Model-Based Development and Testing of Automotive Software – Some Experiences and Challenges

MBT, Berlin 20.10.2011

Matthias Weber – Carmeq GmbH
1. Model-Based Development at Carmeq

2. Stochastic Robustness Testing

3. AUTOSAR

4. The Challenge of Variability

5. The EAST-ADL Architecture Description Language
Motivation

- Developing software-based vehicle functions of increasing complexity
- Cost and quality requirements necessitate efficient development methods
- Model-based development methods (since the 90s) using MATLAB/Simulink/Stateflow and code generation

Goal

- Seamless model-based function and software development
- Avoiding ruptures: evolutionary development of models from concept phase to final code generation
Use of Models in the Development Process

- Model-based Specification
- Model-based Concept Analysis
- System-/Function Design
- Code Generation
- Model-based Software Development
- Implementation
- System-/Functions-Requirements
- Simulation
- Plant and Test model

Concept Model

Behaviour Model

Implementation Model
Relation between Model and Requirements – Ideal Case of Function-Oriented Development

Anforderungen

Funktionsbereich (z.B. Licht und Beleuchtung)

1. Funktionsfamilie (z.B. Blinken)
   1.1 Basisfunktion (z.B. Richtungsblk)
   Eingaben/Ausgaben/Anforderungen

   1.2 Basisfunktion (z.B. Warnblinken)
   Eingaben/Ausgaben/Anforderungen

   ...

2. Funktionsfamilie (z.B. FahrBremsStandlicht)
   1.1 Basisfunktion
   ...

Modell

<<umgesetzt durch>>

1. Basis-Modul1

2. Basis-Modul2

Blinken

FahrBremsStandlicht

LichtUndBeleuchtung
Properties of Functional Model vs. Implementation Model

Functional Model:
- Focus on functional system requirements
- Not all aspects modeled (signal preprocessing, diagnosis,...)
- Structuring according to requirements structure
- Hardware independent
- Data not scaled

Implementation Model:
- Functional and non-functional SW-Requirements
- Usage of resources (time, storage)
- Complete functionality incl. of error handling, signal pre-processing etc. modelled
- Interface to basic software
- Typing and scaling of Data
- largely hardware-independent
- Criteria: Maintainability, Extensibility, Testability, „Separation of Concerns“
Transition from a Functional Model to an Implementation Model

Problematic Changes:
- Change of model architecture for satisfaction of non-functional requirements
- Structural changes to satisfy resource requirements
- Functional extensions, concerning the inner Model structure such as z.B. error treatment, initialisation.

Evolutionary (unproblematic) Changes:
- Functional extensions such as z.B. signal pre-processing, plausibility checking etc.
- Interface to basic software of the ECU
- Scaling of Data
- Local optimisations to lower resource consumption
Example: Extensions for Error Handling and Diagnosis

Functional Model

- Focus on core algorithm
- Conscious decision not to model diagnosis and error handling
Example: Extensions for Error Handling and Diagnosis

Implementation Model

- Significant adaptations to the core algorithm

Extension: Shutdown

Extension: Sensor Watchdog
Lessons Learned

Early validation of model architecture wrt. non-functional requirements
- Early review of functional models by software experts
- Guidelines for model structuration, e.g. using domain specific patterns (e.g. for error handling)

Systematic relation between functional model and implementation model
- Functional model not necessarily structured according to requirements structure, use of tool-supported requirements linking, adapted requirements structuring

Many further aspects
- Separation of functionalities according to available computation slots
- Exclusion of modeling constructs (Switch blocks etc.)
- …
Agenda

1. Model-Based Development at Carmeq
2. Stochastic Robustness Testing
3. AUTOSAR
4. The Challenge of Variability
5. The EAST-ADL Architecture Description Language
Motivation and Goals

Background

Growing complexity of ECU functions

- Rear blind
- Central locking
- Braking lights
- Windshield wiper
- ...

Properties of classical function testing

- Requirements-based test cases
- Sequential process: function after function
- Focus on positive testing
- Pattern: systematic stimulation – waiting period - measurement
Motivation and Goals

Motivation

Sporadic misfunctions not detected by a classical functions test, e.g.

- Brake light goes off, due to rear blind being activated
- Brake lights blink asynchronously during wiping and thus acts as a lane changing signal …

These misfunctions often result from unintended dependencies between functions

- Competing access to limited resources, e.g.
  - Computation time
  - Common SW Services
- Interaction of various individual functions

Lack of predictability which situations and which inputs lead to sporadic misfunctions
Motivation and Goals

Goals

Improvement of Robustness of ECUs based on a new test approach
- Detection of sporadic failures
- Stimulation of ECU with “unusual“ input data

Robustness:
„Property to work - in unusual situations - in a defined manner and to produce meaningful reactions“ (Liggesmeyer)

Extension to classical function test
Motivation and Goals

Technical Challenges

Input complexity
- ignition
- light control
- directional blinking
- warning blinking
- access control
- wiper

Evaluation of test runs
- continuous data
- measurement imprecisions
  - value deviation
  - time delays
- missing expected values
New test approach
Stochastic rule-based robustness test (SRT)

- **Stochastic test data generation**
  - No individual requirements-based test cases
  - Parallel, stochastic activation of input signals
  - Multiple test scenarios, similar to long-term vehicle tests, but better control and analysis means

- **Rule-based evaluation**
  - General system properties
  - Continuous checking of criteria

\[ f(x, t, \text{dice}) \]

- all signal combinations allowed
- check of selected criteria
Stochastic rule-based robustness test

Test data generation

Space of all possible test inputs

Control data for:
- rear blind control
- central locking
- brake light control
- wiper
- ...
  including timing and interaction

Choose typical use cases for each application

- rear blind control: move up/down
- central locking: unlock/lock
- brake light control: normal/emergency braking
- wiper: fast/medium/slow
- ...

Parallel, stochastic activation of the different functions
**Stochastic rule-based robustness test**

Test data generation: example light switch

Light switch positions define „States“

<table>
<thead>
<tr>
<th>Lichtdrehsteller</th>
<th>Aus</th>
<th>Parklicht</th>
<th>Abblendlicht</th>
<th>Fernlicht</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aus</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Parklicht</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Abblendlicht</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Fernlicht</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Transition probabilities**

<table>
<thead>
<tr>
<th>Lichtdrehsteller</th>
<th>Aus</th>
<th>Parklicht</th>
<th>Abblendlicht</th>
<th>Fernlicht</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aus</td>
<td>0,2</td>
<td>0,2</td>
<td>0,1</td>
<td>0,2</td>
</tr>
<tr>
<td>Parklicht</td>
<td>0,6</td>
<td>0,3</td>
<td>0,3</td>
<td>0,3</td>
</tr>
<tr>
<td>Abblendlicht</td>
<td>0,1</td>
<td>0,4</td>
<td>0,5</td>
<td>0,4</td>
</tr>
<tr>
<td>Fernlicht</td>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
<td>0,1</td>
</tr>
</tbody>
</table>
Stochastic rule-based robustness test

Parallel test execution and evaluation

- Single automata for each function
- Parallel test data generation
- Pseudo random numbers => reproducible

Test run evaluation

- Specification of assertions (using simple rules)
  - All assertions are constantly checked during test runs
  - Deviations/problems are detected and stored
Stochastic rule-based robustness test

Rule-based evaluation

Individual functions are usually well-tested, but not the cooperation of functions

Problems are often deviations to general intuitive behavior

- Left and right braking light are not synchronous???
- Backlight blinks very shortly if the gear switch is used???
- The right back light is blinking too long, if the rear blind is moving???

Problem:
Due to different communication channels of signals, there can be delays

Example: signals, which are cyclically send over the network are eventually received later due to bus load.
Stochastic rule-based robustness test

Example of evaluation rule

Rule to be checked
Left and right braking lights are simultaneously activated, delays of max. 50 msec are tolerated.

Pattern *SYNCHRON*

\[ \text{SYNCHRON(Bremslicht\_Links, Bremslicht\_Rechts, 50ms)} \]

Rule to be checked
Blinker may blink for at most 500 +/- 50 msec.

Pattern *HOLDS\_AT\_MOST*

\[ \text{HOLDS\_AT\_MOST(Blinker\_Vorne\_Links, 500ms, 50ms)} \]
Implementation

Test environment

Intelligent, random generated test data
→ Parallel Markov Automata

Assertion checking

<table>
<thead>
<tr>
<th>Zusicherung</th>
<th>Formulierung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blinker synchron</td>
<td>SYNCH(Blinker_Vo_Li, Blinker_Hi_Li, 50ms)</td>
</tr>
<tr>
<td>Blinker leuchten maximal 500 ms</td>
<td>HOLDS_AT_MOST(Blinker_Vo_Li, 500)</td>
</tr>
<tr>
<td>Fernlicht an bei Zündung</td>
<td>IMPLIES(K15==1 &amp;&amp; FernlichtSchalter, FernLicht, 50)</td>
</tr>
<tr>
<td>Quittierungsblinken</td>
<td>CAUSALLY_IMPLIES((K15==0 &amp;&amp; TürAuf), Blinker_Vo_Li, 40))</td>
</tr>
</tbody>
</table>
Implementation

Technical Realisation of Test Environment

Test data generation

C#

Test data generation

C#

Test evaluation

Integration

Automatic Model Generation

Assertions

phys. Adaption
D/A

Prüfstand

invers(a,b)

ident(c,d)

r

Test evaluation

phy. Adaption D/A

CANoe

Technical Realisation of Test Environment

MBT 2011 23
Agenda

1. Model-Based Development at Carmeq
2. Stochastic Robustness Testing
3. AUTOSAR
4. The Challenge of Variability
5. The EAST-ADL Architecture Description Language
“AUTOSAR (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers.”

www.autosar.org
**Scope**

**SW-Development Methodology**
- "Function-based" software development
- Several abstraction layers, e.g. "Virtual Functional Bus"
- Use of modern SW-engineering-methods (Meta-Modelling, UML, XML ...)

**ECU-SW-Architecture**
- Common base architecture for Control-, Sensor, Actuator-ECUs
- Layeres SW-Architecture: µC – BSW – RTE – Application
- Aims to support all relevant automotive µCs and ECU-interfaces
- Basic Software consists of about ca. 50 Modules

**Application-Interfaces**
- Body and Comfort
- Powertrain
- Chassis Control
- Occupants and Pedestrians Safety
- Multimedia, Telematics and Human-Machine-Interfaces...
Use of AUTOSAR at VW

Releases for MQB: AUTOSAR 3.1 Rev. 002 + Extensions

ECUs already using AUTOSAR:
- „Kombiinstrument“ vby JCI (MQB)
- „Lademanager“ (VW360/7)
Phases of the AUTOSAR-Methodology

Funktion Level.

Function Design

- Definition of functional architecture on an abstract level
- Base Concept: „Virtual Functional Bus“ (VFB)
  - Software-Components (SWCs)
  - Communication (Ports + Data Elements)

Function Realisation

- Implementation of function using generated header-files based on VFB-model
(1) Integration of model-based software development with AUTOSAR

- **Current Situation:** proprietary interfaces to ECU
- AUTOSAR offers standardised interfaces and base services such as
  - ECU management,
  - Network management,
  - Storage management,
  - Diagnostic management

- Not all base services can be directly mapped to Simulink
  - Development methodology of Simulink is dataflow-oriented rather than service-oriented, i.e. procedural communication (Client/Server) is difficult to realise.
  - No out-of-the-box solutions for all base services available

⇒ **Possible Solution:** Declarative specification of functions incl. base services for automatic code-generation
(2) Adapting Software Component Testing to AUTOSAR

Current Situation
- Testtools chains adapted for every ECU and supplier

AUTOSAR
- Separation of software parts and standardised interfaces allow for common testing process and tool chain.
  - Use of current tool chain for testing functional software
  - Testing of generated communication with base services (more early)
  - Testing of complete SWC
  - Testing of SWC in the complete system

→ Testing can be performed more early
→ Tool chain can be reused
(3) Implications of SW-Development using AUTOSAR

Current Situation
- Currently *monolithic* functions are exchanged with suppliers

Function Design
- Optimisation of partitioning of software components
  - Standardised AUTOSAR interfaces allow for moving SWCs between tasks and ECUs
  - Better testability due to lower complexity of individual components

Variability Management
- Partitioning of software components improves variability management:
  - Finer-grain modeling of variants
  - Less variants for individual components
  - Better management of variants and their dependencies

Cross-manufacturer functionality capsules
Capsules for vehicle and brand-specific functionality
Agenda

1. Model-Based Development at Carmeq
2. Stochastic Robustness Testing
3. AUTOSAR
4. The Challenge of Variability
5. The EAST-ADL Architecture Description Language
Development Objects of an Electronic Platform

> 100k requirements objects

Hundreds of other artifacts

> 100k function blocks

> 100k test cases
Multi-Dimensional PL-Decomposition – Example
Reuse

State of the Art:

Project → copy & modify → new Project

Goal:

Repository

Granularity of Resuable Assets?
Explicit Representation of Variability

Representation of Dependencies between Coding Values by variability modeling, e.g. feature trees

Zentralverriegelung

Tankdeckeltaster

Tankdeckelschloß

Federsteller

Stellmotor

ZV_bTKLStellMot_COV

needs

ZV_bTKLStellMot_COV = 0

ZV_bTKLStellMot_COV = 1

Var1_COV

Var2_COV

Var3_COV

VarN_COV

Funktional Model
Reuse of Models

State of the Art:

- Function-oriented development, i.e. functional requirements are structured in technical parts
- Functions in the specifications use the same name and the same decomposition
- Running projects lead to project specific differences
- Differences must be considered in retrospect
Goal: compromise between ...
- 1 global product line
- n independent product lines

Reference model

Defines references
(non-mandatory or mandatory)

Modul tool box

Referencing models

Vehicle Project A
- Tempomat
  - Standard
  - Adaptiv
  - Sparsam

Vehicle Project B
- Tempomat
  - Standard
  - Adaptiv
  - Reduzierter Bauraum
  - Radar

conform or deviate
Tool Support Multi-Level Concept

CVM-Framework (Eclipse Plugin)

- Multi-Level Feature Models:
  - Show deviations
  - autom. conformity checking

MuLe (Doors-Extension)

- Multi-Level Concept for specification objects:
  - Show deviations
  - autom. conformity checking
  - bidirectional transfer of deviations (top-down and bottom-up reuse)
Structuring of Feature Models

Current State:
Problems

Inappropriate for customers
- Complexity (>10,000 Features)
- Technical Perspective

Direct connection too rigorous
- Multitude of actors
- Diverging life cycle and validities
- Heterogenity (methods, tools, ...)

Connection to artifact too coarse
- Some artifacts very large
- Above problems within an artifact!
Reuse Approach: Configuration Links

Interface
„Artefact Line“
Benefits

CruiseControl
from vehicle speed sensor
... thresholdValue:Float
rsThr:Threshold
from Radar
... isTooNear
[currSpeed]
ThrottleCalculator
... to throttle

BodyElectronicsSystem

ComfortCC
 Short, Long, Other: Int

Comfort
from Radar

Model
A-Class E-Class C-Class

Market
US EMEA

Car
A-Class E-Class C-Class

Adaptive

MinDistance:Int

ComfortCC

Gradual Shift of Viewpoint
Reduction of Complexity

MBT 2011
Connection of CVM to development artifacts

- Identification von artifact-specific variability components
- Transformation of CVM Entity-Types resp. –Parts
  - Type-Instance-Structure must be considered
- Construct Feature-Models + Configuration Links for Artifact
- Backtransformation into proprietary artifact format incl. added information
- Bidirectional Transformation without information loss
Agenda

1. Model-Based Development at Carmeq
2. Stochastic Robustness Testing
3. AUTOSAR
4. The Challenge of Variability
5. The EAST-ADL Architecture Description Language
The Challenge

Product Related Challenges
- Functionality increase
- Complexity increase
- Increased Safety-criticality
- Quality concerns

Challenges Related to Development Process
- Supplier-OEM relationship
- Multiple sites & departments
- Product families
- Componentization
- Separation of application from infrastructure
- Safety Requirements, ISO 26262
A System Modeling Approach/Architectural Framework that

- Is a template for how engineering information is organized and represented
- Provides separation of concerns
- Embrace the de-facto representation of automotive software – AUTOSAR

**EAST-ADL**

- Feature content
- Abstract functional architecture
- Functional architecture, HW architecture, platform abstractions
- AUTOSAR Software architecture
- Embedded system in produced vehicle (not in model)
Vehicle Level

- A Vehicle is characterized by a set of Features
- Features are *stakeholder* requested functional or non-functional characteristics of a vehicle
- A Feature describes the "what", but shall not fix the "how"
- A Feature is specified by requirements and use cases
- From a top-down architecture approach the features are the configuration points to create a vehicle variant
Analysis Level

Analysis Level is the abstract Functional description of the EE system:

- Realizes functionality based on the features and requirements
- Captures abstract functional definition while avoiding implementation details
- Defines the system boundary
- Environment model and stakeholders define context
- Basis for safety analysis
Design Level captures the concrete functional definition with a close correspondance with the final implementation

- Captures functional definition of application software
- Captures functional abstraction of hardware and middleware
- Captures abstract hardware architecture
- Defines Function-to-hardware allocation
The Implementation Level represents the software-based implementation of the system
- Software components represent application functionality
- AUTOSAR Basic software represents platform
- ECU specifications and topology represent hardware
- Model is captured in AUTOSAR
  - Software component template
  - ECU resource template
  - System Template
Environment Model

The Environment model captures the plant that the EE system control and interact with

- In-vehicle, near and far environment is covered
- Same Environment Model may be used on all abstraction levels
- Different Environment models may be used depending on validation scenario
Traceability between abstraction levels

Realization relations identify which abstract element is realized by a more concrete entity.

- Functions on analysis level realizes features on vehicle level
- Functions on design level realizes functions on analysis level
- SW components or runnables on implementation level realizes functions on design level
Requirements – Basic Relations

- Behavioral Models that specify the requirement in more detail
  - Refine

- System Components which have to satisfy the requirement
  - Satisfy

- Requirement
  - Derive
  - Verify

- New requirements derived after a system decomposition or system refinement

- V&V-Cases that verify the requirement
**Vehicle Level**

**Feature**

**ABS**

**Requirement**

**ABS Functionality**

ABS shall reduce break distance in all driving condition

**Use Case**

**Slippery Road**

**Analysis Level**

**Analysis Function Type**

**ABS Block**

**Requirement**

**ABS Activation**

ABS shall detect individual wheel acceleration

**Requirement**

**ABS Control**

ABS shall control break force via wheel slippage control

**Design Level**

**Design Function Type**

**ABS Slippage controller**

**Requirement**

**ABS Slippage**

ABS shall control Slippage with PI Controller
Requirements – Examples of Relations (2)

**Requirement**

ABSActivation

ABS shall detect individual wheel acceleration

**VVCase**

ABSActivationTest

Simulate slippage behavior at each wheel individually

**VVCase**

ABSActivationHILTest

Simulate slippage behavior at each wheel individually on Chassis-HIL

**VVCase**

ABSActivationFieldTest

Simulate slippage behavior at each wheel individually on test drive ground

<table>
<thead>
<tr>
<th>VVProcedure</th>
<th>Stimulus</th>
<th>Intended Outcome</th>
<th>Actual Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Normal friction</td>
<td>HIL- Skript1</td>
<td>no activation</td>
<td>no activation</td>
</tr>
<tr>
<td>2: Slippery road left</td>
<td>HIL- Skript2</td>
<td>left wheel activations</td>
<td>left wheel activations</td>
</tr>
<tr>
<td>3. Normal friction</td>
<td>HIL- Skript1</td>
<td>no activation</td>
<td>no activation</td>
</tr>
<tr>
<td>4: Slippery road right</td>
<td>HIL- Skript3</td>
<td>right wheel activations</td>
<td>all wheel activations</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Wrap up

1. Model-Based Development at Carmeq
2. Stochastic Robustness Testing
3. AUTOSAR
4. The Challenge of Variability
5. The EAST-ADL Architecture Description Language
Many Thanks especially to

Mark-Oliver Reiser, TU Berlin
Ralf Immig, Hans-Werner Wiesbrock, IT Power
Helko Glathe, Henning Kleinwechter, Andreas Leicher, Carmeq
Many colleagues from the EAST-ADL project (www.east-adl.org)

Discussion

Your questions?