Mälardalen University
School of Innovation Design and Engineering

Vehicular Software Engineering: Understanding the State of the Art

Executive Summary

This annotated bibliography was developed as a means of creating a baseline of the state-of-the-art in vehicular software engineering. When we started the literature review for the project, the focus was on automotive software engineering, and as the review progressed, we realized the importance of expanding the scope to vehicular software engineering. Each of the three broad subsectors within the vehicular sector: automotive, trucks/buses, other specialty equipment; provide different insights into the challenges created by the use of software. Within the three sectors, the automotive sector has the highest volume, and at the same time is driven by the conventional pressures of lowering unit costs. The trucks/buses sector provides the greatest variation in software configurations since each buyer attempts to tailor the software components to their unique requirements. The specialty equipment sector has the least variance, but still leverages software to be a product differentiator.

The literature review was conducted in two phases: the broad overarching review consisting of 152 papers drawn from SAE, IEEE, ACM and LNCS publication, and a more focused development of the annotated bibliography of 31 papers. The annotated bibliography starts with a presentation of the key technical and managerial challenges that emerge with the increased reliance on software to drive innovation in the vehicular industry. This use of software is challenging the conventional industrial structure and way of doing work, and bringing to the forefront the limitations of a lack of research & development in software enable systems (as one of the authors noted, software R&D was historically treated as a 'cost' item).

The current state of software usage in the industry highlights the immaturity of practices and processes, that have struggled to stay current in the face of rapidly evolving technology. In a merging of Moore's law and Murphy's law, software usage has increased rapidly, but even the 36 month doubling in KLOC software estimates cannot keep up with the 18 month cycle time in hardware evolution (Moore's law).The emphasis on supplier-driven innovation for software-intensive systems has changed the relationship between the OEM and the supplier from a power imbalance favoring the OEM to a more 'balanced' sharing of power. One of the challenges of this increased use of software is the payment model used to procure services: the supplier is currently hired on a 'time and materials' based contract, and not on the value of the component delivered. Specifying and assessing 'value' delivery between customers and suppliers remains an open and challenging area for research. Furthermore, the business models associated with the creating, marketing and sustainment of vehicular software remain unclear. As was pointed out during the discussion at the seminar wherein this was shared with other members of the academic community, it was pointed out that in so far, the OEms do not have a business model that allows them to explicitly & profitability sell "software" to their customers.

The architectures that we use on vehicles today have evolved to an integrated modular system (very similar to the evolution in aerospace electronics, when we evolved from point-to-point networks connecting individual sensors, to an integrated system. This evolution was driven by the need to manage wiring complexity and weight. Open research questions exist in the areas of identifying the ideal software process model, modeling at multiple levels of abstraction and preserving meaning during translation; finding the appropriate cost model, managing the increasing number of variants in the market at any given point in time, and last but not least, supporting reuse. This increase in complexity has led to an increased focus on standards and their impact on tools & techniques. From a standards perspective, the literature focused on IEC 61508 (and for the vehicular equivalent, 26262), and the AUTOSAR. The open challenges that remain in the area of connecting and need future research includes:

- Separation between functional and architectural models
- Support for defining the task and resource model
- Modeling support for the analysis and back annotation of scheduling related delays
- Sufficient semantics preservation

From a safety standpoint, there has been an increasing call to certify automotive software, for which standards such as ISO- 26262 are under development. There are increasing efforts to leverage approaches such as model-based development to develop the safety cases in a modular fashion – this is an area that needs further research. From a technology standpoint, the challenges that remain unanswered are all of the 'traditional, open' software engineering questions plus questions that combine these challenges with domain/sector-specific characteristics. The AUTOSAR architecture provides a first step towards alleviating some of the organizational and technical challenges associated with developing/sustaining complex vehicular software, but as many of the authors noted, the specification still remains fluid.

From an MdH standpoint, it becomes important to

- Clearly articulate how the PROGRESS Component model can be applied in the vehicular sector
- Develop novel specification approaches that address the separations of concerns issues, without causing significant loss of meaning
- Gain a deeper understanding of the processes that are currently used in the vehicular sector, and collaborating with key stakeholders from those organizations to define and solve future problems
- Develop greater competence in the software lifecycle phases such that the software development can be morphed/adapted/tailored to create newer products/services.

Annotated Bibliograph

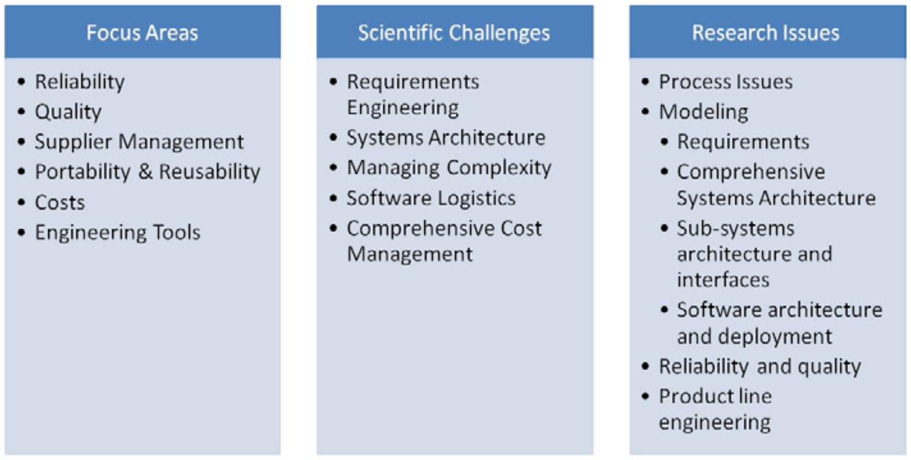
1. Grimm, K. *Software technology in an automotive company - major challenges*. in *Software Engineering, 2003. Proceedings. 25th International Conference on*. 2003.

Grimm notes that competitive challenges are driving the use of software-based innovations, which when coupled with the increased systems complexity, high quality demands, and rising time and cost pressures, makes software-related challenges significant for the automotive industry. Citing experts in DaimlerChrysler, he notes that 80% of future automotive innovations will be driven by electronics, and 90% of those innovations will be driven by software. As a result across the automotive supply chain, key competencies must be built up. Among the key competencies that every automotive organization needs are:

- Requirements engineering – internal DaimlerChrysler experience has shown that 40% of errors during use are attributable to a requirements error caused by immature specifications
- Software architecture at the vehicular level
- Improving productivity –through process improvement, tools, reuse (though extant tool chains do not cover the complete software development lifecycle)
- Domain specific knowledge such as meeting hard real-time requirements, control engineering etc.
- Software quality management (including Supplier management and cooperation)
- System testing and integration

2. Broy, M. *Automotive software and systems engineering*. in *Formal Methods and Models for Co-Design, 2005. MEMOCODE '05. Proceedings. Third ACM and IEEE International Conference on*. 2005.

In this paper, Broy frames the problem of systems and software engineering for embedded systems as one of the *great scientific, methodological and technical challenges* of today. The argument used to substantiate that claim is built on the fact that most modern cars have up to 80 controllers, connected using communication protocols by up to 5 different bus systems, all of which needs to be programmed. In addition to the sheer technical complexity of building these systems, there are two other factors that need to be taken into consideration: the cost of implementing these hardware/software solutions, which will account for 30% of the total vehicular cost in the near future; and that 90% of innovation in vehicles are enabled through these systems. The use of software-intensive solutions is driven by the ability to develop cheaper substitutes for current capabilities, and to create unique capabilities that further enhance the saleability of the vehicle.



Focus Areas

- Reliability
- Quality
- Supplier Management
- Portability & Reusability
- Costs
- Engineering Tools

Scientific Challenges

- Requirements Engineering
- Systems Architecture
- Managing Complexity
- Software Logistics
- Comprehensive Cost Management

Research Issues

- Process Issues
- Modeling
  - Requirements
  - Comprehensive Systems Architecture
  - Sub-systems architecture and interfaces
  - Software architecture and deployment
- Reliability and quality
- Product line engineering

Figure 1. Decomposing Automotive Software Engineering Challenges

Broy presents a two-level framework for the development of software systems: the foundational layer that blends traditional systems & software engineering, and the application layer. The foundational layer consists of the development process, requirement engineering, software architecture, complexity management, software logistics, and cost management; and the application layer consists of advanced driver assistance, drive-by-wire, advanced telematic services, man machine interfaces, and IT systems services. He further decomposes the domain challenges into the six fields of focus, the five scientific challenges, and four research issues, as shown in Figure 1.

3. Broy, M., *Challenges in automotive software engineering*, in *Proceedings of the 28th international conference on Software engineering*. 2006, ACM: Shanghai, China.
4. Broy, M., et al., *Engineering Automotive Software.* Proceedings of the IEEE, 2007. **95**(2): p. 356-373.

This paper is a synthesis of earlier work by the four authors in the area of automotive software engineering. Broy et al. make the case for the importance of field by noting that while a current premium car has about a 100MB of binary code (270 user interaction functions distributed across 70 embedded platforms), the next generation vehicles (circa 2012) will contain about 1 GB of software. More importantly, they highlight that even in the current state over 80% of the innovations come from computer systems, making it a major contributor to the value of contemporary cars. In looking at future projects, they note that 38% of the total value creation in automotive electrics/electronics by 2010 will be obtained through software (the worldwide value creation is expected to grow from 127 billion Euros in 2002 to an expected 315 billion Euros in 2015).

Broy et al. partition the problem space into the two domains of Organization & Economics, and Technology. Within the Organization & Economics domain they identify the major challenges to be:

i. Organization of the Development Process: The automotive value chain is hierarchical, with the OEMs creating an estimated 25% of the total value of a vehicle through the engine, integration and brand marketing. The design modularity enables low unit-cost (leveraging suppliers), but results in geographically distributed development that spans national and organizational boundaries. The use of software has resulted in the car morphing from an assembled product to an integrated system. This results in issues such as unintentional feature interaction, lack of an architectural view on embedded software systems, subsystem integration challenges. The OEMs use a black-box approach to subsystem specifications as a means of localizing errors, but lead to issues such as quality & precision of requirements documents, and limited transfer of IP among OEMs and their suppliers.

ii. Lifecycle & Innovation Cycles: A car is produced for 7-8 years, with an expected life of atleast 15 years after purchase. Given that CPU hardware has an expected life of five years, this creates a mismatch at the lifecycle level (not accounting for the numerous software updates that have to take place along the way). Further compounding the problem is the short innovation cycle in the automotive domain. Since software is optimized for a given hardware platform (driven by the desire to minimize unit cost), the lifecycle impacts are further complicated.

iii. Cost Model: Competition in the automotive market is driven by product price (leading to permanent optimization), product quality, product image, and differentiating product features (drives innovation & brand image). The minimization of unit cost drives engineers to focus on minimizing required memory and computation power. This optimized code is hard to evolve, hard to debug, results in increased coupling between modules, and in some cases, errors in software are a result of optimization. Overall maintenance and reuse become extremely difficult.

iv. Variants: They illustrate the issue of customization by using the work of Butts et al. on identifying 3488 potential component realizations by instantiating various algorithms and their variants. This becomes more complicated as variants are introduced by the OEM over the production lifecycle, and maintenance operations during regular operations leads to a mix of updated and legacy systems. Ideally technically desirable configurations have to be identified, and their correctness established - leading to the need for an elaborate design and test methodology.

v. Reuse: The reuse argument is built on the idea that while the features at the system level evolve across product generations by less than 10%, software across these generations is significantly rewritten. The current state process for managing reuse is currently not managed systematically between the OEMs and their suppliers.

There needs to be a suitable development process that *reduces complexity, enables innovation, saves cost, is transparent and address outsourcing*. These five issues impact the software lifecycle processes as shown in Table 1.

From a technology perspective, the issues that emerge focus on the multi-disciplinary nature of automotive software; the increased requirements on reliability, safety and security; and complexity of the hardware and technical infrastructure needed.

In looking at the key drivers in the automotive industry today and expected features in the cars of the future, Broy et al. identify nine areas that yield potential solutions: integrated comprehensive data models; architecture descriptions that bridge multiple levels of abstraction; model-based development across the lifecycle; domain-specific cost models; process orientation; model-based middleware; more effective reuse; tool support and integration across the lifecycle; and improved quality and reliability.

Table 1. Impact of Organization & Economic Issues on the Systems Development Lifecycle

Lifecycle Phase	**Impact**
Requirements Engineering	Use of tailored requirements processes; Assumption that software written in ECU's is always safety critical; managing variants; Need a comprehensive automotive domain model & corresponding deployment infrastructure
Platform/Software/Hardware Architecture Design	Platform software reuse is becoming common (as illustrated by AUTOSAR), but application level reuse is still not common; furthermore software as a service requires comprehensive software & systems engineering
Coding	OEMs rarely develop code; large amount of code is still handwritten as auto-generated code is not optimized for ECUs
Software/Systems Integration	Virtual integration and architecture verification currently not feasible due to a lack of systems specifications; supplier components do not always work together; missing guidelines for architecture development
Quality Assurance	Short term unit-sale cost focus negatively impacting lifecycle costs
Maintenance	Compatibility of software versions; defect diagnosis and repair; evolving hardware components.

5. Axelsson, J., et al., *Correlating Bussines Needs and Network Architectures in Automotive Applications-a Comparative Case Study.* Proceedings of FET'03: p. 219–228.

Using a comparative case study of network architectures used in passenger cars, heavy trucks, and construction equipment, Axelsson et al. identify the commonalities, differences, and the different demands placed on the communication networks. They note that the use of communication networks in automotive vehicles is driven by the need to replace numerous cables and harnesses, and thereby minimize the number of connection points, costs and weight. This use of multiplexing further enables the creation of new capabilities in the system. By comparing the three cases across the twelve dimensions of order of magnitude of production volumes, number of products, number of platforms, the number of physical configurations per product, amount of information, standards applied, use of network technologies, use of hardware optimization, use of open standards, safety criticality, use of advanced control, and presence of infotainment, they found that:

- The willingness to reduce fixed costs at the expense of variable costs increased with production volume. More specifically, hardware optimization to minimize the number of resources, since software optimization resulted in an increase in the variable costs with no reflected savings on the fixed costs.
- The product volume impacts the level of reuse/ commonality
- Tailoring of communications networks was driven by the high information content, and the level of optimization undertaken
- The use of networks opened up possibilities in the downstream value chain, namely in maintenance and services.

6. Fröberg, J., K. Sandström, and C. Norström, *Business situation reflected in automotive electronic architectures: analysis of four commercial cases*, in *Proceedings of the second international workshop on Software engineering for automotive systems*. 2005, ACM: St. Louis, Missouri.

Fröberg, Sandström, and Norström compare the business context, business requirements, and electronics architecture for four firms that operate in the construction equipment, trucks, buses and cars segments of the vehicular sector. In analyzing the architectures across physical configurations per product, network information, standards (network application level), network technologies, and internally developed nodes, they found:

- Requirements for openness and customer adaptation increase the number of physical configurations per product
- Interface requirements govern the choice of network application level standardization
- Network information requirements govern choice of technologies

- Size of the market influences the make-buy decision at the component level

In analyzing the architectural solution developed by the organizations, they found that key decision variables for architecture selection are product volume, market size, and requirements for openness & customer adaptation. They further identify four areas in which the OEMs face challenges, namely, Integration; Cost, safety, and functionality; Aftermarket; and Variants, brand & commonality. Potential solutions they identify include model-based development, software architectures, network technologies, and By-wire solutions.

7. Nolte, T., H. Hansson, and L.L. Bello. *Automotive communications-past, current and future*. in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*. 2005.

Nolte, Hansson and Bello provide a state-of-practice overview of automotive communications networks by grouping them into current wired; multimedia; upcoming wired; and wireless technologies. They define the communication requirements to be broadly classifiable along the dimensions of fault tolerance, determinism, bandwidth, flexibility, and security; and assess the eight classes of subsystems against those dimensions. After providing the reader with an overview of the technologies (including max bitrate and assessment of cost) that are currently available for the four groups, they assess each of the discussed technologies against Usage (the eight subsystem groupings of chassis, air-bags, powertrain, body& comfort, X-by-wire, multimedia/infotainment, wireless/telematics, diagnostics); and Requirements.

8. Sangiovanni-Vincentelli, A. and M. Di Natale, *Embedded System Design for Automotive Applications.* Computer, 2007. **40**(10): p. 42-51.

Sangiovanni-Vincentelli and Di Natale characterize the automotive supply chain along four tiers, starting with the OEMs, followed by 1st Tier suppliers that provide subsystems, 2nd Tier suppliers who provide components to both the OEMs and the 1st Tier suppliers, and manufacturing suppliers. They note that while the supply chain is relatively stable, the increased use of electronics is stressing the current roles that the various actors have in the supply chain. Given that the supply chain was designed to operate with simple black-box integrated systems with requirements capture and OEM issued specifications (message interface periods and general performance requirements, but without a detailed definition neither of timing & synchronization properties nor of communication protocol requirements), integration could be carried out in a heuristic, ad hoc way. The increased complexity of embedded software gives rise to requirements for system-level analysis and modeling not only for predictability and composability at design-time partitioning of end-to-end functions, but also for providing guidance to the designer at the early stages (in which electronics and software architectures are evaluated and selected for product lines). They highlight three technical challenges that need to be overcome in the presence of real-time & reliability requirements: time predictability; Dependability; Composability & Extensibility versus Efficiency.

They note that most model-based design tools suffer from a lack of:

- Separation between functional and architectural models
- Support for defining the task and resource model
- Modeling support for the analysis and back annotation of scheduling related delays
- Sufficient semantics preservation

They dive deeper into four specific issues that need to be addressed over the long term:

i. Timing predictability and isolation: They note that larger, more complex applications that are deployed with significant parallelism on each ECU, consisting of a densely connected graph of distributed computations and new safety critical functions, require tight deadlines and the guaranteed absence of timing faults (invalidating both the assumptions of slack built in by designers, and graceful degradation of tasks). Therefore a new rigorous science must be established.

ii. Communication and Distributed Systems : more specifically they highlight the capabilities of the FlexRay standard in meeting the requirements for highly deterministic, high speed communications. They also highlight a glaring gap in current RTOS standards with respect to synchronizing the communication and RTOS layers.

iii. Composability & AUTOSAR: They note that component-based design can provide encapsulation and a separation of concerns (and thereby improving reuse) if information hiding is implemented in a manner that allows composability (guarantee preservation of component properties across integration) and compositionality (deduction of composed object's global properties from its component properties – ensures correctness-by-construction). The AUTOSAR metamodel lacks a timing model, and does not provide clear & unambiguous communication & synchronization semantics.

iv. Platform-based architecture selection: Current approaches for architecture selection are built around a set of designer-driven what-if analysis approaches that address specific aspects of systems behavior across multiple-levels of abstraction such as end-to-end latency and schedulability at the task level to product-line cost analysis. They highlight the importance of being able to automatically configure the software architecture through a function to task allocation, task to ECU mapping, and signal to message mapping, provided the designer has the functional and physical architectures specified.

9. Reinfrank, M., *Why is automotive software so valuable?: or 5000 lines of code for a cup of gasoline less*, in *Proceedings of the 2006 international workshop on Software engineering for automotive systems*. 2006, ACM: Shanghai, China.

In his keynote talk at SEAS 2006, Reinfrank notes that 'we are facing a 25% HW vs 75% SW cost ratio for a modern engine control unit', and that non-conformance costs due to software related calls are significantly increasing over the past decades. The capability to design software solutions in a high quality, high performance, and timely to market way has become one of the decisive competitive factors in the automotive supplier industry. Discussing the root causes for the lack of software capability, he notes that two decades ago SW R&D for an engine control system was almost negligible in comparison to the hardware related costs, as a result software R&D was buried by suppliers under the general overhead category, and therefore free of charge to the OEM. Since the predominant approach to pricing software solution is based on effort, and not on the value of the total solution, suppliers have no incentive to dramatically improve their productivity. To escape the bottleneck, he proposes two solutions:

- Addressing the total ownership costs of the vehicle, as opposed to making localized software cost decisions

- Paying for the value of the software, not just the man years put into developing a solution

10. Hwong, B.M. and S. Xiping. *Tailoring the Process for Automotive Software Requirements Engineering*. in *Automotive Requirements Engineering Workshop, 2006. AuRE '06. International*. 2006.

Hwong and Song report on the lessons learned in trying to implement a requirements management process in a software organization that transitioned from being part of the OEM to becoming a supplier to the OEM. They highlight three key lessons learned: tailoring the new corporate process to address the needs of a small group; creating formal artifacts to better define the boundary between the customer and the supplier; learning to work in a distributed fashion. Overall, they recommend the use of a product-line approach as it would enable the organization to reuse software documents and code more effectively.

11. Möller, A., et al. *Industrial grading of quality requirements for automotive software component technologies*. 2005.

Möller et al note that while component based development approaches have been successful in the PC application domain, the requirements for their use is the automotive domain is not well understood. Towards that end, they develop a graded set of industrial requirements for component technologies, that can be used to guide modifications/extensions to make component technology more applicable in the automotive context. They extend earlier work on defining component requirements from the heavy vehicles sector using a Delphi approach wherein experts participated in a workshop in which they graded twelve previously developed requirements. The requirements were broadly classified into three groups:

- Technical requirements – analyzable, testable & debuggable, portable, resource-constrained, component modeling, and computational model
- Process requirements – introducible, reusable, maintainable, understandable
- Derived/Synthesized requirements – source code components, static configuration

The top four requirements were testable & debuggable, reusable, portable and maintainable.

12. Botaschanjan, J., et al., *Towards verified automotive software*, in *Proceedings of the second international workshop on Software engineering for automotive systems*. 2005, ACM: St. Louis, Missouri.

Botaschanjan et al. introduce a verification approach that includes a framework of verified modules to assist in the verification of the actual application, to guarantee the correctness of the developed systems, and at the same time reduce verification and integration costs. They note that testing is the dominant approach used in the automotive industry to ensure that independently developed subsystems (ECUs) are integrated to safely perform a required critical function. In order to deliver a correctness proof for the fulfillment of safety critical properties of the system:

- The affected subset of functionality needs to be identified and isolated
- Correctness of the demanded functionality as to be proven
- Guarantee that the rest of the system cannot interfere with the safety critical subset

Their verification framework uses a formalized verified operating system (OSEKtime), the FlexRay communication protocol, and formal models of hardware behavior, within which models of the application can be verified. Additionally, their framework supports automatic C code generation from the verified models. The workflow that they propose for systems development (shown in Figure 2.), starts with natural language requirement specifications, that are formalized using the AutoFocus tool (a model based case tool with time synchronous operational semantics). The verification is carried out using Isabelle theories that can be either auto generated from the model or derived manually from the requirements.

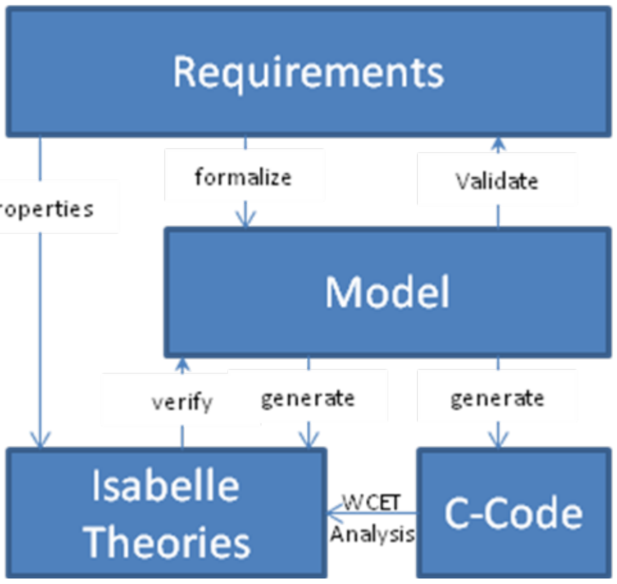


Figure 2. Development Workflow

They demonstrate the feasibility of the framework using an emergency call example to prove two properties: the emergency call is not initiated as long as there is no crash, and that the emergency call is performed within a specified period of time.

13. Weber, M. and J. Weisbrod, *Requirements engineering in automotive development: experiences and challenges.* Software, IEEE, 2003. **20**(1): p. 16-24.

Weber and Weisbrod present the motivation for, challenges in and solutions developed for requirements engineering of software-based automotive systems at DaimlerChrysler. They motivate the need for new requirements engineering approaches by highlighting the fact that systems development cycle times are shrinking for automotive electronics targeted at high-end cars, while at the same time costs are increasing. Since components are developed and tested over a sequence of prototyping phases, they are also developed in several different variants in different schedules, as a result, specification activities have reached a level of complexity that cannot be managed by a single person using textual specifications alone. Using experiences in the domains of entertainment & telematics, car interior & passenger comfort, and driver assistance & safety critical systems , they discuss lessons learned from the general, process and technical perspective. The overarching approach they used for deploying their requirements engineering approach is shown in Figure 3.

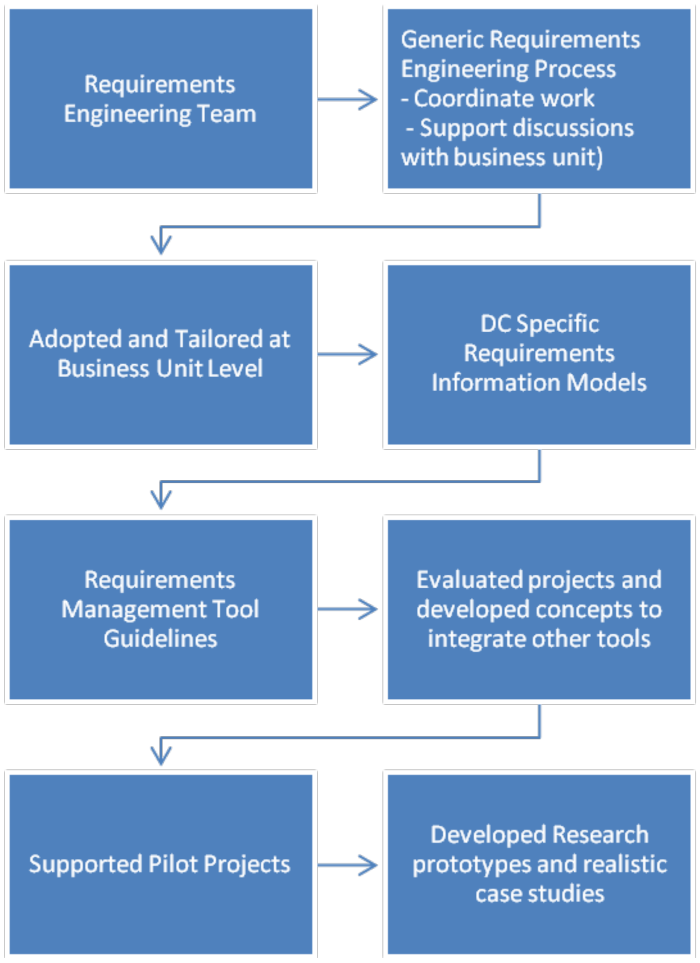

Requirements Engineering Team
Generic Requirements Engineering Process - Coordinate work - Support discussions with business unit)
Adopted and Tailored at Business Unit Level
DC Specific Requirements Information Models
Requirements Management Tool Guidelines
Evaluated projects and developed concepts to integrate other tools
Supported Pilot Projects
Developed Research prototypes and realistic case studies

Figure 3. Developing Requirements Engineering Process at DC

General issues

- Textual requirements alone insufficient for automotive development: need to have additional information such as attributes (for example working state, due date etc.), requirements history (both local and with respect to some overall baseline), illustrative pictures (design rationale, test information, interface details etc), and dependency information. This led to the creation of a modular company-wide Requirements Management Information model (RMI).

- Need to maintain the look-and-feel of a document oriented world: Since system developers focus on document-centered requirements engineering, the tool needs to provide basic text processing while adding the management and traceability functionality over a database management structure. In addition to specification generation, the tool must also support management reporting.

- Lack of clear boundary between the requirements specification and supplier system specification

- Redundancy in system specification: in large projects, when hundreds of specifications exist, redundant information leads to inconsistencies and wasteful work. In the case of legacy systems, large document structures already exist, which often mirror organizational and/or the automotive business structure, making harmonization difficult. The end-goal would be to create a database of specification objects distributed across manufacturers and suppliers from which a single document can be generated.
- Managing presentation of content and comply with the RMI
- Promoting reuse of existing specifications

Requirements Engineering Process

- Detailed specifications documented only at the leaves of the system decomposition tree: systematic processing of high-level requirements and design decisions is insufficient to manage the complexity of the system, neither is bottom up aggregation from detailed specifications (as rationale is lost)
- Engineers fail to manage non-functional requirements, acceptance criteria, and test cases
- Change is constant: The major underlying reason for the frequency of change is the growing complexity, parallelization, and time restrictions on the development process, which lead developers to introduce assumptions into the system. There are two levels of granularity when considering requirements management- 'official' change management between the customer and supplier, and requirement change management at the technical level (refinements and clarifications) .
- User-adaptable views are a powerful instrument for guiding and managing specification change
- Well organized reviews are essential for effective supplier management

Technical Issues

- Have to distribute the requirements engineering process
- Deciding what traces need to be maintained in the system
- Coupling between the document-based tool and model-based tool not used effectively: the challenges include speed, integrated document generation, user-interface, and automation.
- RM tools are the primary instrument to leverage requirements engineering practices

14. Caulfield, I. and N. Power, *Domain focused requirements engineering: A case study of successful requirements engineering in a market-oriented automotive software development company.*

Caulfield and Power focus on requirements engineering practices in a market-driven automotive aftermarket scenario using a longitudinal case study that covers all the steps in the requirements process – requirements elicitation & analysis through negotiation, documentation, validation, verification & inspection of requirements artifacts. They identify which requirements engineering practices are effective in a given situation, and identify the factors that enable them to be successful. They present the evolution of requirements management at a company called Independent Tools that initially had a hierarchical, sequential development model. The implementation of the model in the organization was perceived to have serious issues ranging from taking too long, to not customer focused enough, to project managers acting as 'program police', to not working as a team (consensus difficult to obtain, not meeting face to face, apportioning blame, and not meeting ace to face). The new product realization cycle was built around a customer focus (using software QFD practices) using cross-functional teams using face to face meetings, with a set target date.

15. Tindell, K., et al. *Safe automotive software development*. in *Design, Automation and Test in Europe Conference and Exhibition, 2003*. 2003.

This paper is an aggregation of three expert opinions on the challenges of developing safe automotive software. Ernst frames the problem by noting that traditional simulation-based verification approaches are not sufficient to address the complex distributed nature of automotive software and the additional challenges imposed by the integration of such software, leading to research into formal approaches that could become a model for safe automotive software development.

Tindell notes that as ECU software becomes more complex, the need to run more than one subsystem from within an ECU, such as the mix of low and high integrity software in a X-by-wire system, increases. To enable such capability, an operating system is needed, that can provide protection between these components. The two critical requirements laid out by the HIS group (Audi, BMW, DaimlerChrysler, Porsche, and VW) include: protection between applications and tasks (in both functional and timing domains); and OS designed to tolerate malicious software. In addition, component costs must be as low as possible (therefore choice of hardware is limited). Tindell presents the characteristics of the APOS prototype that supports both execution time monitoring and resource usage time.

Koeptz notes that safety is an emergent property of systems, not that of a component. A safety case is an accumulation of evidence about the quality of the components and their interaction patterns in order to convince an expert that the probability of an accident occurring is below the acceptable level. To make the reliability assessment tractable, the architecture must ensure the independent failure of components, which therefore requires fault containment (limiting the impact of a fault to a well-defined region of the system) and error containment (ensuring that the consequences of faults, namely errors, cannot propagate to other components and mutilate their internal state) at the architecture level. In a safety critical system, he notes, an error containment region requires at least two fault containment regions. Furthermore, any safety-critical system design must start with a precise specification of a fault hypothesis (partition the system into fault-containment regions, state the assumed failure modes & associated probabilities, and establish the error propagation boundaries). This concept has been effectively implemented and demonstrated using the time triggered architecture (TTA).

Wolf uses a three-layer model to discuss the various classes of automotive software: lowest layer of system functions (RTOS and Basic I/O), basic software (functions that are specific to the controller, such as fuel injection for engine control), and vehicle functions (for example, adaptive cruise control). These vehicle function provide a means to the OEM for product differentiation, hence they are willing to invest in these function to create added value. From an OEM perspective, it is preferable to have a software integration flow wherein vehicle functionality is not exposed to the supplier, and allows for rapid design space exploration.

16. Jersak, M., et al. *Formal methods for integration of automotive software*. in *Design, Automation and Test in Europe Conference and Exhibition, 2003*. 2003.

Jerask et al. note that there is no well defined software integration process for engine control units that has been defined (circa 2003) that meets all the key requirements of automotive manufacturers, and define an approach to meet that gap. The methodology that they propose, builds on the notion that in principle, techniques and required information are available for formal timing analysis of automotive software at all levels. The process they define, and implicitly, the information flow between the automotive software manufacturer, different ECU suppliers, RTOS supplier and system integrator, enables a certifiable software integration flow that:

a. Allows the exchange of key information between the individual partners

b. Protects the IP for each stakeholder

The proposed integration & certification flow consists of a well defined OSEK configuration, adherence to software interfaces, performance analysis & characterizations of all entities involved, and a performance analysis of the entire system. Partners exchange properly specified blackbox components as specified in the agreements (including conventions for RTOS configuration, communication & memory budgeting). The open issue with respect to adopting the methodology is the issue of performance validation – in particular, the validation of timing.

17. Vitkin, L., et al., *Effort Estimation in Model-Based Software Development. SAE Paper 2006-01-0309.* Society of Automotive Engineers, 2006.

Vitkin et al. note that while the primary goal of using model based development (workflow shown in Figure 4.) is the reduction of project development time through the use of executable modeling and autocoding, efficient production quality code cannot be generated through a push of a button. While there are still time savings obtained through the process of manual settings of the software properties of model blocks, and the verification of the generated code, more precise estimates are needed for the engineering effort needed to perform autocoding, and autocode generation. Building on experiences with model-based development at Delphi Corporation, they develop an estimation model that enables developers to more accurately estimate the efforts for autocoding and autocode verification.

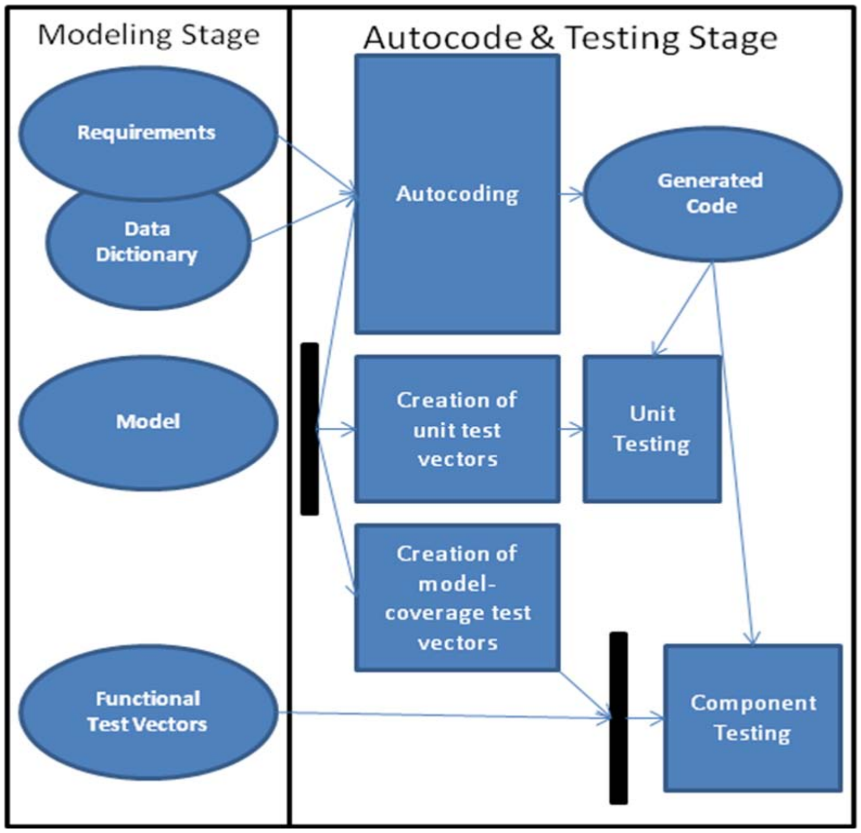


Figure 4. Model-Based Development Workflow

They develop an estimate of the autocoding complexity as a function of the module software number (MSON) and the module connectivity number (MCN), as shown in Table 2.

$$MSON = \sum_{k=1}^{n} BSON_k$$

$$MSONco = \sum_{k=1}^{N_{co}} BSONco_k$$

$$MSONnb = \sum_{k=1}^{N_{nb}} BSONnb_k$$

$$MSONdelta = C * MSONco + MSONnb$$
$$C = 1 - (MSONco / MSONorg)$$

$$MCN = \sum Oi$$
$$MCN = MCNbool + MCNnbool$$
$$MC = k1 * MSONdelta + k2 * MCNnbool\_delta$$

Table 2. Effort Estimation for Autocoding

The factors k1 and k2 are estimated for both fixed point code (0.8-1.0, and 0.0-0.2) and floating point code (0.5-0.8, and 0.0). Similarly, they develop an estimate for the testing effort needed at both the unit testing and integration testing levels.

18. Sturmer, I., et al., *Experiences with model and autocode reviews in model-based software development*, in *Proceedings of the 2006 international workshop on Software engineering for automotive systems*. 2006, ACM: Shanghai, China.

Stumer et al. highlight the importance of model reviews and autocode reviews in the model based development process, and present an approach for combined model and autocode generated reviews. Model reviews are carried out to:

- o check whether the functional requirements are realized in the model
- o ensure that relevant guidelines for model development are followed
- o check that selected quality criteria are met
- o check that the implementation model meets the requirements for generating robust and efficient code

Similarly autocode reviews are carried out to:

- o find errors that have been introduced due to inappropriate modeling or faulty use of the code generator
- o identify errors in the model or in the autocode that are easier to detect in code than in the model itself
- o ensure that custom code is correctly integrated
- o identify efficiency improvements
- o reveal code generation errors

The overall quality assurance approach that they propose for model-based development is the execution of the following steps in sequence: model reviews, implementation model tests, static analysis of the autocode, autocode reviews, implementation model regression tests, back-to-back tests of the implementation model code versus the fixed-point autocode. The propose a format for the reviews in which the findings are divided into five classes: questions, editorial remarks, uncritical remarks, important remarks, and critical remarks. All of the findings to be recorded using DOORS and referenced to the affected model or autocode element. When doing model reviews, the reviewers were provided with a 'top 10' checklist that focused on ensuring safe and efficient code generation (as opposed to general modeling concerns), and each module was reviewed using the '4-eye principle'. The autocode review followed a two phase approach, with phase 1 focusing on aspects such as comprehensibility, structure and form of the code; while the second phase consisted of specific error causes. Additionally each code line had to be mapped back to a specific model pattern to ensure traceability, and to determine if the code generator missed any model parts.

Using their approach, they identified 236 finding in the model review, resulting in 146 revisions to the implementation model. As they noted, the checklist helped with short reviews, but was not effective in –depth reviews. The autocode review resulted in 189 findings, which in turn led to 20 revisions of the implementation model. The two stage model proved to be successful and the reviewers were able to cover between 84 to 257 executable lines of C code per hour.

AUTOSAR & MISRA Related Papers [SAE - 42 for AUTOSAR, 29 for MISRA]

Context/Overview:

19. H. Heinecke, K.-P. Schnelle, H. Fennel, J. Bortolazzi, L. Lundh, J. Leflour, J.-L. Mat´e, K. Nishikawa, and T. Scharnhorst. *AUTomotive Open System ARchitecture — an industry-wide initiative to manage the complexity of emerging automotive E/E-architectures*. In Proceed. Convergence 2004, International Congress on Transportation Electronics, Detroit, October 2004.
20. H. Fennel, S. Bunzel, H.Heinecke, J. Bielefeld, S. Fürstand K.P.Schnelle, W. Grote, N. Maldenerand T. Weber, F. Wohlgemuth, J. Ruh, L. Lundh, T. Sandén, P. Heitkämper, R. Rimkus, J. Leflour, A. Gilberg, U. Virnich, S. Voget, K. Nishikawa, K. Kajio, K. Lange, T. Scharnhorst, and B. Kunkel. *Achievements and exploitation of the AUTOSAR development partnership*. In *Convergence 2006*, Detroit, USA, October
21. Rolina, T., 'Past, Present, and Future of Real-Time Embedded Automotive Software: A Close look at basic concepts of AUTOSAR', SAE international, 2005,
22. Kinkelin, G., A. Gilberg, B. Delord, H. Heinecke, S. Fürst, J. Moessinger, A. Lapp, U. Virnich, S. Bunzel, and T. Weber. 2008. AUTOSAR on the Road.

Heinecke et al. (2004) motivate the creation of the AUTOSAR standard using the heterogeneity in the automotive electric/electronic architecture landscape as an illustrative example. They note that this increasing complexity in architectures can only be solved by an industry tconsortium as a whole.

The AUTOSAR serves as the basic infrastructure for management of function with both future applications and current software. In looking at the drivers of standardization, they note:

- Management of E/E complexity is associated with growth in functional scope
- Flexibility for product modifications/upgrades/updates
- Scalability of solutions within and across product lines
- Improved quality and reliability of E/S systems

AUTOSAR was founded in 2003 to serve as a platform for the implementation of future vehicle applications, and to minimize current barriers between functional domains (vehicle centric versus passenger centric). The key benefits that they list for the dominant stakeholder groups (OEM, Supplier, Tool Vendors, and New Entrants) make it a win-win to support development and participation.

Fennel et al. (2006) document the lessons learned from Release 2.0 validation efforts with core partners and premium members. They note that while eight new concepts were added to the release, the major achievement was in the industry-wide consolidation of 'existing' basic software designs

Rolina (2009) notes that standardization comes at a cost, and his benchmarking effort showed that AUTOSAR-based applications were more demanding of RAM/ROM. The associated impact is visible on overall run-time performance. As with OSEK, the use of a static AUTOSAR configuration supports the design space exploration and optimization at runtime. Most importantly, the strategy that he suggests for transitioning to using AUTOSAR, especially in the legacy systems context, is to work incrementally.

Experiences:

23. Kajio, K. 2007. TOYOTA Electronic Architecture and AUTOSAR Pilot.

Nishikawa and Kaijo present a case study of Toyota's usage of AUTOSAR through a discussion of the Toyota electronic architecture, and lessons learned through a pilot case study. Highlighting Toyota's evolution from independent subsystem control in the 1990s to system integration in the 2000s, they note that the Lexus currently has over 70 ECUs executing more than 7 million computation steps through in-vehicle code. Toyota's unified process for developing electronic systems leverages multiple techniques such as model-based-design, auto-code generation, hardware-in-the-loop & software-in-the-loop simulations, and a standardized software platform called basic software. Discussing the lessons learned from their pilot study on the AVENSIS Outer Mirror Pilot Implementation(wherein they found "hundreds of bugs and drawbacks" from different implementations on the same hardware platform, they note:

- AUTOSAR communications stack supports the major communications protocols (unlike OSEK COM)

- If functional interfaces are standardized to a 'certain' level of granularity, application portability and reusability would be much easier

- Overhead on run-time and RAM is currently too high for the ECU, especially in the case where it does not need the services of the OS/RTE

- Software quality and migration risks are high since the specification for AUTOSAR is still under development.

24. Schoof, J., and D. Wybo. 2007. 2006-01-0168 No Detour Needed: Getting to AUTOSAR Via OSEK. *SAE TRANSACTIONS* 115, no. 7: 50.
    Schoof and Wybo provide a short overview of software standardization efforts in the automotive industry, using the evolution of OSEK compliance as the underlying theme. They take the readers from the development of CAN drivers to the creation of the OSEK standards covering the operating system, communication system, and network management system; leading to the creation of the HIS I/O library. In addressing the concerns about transitioning to AUTOSAR, they note that almost all the OSEK specifications have actually made it into the AUTOSAR specification. From the perspective of providing guidance, they note that the transition using AUTOSAR could be effectively accomplished by transitioning first to OSEK, anda then to AUTOSAR. They make a crucial point that the AUTOSAR philosophy *will result in a significant change in the workflow behind vehicular software development.*

25. Freund, Ulrich, Vivek Jaikamal, and Joaichm Löchner. 2009. Multi-Level System Integration of Automotive ECUs based on AUTOSAR.
    Freund, Jaikamal, and Löchner show how model-based control-algorithm validation can be interleaved with the AUTOSAR configuration approach. In discussing the control algorithm development process, they note that the process is a threefold v cycle, starting with simulation, moving onto rapid prototyping, and concluding with production code generation. They note that currently more than 60 million vehicles run ASCET generated code (wherein the generator is certifiable according to IEC 61508). They show how AUTOSAR can be integrated into the threefold V process at the code generation stage (in the case of atomic software components), and across the initial stages of simulation and rapid-protoyping for validation purposes. Furthermore, they note that this approach was used in a cruise-control model integration project. Since both AUTOSAR is based on the same ideas of meta-model based code generation, the authors point out that interweaving the approaches is easily carried out.

Safety:

26. Johannessen, P. 2006. AUTOSAR Safety Approach.

Johannessen points out that compliance to the AUTOSAR alone is not sufficient to ensure safety for functions or systems. Of the nine project objectives of AUTOSAR (further refined into 39 main requirements), the first objective is the '*consideration of availability and safety requirements*'- which as he points out, ensures a general safety focus across the entire effort. Looking deeper at the specific requirements, the four that he points out are: Main 10, Main 30 from a safety requirements perspective, and Main 20 and Main 50 from a redundancy activation perspective.

Table 3: AUTOSAR Requirements Related to Safety

Main 10 = Using AUTOSAR, the system reliability shall be achievable with a failure rate down to 10-8 per hour
Main 30 = A SIL 3 compatible development process must be possible with AUTOSAR
Main 20 = AUTOSAR shall provide mechanisms that support redundancy paths
Main 50 = AUTOSAR shall support inter- and intra-ECU communications mechanisms with high reliability

The paper concludes with a discussion of the deliverables of the safety team: Technical Safety Concept, The Project Safety Guidelines, the HW Fault Model, the Safety Related Extensions.

27. Ward, D. D. 2008. MISRA Activities for Safety-related Software Development.

28. Johansson, Rolf, and Thomas Heurung. 2009. ISO-26262 Implications on Timing of Automotive E/E System Design Processes. In . Detroit, Michigan, USA: SAE International, April 20.

Johansson and Heurung note that new automotive electronic systems pertaining to safety are increasingly distributed, and that these complex solutions require the support of specific development processes and tools. They present an overview of the ISO-26262 standard, and illustrate how the safety requirements of the standard with respect to timing can be met and enabled through the use of AUTOSAR. They build on the idea of a contract-based approach that uses on standardized timing models (those of AUTOSAR or the TIMMO initiative) to develop a safety case for a distributed automotive embedded system using an example of an adaptive cruise control system.

Tools:

29. Mjeda, A., G. Leen, and E. Walsh. 2007. 2007-01-0509 The AUTOSAR Standard-The Experience of Applying Simulink According to its Requirements. *SAE SP* 2126: 93

Mjeda, Leen and Walsh cite a Frost & Sullivan report that projects that software is expected to account for 11% of the cost of vehicles by 2011 to reinforce the notion that software is the driving force behind innovation in the automotive industry. By mapping the AUTOSAR standard to the Simulink counterparts, they identify the changes that need to be made to make the model AUTOSAR compliant, and illustrate their findings using a case study of an experimental engine management system.

30. Niggemann, O., U. Eisemann, M. Beine, and U. Kiffmeier. 2007. Behavior Modeling Tools in an Architecture-Driven Development Process-From Function Models to AUTOSAR.

Niggerman et al. note that the benefits of application software module reuse include the ability to utilize software modules that are already tested, thereby minimizing potential software hazards; reduce development workload; and transfer the product line approach more easily into software development. In presenting the problem of separations of concern across the various levels of abstraction, they highlight how the AUTOSAR approach can be leveraged using a tool called TargetLink, and identify the necessary adaptations that need to be made for the workflows.

31. Stichling, D., O. Niggemann, R. Otterbach, and K. Hoffmeister. 2007. Effective Cooperation of System Level und ECU Centric Tools within the AUTOSAR Tool Chain.

Stichling et al. identify two main workflows that emerge from their analysis of software development processes at automotive manufacturers and suppliers, using the dSPACE system level design tool *SystemDesk* and the Elektrobit ECU centric tool *tresos ECU.* They motivate the need for the analysis by pointing out that the AUTOSAR methodology defines a software development workflow for ECUs that does not cover the real processes both between OEMs and their suppliers, and within the organization. The challenges that they found with the tool chain connecting system-level and ECU-centric design included:

- Compatibility of the interfaces
- Connecting application components to basic software components