automotive Software and Electronics: A Method Based on ATAM and AHP," in Software Engineering for Automotive Systems, 2007. ICSE Workshops SEAS '07. Fourth International Workshop on, 2007, pp. 5-12.

[9] S. Furst, "Challenges in the design of automotive software," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010, 2010, pp. 256-258.

[10] AUTOSAR development partnership, "AUTOSAR-Technical Overview". [retrieved: March, 2015] . Available: http://www.autosar.org/about/technical-overview/

[11] D. Diekhoff, "AUTOSAR basic software for complex control units," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010, 2010, pp. 263-266.

[12] B. Huang, H. Dong, D. Wang, and G. Zhao, "Basic Concepts on AUTOSAR Development," in Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on, 2010, pp. 871-873.

[13] C. JungEun, L. DongSun, and L. Chaedeok, "Process-Based Approach for Developing Automotive Embeded Software Supporting Tool," in Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on, 2009, pp. 353-358.

[14] International Organization for Standardization, "ISO 26262-10:2012," ed, 2012. [retrieved: March, 2015]. Available: https://www.iso.org/obp/ui/#iso:std:iso:26262:-10:ed-1:v1:en

[15] V. Bonfiglio, L. Montecchi, F. Rossi, and A. Bondavalli, "On the Need of a Methodological Approach for the Assessment of Software Architectures within ISO26262," presented at the SAFECOMP 2013 - Workshop CARS (2nd Workshop on Critical Automotive applications : Robustness \& Safety) of the 32nd International Conference on Computer Safety, Reliability and Security, Toulouse, France, 2013, pp. 51-56.

[16] K. Hyungju, R. Itabashi-Campbell, and K. McLaughlin, "ISO26262 application to electric steering development with a focus on Hazard Analysis," in Systems Conference (SysCon), 2013 IEEE International, 2013, pp. 655-661.

[17] B. Böddeker and R. Zalman, "AUTOSAR at the Cutting Edge of Automotive Technology," in 7th International Conference on High-Performance and Embedded Architectures and Compilers, 2012. [retrieved: March, 2015]. Available: http://www.hipeac.net/system/files/zalman-keynote.pdf