

# Challenges in Automotive Software Engineering

Manfred Broy  
Institut für Informatik  
Technische Universität München  
D-80290 München, Germany  
broy@in.tum.de

## ABSTRACT

The amount of software in cars grows exponentially. Driving forces of this development are cheaper and more powerful hardware and the demand for innovations by new functions. The rapid increase of software and software based functionality brings various challenges (see [21], [23], [25], [26]) for the automotive industries, for their organization, key competencies, processes, methods, tools, models, product structures, division of work, logistics, maintenance, and long term strategies. From a software engineering perspective, the automotive industry is an ideal and fascinating application domain for advanced techniques. Although the automotive industry may adopt general results and solutions from the software engineering body of knowledge gained in other domains, the specific constraints and domain specific requirements in the automotive industry ask for individual solutions and bring various challenges for automotive software engineering. In cars we find literally all interesting problems and challenging issues of software and systems engineering.

## Categories and Subject Descriptors

D.2 [Software Engineering]: D.2.1 Requirements/Specifications (D.3.1), D.2.2 Design Tools and Techniques, D.2.10 Design (D.2.2), D.2.11 Software Architectures

## General Terms

Design, Economics, Reliability, Experimentation, Human Factors, Standardization

## Keywords

Automotive Software Engineering, Model Driven Development, Embedded Systems

## 1. INTRODUCTION

In many technical products, software plays a dominant role today. In cars, this applies even to the extreme. Today software in cars is a dominant factor for the car industry, bringing various problems but being nevertheless decisive for competition.

One can easily see that the amount of software in cars has been growing exponentially over the last 30 years, and one can expect this trend to continue for another 20 years at least.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*ICSE'06*, May 20-28, 2006, Shanghai, China.  
Copyright 2006 ACM 1-59593-085-X/06/0005...\$5.00.

The first software found its way into cars only at a time about thirty years ago – so software grew in only more or less four generations of cars. From one generation to the next, the software amount was growing by a factor of ten, or even more. Today we find in premium cars more than ten million lines of code and we expect to find in the next generation ten times more.

Many new innovative functions in cars are enabled and driven by software. Recent issues are energy management and the current step into hybrid solutions, which can only be realized in an economic way by plenty of software. It is mainly the application domain specific innovations with their stronger dependencies and feature interactions that ask for cross application domain organizations.

In the following, we shortly describe the history of software in cars as far as it is relevant to understand the current challenges. Then we sketch the state of practice with its problems, challenges, and opportunities. Based on a short estimation of the future development we describe our expectation how the field will develop. Finally we describe current research in the domain of automotive software engineering (see also [14]).

## 2. The History

Just 30 year ago, software was first deployed into cars to control the engine and, in particular, the ignition.

The first software-based solutions were very local, isolated and unrelated. The hardware/software systems were growing bottom up. This determined the basic architecture in cars with their dedicated controllers (Electronic Control Units or ECUs) for the different tasks as well as dedicated sensors and actuators. Over the time to optimize wiring, bus systems (see [29]) were deployed into the cars by which the ECUs became connected with the sensors, and actuators.

Given such an infrastructure, ECUs got connected, too, and could exchange information. As a result the car industry started to introduce functions that were realized distributed over several ECUs connected by the bus systems. Such functions were built bottom up. A systematic top down design was never used. If we would not go in evolutionary steps but re-design the hardware/software systems in cars from scratch today, we would certainly come up with a quite different solution.

## 3. State of Practice

Today premium cars feature not less than 70 ECUs connected by more than 5 different bus systems. Up to 40 % of the production costs of a car are due to electronics and software.

### 3.1 The Role of Software in Cars

Within only 30 years the amount of software in cars went from 0 to more than 10.000.000 lines of code. More than 2000 individual functions are realized or controlled by software in premium cars,

today. 50–70% of the development costs of the software/hardware systems are software costs. For a view on the network in a car see Figure 1.

Software as well as hardware became enabling technologies in cars. They enable new features and functionalities. Hardware is becoming more and more a commodity – as seen by the price decay for ECUs – while software determines the functionality and therefore becomes the dominant factor.

### 3.2 Embedded Software as Innovation Driver

Software is today the most crucial innovation driver for technical systems, in general. By software we realize innovative functions, we find new ways of implementing known functions with reduced costs, less weight or higher quality, we save energy and, what is, in particular, important, we combine functions and correlate them into multi-functional systems.

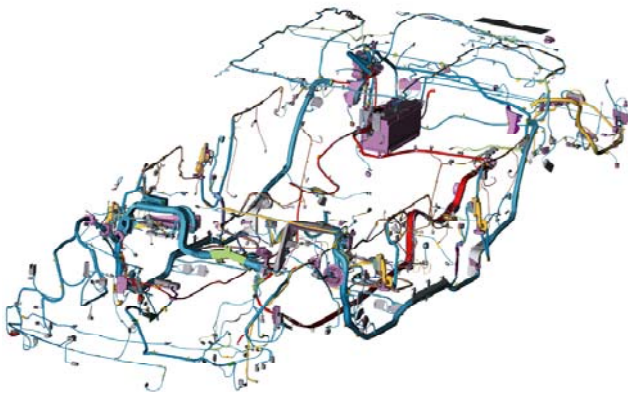


Figure 1. Onboard Network

This way software allows for completely new solutions of, in principle, known tasks. What has been said here for embedded systems, in general, applies to cars, in particular.

### 3.3 Deficits in Engineering Software in Cars

Today the engineering of software in cars is still in its infancy. The quick increase of software and the traditional structures of the car industry make it difficult for this old economy to adapt fast enough to the quite different requirements of software-intensive system, which cars become more and more.

Following its tradition to find its own proprietary solutions (see [7]) the car industry developed to a large extent its own approaches also in the software domain. It is amazing to see the amount of proprietary technology in the software in cars. This applies to operating system, communication protocols, tools, architecture, in fact, basically to all aspects of software in cars. Of course, automotive software engineering has its own domain specific requirements (see below). Nevertheless the car industry could have done much better by benefiting from existing experiences and technology from other domains, in particular, telecommunication and avionics.

Lifecycle management of software in cars is in its early stage. Many suppliers and even some OEMs are not even at CMM level 2. This is, in particular, a problem in a situation where the systems are developed by distributed concurrent engineering and the

software is highly complex, multi-functional, distributed, real time and safety critical.

Reuse of solutions (see [22]) from one car to the next is insufficient and only done in a consequent way in some limited areas. In many sub-domains the functionality from one car generation to the next is only changed and enhanced by 10 % while more than 90 % of the software is rewritten. The reason is a low level, hardware specific implementation, which makes it difficult to change, adopt, and port existing code.

Finally, the amount of automation in software production for software in cars is quite low. Tools are only used in an isolated manner. There is neither a properly defined design flow nor seamless tool chains for distributed functions.

## 4. The Domain Profile

Traditionally the car industry is highly vertically organized. In software engineering we would say it is modular. The mechanical engineers worked hard for over 100 years to make the various sub-systems in cars in their development and production quite independent. This facilitates independent development and production of the sub-parts and allows for an enormous division of labor.

As a result, suppliers could take over a considerable part of the engineering, the development, and also the production by a consequent outsourcing. Ideally, the parts of cars are produced by a chain of suppliers and more or less only assembled by the car manufacturer (called OEM in the following). Thus, a large amount of the engineering and production is outsourced and the cost and risk distribution can be optimized. A car is (or better was) considered as a kit of subparts that are assembled by the OEM.

With software becoming a major force of innovation the situation changed drastically:

- Traditionally quite unrelated and independent functions (such as braking, steering, or controlling the engine) that were freely controlled by the driver get related and start to interact. The car turns from an assembled device into an integrated system. Phenomena like unintentional feature interaction become issues.
- Assembling sub-parts becomes system integration.
- The behavior of cars becomes much more programmable. Certain properties, such as comfort or sportive handling are no longer solely determined the mechanics but also by the software.
- The costs of cars get more and more influenced by development costs of software, for which the traditional cost models dominated by the cost by part paradigm are no longer fully valid.

Size and structure of the embedded software/hardware systems in cars are enormous. The application software is built on top of real time operating systems and bus drivers. Most of the software is hard real time critical or at least soft real time critical.

The requirements for the software systems in cars are quite specific:

- wide range of different users (drivers and passengers, but also maintenance)

- specific maintenance situation
- safety critical functions
- specific context of operation of the systems
- heterogeneity of functions (from embedded real time control to infotainment, from comfort functions like air condition to driver assistance, from energy management to software download (flash) functionality, from air bags to on board diagnosis and error logging).

As a result the complexity and spectrum of requirements for on board software is enormous.

## 5. THE FUTURE

The increase of software and functionality in cars is not close to an end, in the contrary. We can expect a substantial growth in the future. The future development is driven by the following trends:

- high demand of new innovative or improved functionality
- quickly changing platforms and system infrastructures
- rapid increase in development cost in spite of a heavy cost pressure
- demand for higher quality and reliability
- shorter time-to-market
- increased individualization

As a result there is a high demand for dedicated research on software and systems engineering.

### 5.1 Innovation in Functionality

Software will remain the innovation driver in cars for the next two decades. We will see many new software-based functions in cars in the future. Each new software based function that comes into the cars enables several further features. This accelerates the development.

#### 5.1.1 Crash Prevention, Crash Safety

Already today the safety standards in cars are very high. The rates of people seriously injured or killed in their cars in accidents are decreasing in spite of increased traffic. Statistically, software in cars helped impressively to prevent accidents and many severe injuries in accidents. Nevertheless, the systems are far from being perfect by now. New generations of crash prevention, pre-crash, and crash mitigating functions are in preparation.

#### 5.1.2 Advanced Energy Management

Hybrid cars are just at their beginning. In future cars we can expect a technical infrastructure that takes care of many in-car issues like energy consumption, car calibration, and management of the electric energy available.

#### 5.1.3 Advanced Driver Assistance

The complexity of the software systems in cars for their drivers, passengers, but also for maintenance is too high. What can help is various driver assistance functions at all levels, supporting instantaneous driver reactions but also providing short term driving assistance in, for instance, lane departure or tour planning.

#### 5.1.4 Adaptable MMI

What seem less far in the future are integrated seamless adaptive MMI (Man Machine Interface) systems in cars. Cars get more

complex also due to software – but they get safer due to that software and they get more convenient. But to get an easy access to this convenience we have to offer those functions to drivers and passengers in a way where they do not have to operate all this complexity explicitly. Adaptive context aware assistance systems which grasp the situations and are able to react within a wide range without too much explicit interaction by the driver or the passengers can lead to a new quality of MMIs.

#### 5.1.5 The Programmable Car

Equipped with various actuators and sensors, as premium cars are today, we get already close to a point where we can purely by programming, by introducing new software, create new functions for cars. This comes close to the vision of the programmable car.

#### 5.1.6 Personalization and Individualization

A promising issue is personalization and individualization of cars. Drivers are quite different. When cars get more and more complex, of course, it is crucial to adapt the ways cars have to be operated to the individual demands and expectations of the users.

#### 5.1.7 Interconnection Car Networking

Another notable line of innovation is the networking of on-board and off-board systems. Using wireless connections, in particular peer-to-peer solutions, we can connect cars, which gives many possibilities to improve safety issues in the traffic or to find new solutions in the coordination of traffic far beyond the classical road signs of today. For instance, in the long-term future when we can imagine that all road signs are complemented by digital signals between the cars, we can have a completely different way of coordinating traffic.

## 5.2 Cost Reduction

The software costs in car increase enormously. These are not only pure development costs. Sometimes even more significant are maintenance costs and especially warranty costs.

## 5.3 Innovative Architectures

The car of the future will certainly have much less ECUs in favor of more centralized multi-functional multipurpose hardware, less communication lines and less dedicated sensors and actuators. Arriving today at more than 70 ECUs in a car, the further development will rather go back to a small number of ECUs by keeping only a few dedicated ECUs for highly critical functions and combining other functions into a small number of ECUs, which then would be rather not special purpose ECUs, but very close to general-purpose processors. Such a radically changed hardware would allow for quite different techniques and methodologies in software engineering.

## 6. CHALLENGES

Software issues hit the car industry in a dramatic way. In a time of only 30 years the amount of software related development activities went from 0 to 30 or even 40 %. If we assume that an engineer works about 35 to 40 years in industry, it is obvious that the companies were not able to gather sufficient competencies quickly enough. A second problem is that there are not enough software engineers educated by the universities in the skills needed in the embedded and especially the automotive domain.

The high intensity of software in cars puts the car industry under stress and a high change pressure. The reasons are manifold. First

of all, an important issue is the dependencies between the different functions in the car leading to all kinds of wanted or unwanted feature interactions.

This is quite different from what the automotive industry was used to before software came into the cars. Over a hundred years the car industry managed to make their different functionality as independent as possible such that cars could be developed and produced in a highly modular way. With the coming up of software-based functions in the cars these independence disappeared. Today a car has to be understood much more as a complex system where all the functions act together. So software engineering in cars needs to take a very massive step into systems engineering.

## 6.1 Competency and Improved Processes

The growth of software in cars means that the car industry needs new competencies. It has to master the competency management to build up software competencies as fast as needed.

On the other hand the car industry needs completely new development processes. Processes that – in contrast to those used today – are much more influenced by software issues. It is fascinating to see how the processes and models of software engineering influence more and more what is going on in mechanical engineering in the automotive domain.

### 6.1.1 From Software to Systems Engineering

In the end the whole structure and organization of the automotive industry starts to change. One issue is the amount of development done by the OEM. The general tendency is that basically all implementation is done via outsourcing. However, as long as the OEMs are interested to do the integration work themselves it is obvious that the OEM has to gain a deep understanding of the software intensive systems.

### 6.1.2 The Role of Control Theory

Traditionally control theory plays a prominent role in the car development. However, today a lot of the software in cars is not actually control theoretic but event based. The challenge here is to find the right theory and methodology to combine control theory and the engineering of discrete event systems.

### 6.1.3 Chances and Risks

The speed of the development, the complex requirements, the cost pressure and the insufficient competency in the field bring enormous challenges and risks but also high potentials and opportunities for improvements.

## 6.2 Innovation in Architecture

The enormous complexity of software in cars asks for an appropriate structuring by architectures in layers and levels of abstraction.

### 6.2.1 Functionality

One of the most interesting observations is the rich multi-functionality that we observe in a car today. In premium cars we find up to 2000 and more software based functions. Those functions address many different issues including classical driving questions but also other features in comfort and infotainment and many more. Most remarkably, these functions do not stand alone, but show a high dependency between each other. In fact, many

functions are very sensitive with respect to other functions operated at the same time.

So far, the understanding of these feature interactions between the different functions in the car is insufficient. We hope to develop much better models to understand how to describe a structured view on multi-functional systems like those found in cars.

### 6.2.2 MMI

What is obvious today and what is well understood by now is that the man machine interfaces (MMI) in cars have to be done in a radically different way. It was BMW that was brave enough to do a first step in the right direction. Their iDrive concept is very much influenced by the interaction devices of computer systems like mice or a touch pads we are used to by our computers, today. BMW got a lot of criticism for that step. In the meanwhile all its competitors have followed the same road.

But quite obviously, these are merely first steps. Multi-functionality of cars needs flexible ways of addressing and operating and interacting with all those functions.

What makes the situation more difficult than in classical computers is, of course, that car drivers cannot pay as much attention as computer users would but rather must concentrate on the traffic and driving; thus drivers should get an user interface which allows them to deal with the many functions in a car in a way that takes not too much of their attention compared to attention given to the traffic.

### 6.2.3 Complex Comprehensive Data Models

Currently in cars there is a very distributed uncoordinated data management. Each of the ECUs contains and manages its own data. But we should not think about this data as a distributed database that is well organized with some kind of data integrity. Instead, all the different ECUs and functions keep a large portion of their data separately. This can lead to a kind of schizophrenic situation in cars where some ECUs think, according to their local data, that the car is moving, while others believe that the car has stopped.

It would be an interesting exercise to design the architecture of a car in a way that there is an integrated inter-function data model that includes sensor fusion and overall car data management.

## 6.3 Development and Maintenance Processes

Certainly the development process that is needed for the development of software systems gets more and more complex. What we need is a suitable process that reduces complexity, enables innovation, saves costs, is transparent and addresses outsourcing.

### 6.3.1 Requirements Engineering

One of the biggest problems in automotive software engineering is a fitting requirements engineering. That this is essential is quite obvious because a lot of the functions in cars are innovative and completely new. When introducing new functions, of course, we have no experience with them. What is the best way to work out the detailed functionality, what are the best dialogs to access the functions, what are the best reactions of the systems? By software we get a much larger design space for solutions than in cars before. Therefore requirements engineering is one of the crucial issues (see [19], [20]).

In addition, some of the requirements engineering has to be done inside the OEMs and the supplementary requirements engineering has to be added by the suppliers, which usually carry out the implementation of the functions. Therefore the communication between OEMs and suppliers has to be organized via the requirements documents, which nowadays are often not precise and not complete enough.

This brings us to the issue of distributed concurrent engineering. Typically in the automotive industries we have a change. The more complex systems become, the more important it is to use good product models to support the integrity of the information exchange between the supplier chains.

### 6.3.2 Design

Designing the architecture in an IT system in a car means to determine the hardware architecture consisting of ECUs, bus systems and communication devices, sensors, actuators and the MMI. On this hardware structure the software infrastructure is based including the operating system, the bus drivers, and additional services. This system software forms, together with the hardware, the implementation platform.

The application software is based on the platform and consists of the application code. This shows the significance of the platform for many typical software engineering goals such as suitable architecture, separation of concerns, portability, reusability etc.

### 6.3.3 Coding

Suppliers carry out most of the coding, today. Only in extraordinary cases the OEM, for instance, produces code for some of the infrastructure (such as bus gateways).

A lot of the code is still written by hand, although some tools generate good quality code. Code generation, however, is often considered not efficient enough to exploit the ECUs in the optimal way. Highly optimized code, however, makes reuse and maintenance quite hard.

### 6.3.4 Software and Systems Integration

Since today, by their design, architecture and the interaction between the sub-systems are not precisely specified, and since the suppliers realize the sub-systems in a distributed process, it is not surprising that integration is a major challenge.

First of all a virtual integration and architecture verification is not possible, today, due to the lack of precise specifications. Second, in turn the sub-systems delivered by the suppliers do not fit together properly and thus the integration fails. Third when trying to carry out the error correction due to the missing guidelines of architecture, there is no guiding blue print to make the design consistent.

### 6.3.5 Quality Assurance

A critical issue of the car industry is quality. Since the car industry is so much cost aware, quality issues are often not observed in the way advisable for software system (see [15]). This and the application of established certification processes in the avionic industry are the reasons for airplanes' reliability outmatching cars' reliability by far.

### 6.3.6 Maintenance

A critical issue is of course that cars are in operation over more than two or three decades. This means we have to organize long-term maintenance.

#### 6.3.6.1 Compatibility

Doing maintenance for the software in cars is not so easy. Today new versions of software are brought in during maintenance by flashing techniques, i.e. replacing the software of an ECU. But doing this, one has to be sure that the new versions interoperate with the old version. In other terms we have to answer the question whether the new version is compatible (see [34]) with the one we had before. A lot of the problems we see today in cars in the field are actually compatibility problems.

#### 6.3.6.2 Defect Diagnosis and Repair

An interesting observation says that today more than fifty percent of the ECUs that are replaced in cars are technically error-free. They are just replaced because the garage could not find a better way to fix the problem. However, often the problem does not lie in broken hardware but rather ill designed or incompatible software.

Actually we need much better adapted processes and logistics to maintain the software of the cars. Understanding how we do a further development of the software architecture in cars, understanding the configurations and version management and making sure that not very well trained people in garages really can handle the systems is a major challenge.

#### 6.3.6.3 Changing Hardware

Hardware has to be replaced in cars if it is broken. Moreover, over the production time of a car model, which is about 7 years, not all the ECUs originally chosen are available in the market the whole period. Some of them will no longer be produced and have to be replaced by newer types. Already after the first 3 years of production 20 to 30 percent of the ECUs in the car typically have to be replaced due to discontinued ECUs. As a result the software has to be reimplemented, since it is tightly coupled with the ECU. Therefore portability and reusability become more and more important for car industry.

## 6.4 Hardware and Technical Infrastructure

Today in cars we find a very complex technical infrastructure. We have up to five bus systems and more. We find real time operating systems, a lot of system technical infrastructure on which the applications are based. This is why relatively simple applications get wildly complex since they have to be distributed and they have to communicate over a complex infrastructure.

One of the problems of this infrastructure is that on the bus level there is a lot of multiplexing going on. The same holds for the ECUs where there are tasks and schedulers. Actually we can find all the problems of distributed systems – and this in a situation where physical and technical processes have to be controlled by the software, some of them highly critical and hard real time.

What creates the big problems due to the multiplexing going on in the cars? In the transmission time of the messages, there is some jitter and delay such that systems appear to be nondeterministic and that in many cases time guarantees cannot be given. This is one of the reasons why we do not have more X-by-wire solutions in cars today. On one hand the reliability today is not good

enough, on the other hand the time guarantees are not good enough. Therefore a lot of interesting potentials for improvement looking at drive-by-wire systems are not realized so far.

## 6.5 Cost Issues

Traditionally the car industry is very cost aware. Competition is to a large extent determined by prices on one side and by brand image on the other side. Image is determined by design, quality, comfort, and innovation. The last three factors are heavily influenced by software in cars.

### 6.5.1 Software Cost Control

Software cost control is today of course closely related to the traditional cost per piece and production-centric cost models in the car industry. However, we observed an exponential growth of software costs in cars in recent years.

At the moment most of the software in cars is re-implemented over and over again. The reasons for that are to a large extent the optimization of the costs per part. The car industry always tries to use the cheapest processors they can find and to exploit more than eighty percent of their power and capacity. As a result, late changes bring those processors close to their limits and thus the software has to be highly optimized. This causes that the software cannot be reused from one ECU to the other.

### 6.5.2 New Players in Field

Software will become an independent sub-part in the automotive domain. This due to the fact that more and more ECUs are no longer dedicated to one application, but are multiplexing several sub-applications. This means that suppliers no longer produce integrated solutions where ECUs, sensors, actuators, software and hardware as well as the mechanical devices are developed as one integrated piece. The software then has to be delivered separately running on an ECU not delivered by the same supplier. As a result the software is rather like a device driver of the mechanical device. In fact, then there is no real reason why the software has to come from the supplier producing the mechatronic part, it may come separately from a software house. This way, software becomes an independent sub-product and sub-system for the car industry.

Due to this observation it seems very likely that new players come into the industry. Sometime ago a software house would not be an interesting first tier supplier for the embedded systems of the car industry. This is about to change radically.

### 6.5.3 Long Term Investment

An economically interesting question is who will, in the long run, own the investment that is created by the development of software in the car. It is not clear at all who will own the intellectual property for that investment and therefore who will, on the long run, be the dominant player in the industry.

### 6.5.4 Reuse and Product Lines

One of the big hopes for cost reduction in software development for the automotive domain is product line approaches. But so far, product line engineering is only used by a few suppliers and not systematic at all by the OEMs.

## 7. RESEARCH CHALLENGES

As explained, the car industry is facing many challenging issues for software in cars. This opens a wide field for research.

## 7.1 Comprehensive Architecture for Cars

Due the multi-functionality and all the related issues we need a sophisticated structural view on the architecture in cars that addresses all the aspects that are relevant. In such an architecture (see [35]), we distinguish ingredients that we briefly explain in the following.

### 7.1.1 Functionality Level – Users View

The usage view aims at capturing all the software-based functionality offered by the car to the users. Users include not only drivers and passengers but also the people in a garage and maintenance staff, perhaps even the people in the production and many more. We call this the functionality level (see [28]).

In any case, the functionality level provides specifically a perspective onto the car that captures its family of services and aims at understanding how the services are offered and how they depend on and interfere with each other. So-called feature or function hierarchies can model this best.

### 7.1.2 Design Level – Logical Architecture

The design level addresses the logical component architecture. In a logical architecture, the functionality hierarchy is decomposed into a distributed system of interacting components.

At the design level we describe the distributed architecture of a system, independent from the fact whether the components are implemented by hardware or software. The logical architecture can be described by a number of interfaces for communicating state machines with input and output that realize the functions that are found in the system. Via their interaction, they realize the observable behavior described at the functionality level. The logical architecture describes abstract solutions and to some extent the protocols and abstract algorithms used in these solutions.

### 7.1.3 Cluster Level

In the clustering we rearrange the logical architecture in a way that prepares the deployment and the step towards the software architecture.

### 7.1.4 Software Architecture

The software architecture consists of the classical division of software in platforms like operating systems and bus drivers on one side and the application software represented by tasks, which are scheduled by the operating system, on the other side. This software has to be deployed onto the hardware.

### 7.1.5 Hardware Level – Hardware Architecture

The hardware architecture consists of all the devices including sensors, actuators, bus systems, communication lines, ECUs, man machine interface and many more.

### 7.1.6 Deployment – Software/Hardware Codesign

Finally we need a deployment function that relates hardware to software. The hardware/software and the deployment function together have to represent a concrete realization of the logical architecture that just describes the interaction between the logical components.

### 7.1.7 Architecture Modeling and Description

A modeling approach has to be expressive enough to deal with all the mentioned aspects of architecture.

## 7.2 Reducing Complexity

One of the biggest problems in car industry today is the overwhelming complexity of the systems they face today. How can we reduce complexity?

Of course, we can use classical techniques from software engineering, which is structuring, separation of concerns, and abstraction. Structure is what we have achieved if we get an appropriate architectural view with the levels as described above. Abstraction is what we gain if we use models. Model orientation remains one of the big hopes for the car industry to improve its situation.

## 7.3 Improving Processes

A key issue is process orientation and software development processes. So far the processes in the car industry are not adapted to the needs of software intensive systems and software engineering. Process orientation will introduce a much higher awareness for processes.

### 7.3.1 Maturity Levels

A good example is the introduction of Spice and CMMI techniques into the car industry, which already has helped a lot to improve the competencies there.

Actually, a deep process orientation on the long run would need a well-understood handling of the product models. The product data of course need a comprehensive coherent and seamless model chain. Here we find an exciting dependency between engineering support software and embedded on board software.

## 7.4 Seamless Model Driven Development

One of the great hopes for improvement is seamless model driven development. This means that we work with a chain of models for the classical activities:

- Requirements modeling
- Design modeling
- Implementation modeling
- Modeling test cases

A vision would be an integrated modeling approach where the relationship between all the models is captured and parts of the next models are generated from the models before.

### 7.4.1 The Significance of Models

Models, if they are formalized and have a proper theory, are useful in many respects. Starting with documentation, formalization and making informal descriptions precise we can go on with analysis, reuse, transformation, code generation, and finally enter into product lines. In particular, models are the key to tooling and automation.

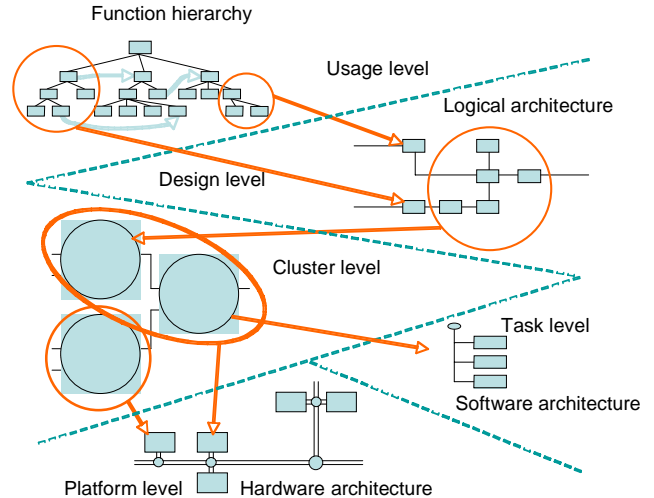


Figure 2. Comprehensive Architecture

Finally, models capture development knowledge and give guidelines for the development processes.

### 7.4.2 Weaknesses of Modeling Today

Today models and model-based development are used to some extent in the automotive industry but their use is fractal. Modeling is applied only at particular spots in the development process. So a lot of its benefits get lost that could be exploited if we had integrated model chains.

Since the models are only semiformal and the modeling languages are not formalized, a deeper benefit is not achieved. Typical examples are consistency checking or the generation of tests from models (see [17]). Another issue is that models could help very much in the communication between the different companies such as OEMs and first and second tier suppliers; also here models are used only in a limited way so far.

Pragmatic approaches such as UML (see [30]) and other modeling approaches used in industry are not good enough. Such approaches are not based on a proper theory and sufficient formalization. As a result, tool support is weak, possibilities to do analysis with models are weak, and the preciseness of the modeling is insufficient.

It is important to keep in mind that modeling is more than documentation. UML should rather be called UDL: „Unified Documentation Language“. So far UML is not a full modeling language. It does not offer a good basis for automation and refinement.

Modeling can help to improve the quality of the automotive software systems. As well known, quality has a wide spectrum of different aspects and facets.

Unfortunately, what we can find in modeling today can not so much help to improve the quality of the development processes and of the software intensive products because it does not give precise and uniform views onto systems based on theories.

### 7.4.3 Potentials of Modeling

Using models in an integrated seamless way, we come up with a vision of a strictly model based process in the automotive industry. The idea is as follows:

After first business requirements are captured and the most important informal requirements are brought together, we split the requirements into functional and nonfunctional requirements. Functional requirements are formalized by a function hierarchy where all the atomic features are described by interacting state machines or by interaction diagrams between the different functions. In the function hierarchy, dependency relations are introduced.

The non-functional requirements are process requirements and quality requirements for the product. For the product quality requirements a quality profile is drawn up. We have to understand how the quality requirements determine properties of the architecture. This way we form a quality driven architecture onto which we map the functional requirements.

Along this line we arrive at system decomposition. In the decomposition we have to specify a logical architecture and the interfaces (see [33]) of its logical components. At that level we can already do a proof of concept in terms of a virtual verification of the logical architecture as long as we have formal models for the logical components and their composition. This step already proves that at the application level the architecture is correct.

From here on we do the decomposition of the components in software parts, we design the hardware architecture and we design the deployment.

## 7.5 Integrated Tool Support

Tool support is a key issue. Today we find a rich family of tools in use but unfortunately these tools are not integrated. Therefore there are a number of attempts to create pragmatic tool chains by connecting the tools in a form where the data models of one tool are exported and imported by the next tool. Unfortunately this does not help if there is not a semantic understanding how the different tools are based on joint concepts.

## 7.6 Learning from Other Domains

In the automotive industry we find basically all the problems we find software engineering, in general, sometimes even in a more crucial way. It is an appealing question, what we can do to bring solutions and ideas from software engineering in general into the automotive domain to try them out there, to improve them and on the other hand to get some feedback and maybe improvements and new ideas for software engineering as such.

## 7.7 Improving Quality and Reliability

Today the reliability of software in cars is insufficient. In avionics reliability numbers of  $10^9$  hours mean time between failures and more are state of the art. In cars we do not even know these figures. Only adapted quality processes can improve the situation. As an example, Toyota is quite successful with reviews based on failure modes.

### 7.7.1 Automatic Test Case Generation

The amount of quality assurance in cars is enormous. Today the industry relies very much on hardware and software as well as system in the loop techniques (HIL/SIL). There sub-systems are executed under simulated environments in real time. However,

this technology is coming to its limits because of the growing combinatorial complexity of software in cars.

More refined test techniques can help here, where test are generated based on models.

### 7.7.2 Architecture and Interface Specification

A critical issue for the precise modeling of architectures is the mastering of interface specifications. So far interface specification methods in software engineering in general are not good enough. This fact is, in particular, a disaster for the car industry, because of its distributed mode of development.

The OEM has to do the logical architecture today and then distributes the development of the components that correspond to the components of the logical architectures to the suppliers. The suppliers do the implementation, even supply the hardware and bring back pieces of hardware and software that then have to be integrated into the car and connected to the bus systems. Since the interfaces are not properly described, the integration process gets into a nightmare. A lot of testing and experimentation has to go on, a lot of change processes have to be needed to understand and finally work out the integration process.

### 7.7.3 Error Diagnosis and Recovery

Today the amount of error diagnosis and error recovery in cars is limited. In contrast to avionics, where hardware redundancy and to some extent also software redundancy is used, in cars we do not find an extensive error treatment. In the CPUs some error logging takes place, but there is neither any consideration nor logging of errors at the level of the network and the functional distribution nor is there neither a comprehensive error diagnosis nor any systematic error recovery.

There are some fail safe and graceful degradation techniques found in cars today, but there is not a systematic and comprehensive error treatment.

One result of this deficiency is problems in maintenance. Today – as mentioned above – in the cause of repair of a defect more than 50 % of the hardware devices that are replaced in garages are physically and electro-technically without defects. They are changed, since a successful diagnosis, error tracing, and error location did not work and thus by replacing the CPU software is replaced, too, such that the error symptom disappears – but often further, different errors show up later.

### 7.7.4 Reliability

Another critical issue is reliability. We do actually not know how reliable our cars are in terms of mean time to failure. In fact there are no clear reliability numbers and figures. In contrast to the avionic industry there is no serious reliable calculation for cars.

The high reliability in the avionic field is of course, to a large extent, due to the fact that they use very sophisticated error tolerance methods, redundancy techniques and at the same time they work with a sophisticated way of error modeling (like FMEA). In cars today errors are only captured within processors in so called error logging stores.

What we need on the long run are comprehensive error models in cars and also software that for the detection of errors. On such models we can base techniques to guarantee fail safe and graceful degradation and in the end also error avoidance by the help of redundancy. Another issue is error logging to get into a better error diagnosis for maintenance.



## 7.8 Software and System Infrastructure

An important step in car industry is to go away from proprietary solutions and to develop standards. A promising step in that direction is the AUTOSAR project (see [4], [5], [31]) that defines a uniform platform architecture. AUTOSAR is only one small step, however.

## 8. RESEARCH AT TUM

In our research group at TUM we started research in the area of automotive software engineering more than 10 years ago. Over one decade, we developed, step-by-step, very tight research co-operations with chief OEMs and suppliers leading to a kind of strategic partnership.

On the one hand, our research agenda is strongly influenced by burning questions in industry. On the other hand, our research results are accepted as helpful contributions and input to improve the situation. This enables also a lot of fruitful joint research and development.

### 8.1 Foundations: Distributed System Models

Software in car forms a concurrent, distributed, interacting, often hard or at least soft real time system. Modeling and understanding such systems lies in the center of software and systems engineering. What we need there is a comprehensive theory of modeling as a basis for capturing the functionality and the architecture. We have worked out a large part of such a theory of modeling in recent years (see [11], [12], [13], [16]).

It has to contain a theory of functions, components, features, interfaces, state machines, function combination and component composition, property, interface as well as time granularity refinement, hierarchical decomposition and architecture, communication, levels of abstraction, architectural layering, and interaction processes and how all these are connected on a conceptual basis which includes a notion of time.

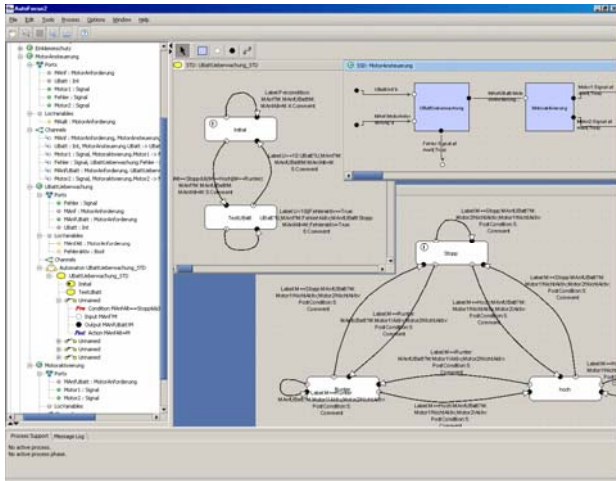


Figure 3. Screenshot from the Tool AutoFocus 2

If we work such a theory the right way we gain all what is needed to attack many of the issues, we need to understand to be able to model and develop automotive software systems in a systematic way.

### 8.2 Adapted Development Process

From our observations we aimed at improving the development process, in particular, adding roles for the principal and the

supplier into a software lifecycle model, the V-Model XT (see [18], [24], [27]) developed by our group for the German Government.

#### 8.2.1 Requirements Engineering

Requirements engineering has to address the needs of multi-functional systems and concurrent distributed engineering. In particular, such models are needed to support requirements engineering.

#### 8.2.2 Architecture

We have developed a comprehensive view onto architectures defining levels and layers of abstraction along the lines described above (see [6]).

#### 8.2.3 Methodology

In a strictly model based development we support all steps in the development process via models and techniques of validation, verification, transformation, and generation.

#### 8.2.4 Testing

Automatic test generation is a very promising approach. Once models have been created, we can generate test cases from them, leading to a much better coverage and much deeper test impact.

#### 8.2.5 Verification

Today we are at the point where we do verification for industrial type software systems in cars. An ambitious approach is provided by the Verisoft project (see [9]) where we verify the automatic emergency call function, which is found in cars today, in all its details. The example of the emergency call function is verified by modeling it together with the used infrastructure like FlexRay, the real time operating system OSEKtime, and the code running on processors in all details. In Verisoft we use Autofocus as the modeling tool and Isabelle as the verification engine.

## 8.3 Tool Support

We have developed prototyping tools like AutoFocus for design and its enhancement AutoRaid (see [32]) for requirements engineering (see [1], [2], [3]).

Figure 3 shows a screen shot of the AutoFocus tool, with extensions like AutoFlex especially target the development of embedded control applications.

## 9. CONCLUSION

It is absolutely clear that software in cars is one of the big challenges and at the same time one of the interesting fields of research for the software engineering community. Much can be gained in this area, much can be learned in this area, much can be contributed by well-targeted research. We consider automotive software engineering as one of the very challenging fields where we have the chance to get into a fruitful dialog between application domain and software engineering experts.

Software and systems engineering for embedded systems in cars is one of the major challenges for software engineering research today. In a car we find many of the great research issues for software engineering in a nutshell.

### Acknowledgement

I am grateful to a number of people, including Martin Rappl, Doris Wild, Christian Kühnel, Jan Jürjens, Alexander Pretschner, Christian Salzmann, Hans Georg Frischkorn, Bernhard Schätz, Florian Deissenböck as well as engineers from BMW, Daimler Chrysler, Ford, Honda, Porsche, Audi, Volkswagen, ESG, Validas

AG, Bosch, and Siemens VDO for stimulating discussions on topics of automotive software engineering.

## 10. REFERENCES

- [1] AutoFOCUS – Webseite, <http://autofocus.in.tum.de>, 2005.
- [2] AutoFOCUS 2 – Webseite, <http://www4.in.tum.de/~af2/>, 2006.
- [3] AutoRAID – Webseite, <http://www4.in.tum.de/~autoraid/>, 2005.
- [4] AUTOSAR consortium: [www.autosar.org](http://www.autosar.org), 2005
- [5] AUTOSAR Development Partnership. [www.autosar.com](http://www.autosar.com)
- [6] A. Bauer, M. Broy, J. Romberg, B. Schätz, P. Braun, U. Freund, N. Mata, R. Sandner, and D. Ziegenbein. Auto-MoDe—Notations, Methods, and Tools for Model-Based Development of Automotive Software. In *Proceedings of the SAE 2005 World Congress*, Detroit, MI, Apr. 2005. Society of Automotive Engineers
- [7] M. Bloos: Echtzeitanalyse der Kommunikation in Kfz-Bordnetzen auf Basis des CAN Protokolls, Dissertation TU München, 1999.
- [8] BMW Group: Annual Report 2004.
- [9] J. Botaschanjan, L. Kof, Ch. Kühnel, M. Spichkova: Towards Verified Automotive Software. ICSE, SEAS Workshop, St. Louis, Missouri, USA May 21, 2005
- [10] M. Broy, K. Stølen: Specification and development of interactive systems – FOCUS on Streams, Interfaces and Refinement, Springer, 2001.
- [11] M. Broy: Automotive Software Engineering. Proc. ICSE 2003, pp. 719-720.
- [12] M. Broy: Modeling Services and Layered Architectures. H. König, M. Heiner, A. Wolisz (Eds.): Formal Techniques for Networked and Distributed Systems. Berlin 2003, Lecture Notes in Computer Science 2767, Springer 2003, 48-61
- [13] M. Broy: Service-oriented Systems Engineering: Specification and Design of Services and Layered Architectures - *The JANUS Approach*. In: Manfred Broy, Johannes Grünbauer, David Harel and Tony Hoare; Engineering Theories of Software Intensive Systems, Marktoberdorf, 3 – 15 August 2004, Germany, NATO Science Series, II. Mathematics, Physics and Chemistry – Vol 195, Springer
- [14] M. Broy, A. Pretschner, C. Salzmänn, T. Stauner: Software-intensive systems in the automotive domain: challenges for research and education. SAE 2006, to appear
- [15] M. Broy, F. Deißeböck, M. Pizka: A Holistic Approach to Software Quality at Work. 3rd World Congress for Software Quality (3WCSQ) J. Dannenberg, C. Kleinhans.: The Coming Age of Collaboration in the Automotive Industry, Mercer Management Journal 18:88-94, 2004.
- [16] D. Herzberg, M. Broy: Modeling layered distributed communication systems. Applicable Formal Methods. Springer Verlag, Volume 17, Number 1, May 2005
- [17] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner (Eds.); Model-Based Testing of Reactive Systems, Advanced Lectures, Springer-Verlag Berlin Heidelberg 2005
- [18] Manfred Broy, Andreas Rausch: Das Neue V-Modell XT, Informatik Spektrum A 12810, Band 28, Heft 3, Juni 2005
- [19] A. Fleischmann, J. Hartmann, C. Pfaller, M. Rappl, S. Rittmann, D. Wild: Concretization and Formalization of Requirements for Automotive Embedded Software Systems Development, In: The Tenth Australian Workshop on Requirements Engineering (AWRE), Melbourne, Australia, K. Cox, J.L. Cybulski et.al (ed.), 2005, 60-65
- [20] E. Geisberger: Requirements Engineering eingebetteter Systeme – ein interdisziplinärer Modellierungsansatz, Dissertation TU München, 2005
- [21] K. Grimm: Software Technology in an Automotive Company—Major Challenges. Proc ICSE 2003, pp. 498-505.
- [22] B. Hardung, T. Kölzow, A. Krüger: Reuse of Software in Distributed Embedded Automotive Systems. Proc. EMSOFT'04, pp. 203—210, 2004.
- [23] H. Heinecke: Automotive System Design—Challenges and Potential. Proc. DATE'05.
- [24] M. Kuhrmann, D. Niebuhr, A. Rausch: Application of the V-Modell XT - Report from A Pilot Project. In: Unifying the Software Process Spectrum, International Software Process Workshop, SPW 2005, Beijing, China, May 25-27, Mingshu Li, Barry Boehm, Leon J. Osterweil (ed.), pp. 463-473, Springer, 2005
- [25] A. Pretschner, C. Salzmänn, T. Stauner: SW engineering for automotive systems at ICSE'04. Software Engineering Notes 29(5):5-6, 2004.
- [26] A. Pretschner, C. Salzmänn, T. Stauner: SW engineering for automotive systems at ICSE'05. To appear in Software Engineering Notes, 2005.
- [27] A. Rausch, Ch. Bartelt, Th. Ternité, M. Kuhrmann: The V-Modell XT Applied – Model-Driven and Document-Centric Development. In: 3rd World Congress for Software Quality, VOLUME III, Online Supplement, 2005, pp. 131 - 138
- [28] S. Rittmann, A. Fleischmann, J. Hartmann, C. Pfaller, M. Rappl, D. Wild: Integrating Service Specifications on Different Levels of Abstraction, In: IEEE International Workshop on Service-Oriented System Engineering (SOSE), IEEE (ed.), IEEE, 2005.
- [29] Robert Bosch GmbH. CAN Specification Version 2.0, 1991.
- [30] C. Salzmänn, T. Stauner: Automotive Software Engineering. Languages for system specification: Selected contributions on UML, systemC, system Verilog, mixed-signal systems, and property specification from FDL'03, pp. 333-347, Kluwer 2004.
- [31] C. Salzmänn, H. Heinecke, M. Rudorfer, M. Thiede, T. Ochs, P. Hoser, M. Mössmer, A. Münnich. Erfahrungen mit der technischen Anwendung einer AUTOSAR Runtime Environment, VDI Tagung Elektronik im Kraftfahrzeug, Baden-Baden, 2005.
- [32] B. Schätz, A. Fleischmann, E. Geisberger, M. Pister. Model-Based Requirements Engineering with AutoRAID. In: Informatik 2005, Springer, 2005.
- [33] B. Schätz, Interface Descriptions for Embedded Components. In: Object-oriented Modeling of Embedded Real-Time Systems (OMER'05), Paderborn 2005
- [34] T. Stauner. Compatibility Testing of Automotive System Components. 5<sup>th</sup> Int. Conf. on SW Testing (ICSTEST), Düsseldorf, 2004.
- [35] D. Wild, A. Fleischmann, J. Hartmann, C. Pfaller, M. Rappl, S. Rittmann: An Architecture-Centric Approach towards the Construction of Dependable Automotive Software, SAE Technical Paper Series 2006, Detroit, 2006