Verlässliche Echtzeitsysteme – Können wir unseren Autos noch vertrauen?

Bernhard Sechser
Method Park Consulting GmbH, Erlangen
29.04.2013

Who is Method Park?
Why do we need Safety Standards?
Process and Safety demands in Automotive
Hazard Analysis and Risk Assessment
Functional and Technical Development
Software Process in detail
Tool Qualification
Summary

Method Park - Facts and Figures

Facts
- Founded in 2001
- Locations:
  Germany: Erlangen, Munich
  USA: Detroit, Miami

Awards

Revenue & employees

Business unit revenue

- Products
- Training und Consulting
- Engineering

30%
25%
45%
Portfolio

Engineering
Areas:
- Project Coaching
- Software Development & Support
- On Site Support
- Off Site Projects
- Fixed Price Projects

Consulting/Coaching
Topics:
- Software Process Improvement
- CMMI®, SPICE, Automotive SPICE®
- AUTOSAR, Functional Safety
- Requirements Management
- Project and Quality Management
- Software Architecture & Design
- Software Testing

Training
Wide range of seminars in the division systems and software engineering
Accredited by the following organizations:

- SEI
- ISTQB, ISQI, INTACS, IREB, ISAE, ECQA

Our Customers

Automotive
- Audi
- Automotive Lighting
- Blaupunkt
- BMW
- Bosch
- Brose
- Continental
- Daimler
- Delphi
- EAH
- ECE
- IAV
- Johnson Controls
- Knorr-Bremse
- MAN
- Peiker Acoustic
- Pilkington
- Thales
- TRW
- Volkswagen
- Webasto
- ZF
- Zollner

Engineering/Automation
- 7 layers
- ABB
- AEC
- ATR
- Avio
- Zahn Imaging
- Newtec
- Innovations Software
- Technology

Healthcare
- Carl Zeiss
- Fresenius
- Agfa
- Raoul Imaging
- Newtec
- Innovations Software
- Technology

Government/Public
- Bundesagentur für Arbeit
- Kassenärztliche Vereinigung Baden-Württemberg

Defence
- Airbus Deutschland
- Diehl
- EADS
- Elbit
- Orbital
- Raytheon Anschütz
- KID

Further
- Bosch und Siemens Hausgeräte
- Deutsche Post
- GMC Software Technologies
- KBA
- Landesbank Kiel
- Raabe Karcher
- Geisec & Devrient
- Thyssen Rail Signaling

Examples

Ariane 5 (July 4th, 1996)
Detonation shortly after takeoff because of an error in the control software
Root cause: Insufficient tests of a reused "proven in use" software component
Examples

Application that can cause harm (a risk):
- Airbag exploding when infant is sitting in front seat

Need to assess the risk
- Infant getting injured – “not good at all”

Find a mitigation strategy, e.g. a safety function:
- Detecting infant in front seat and disabling airbag
  a) sensor delivers signal to
  b) software/hardware controlling an
  c) actuator (disabler)

Functional Safety is then:
- An infant in front seat is not exposed to an unacceptable (unreasonable) risk

Question: How to measure and agree on the measures?

Examples

Warning

Your Brake Function is temporarily unavailable, please STOP the Car immediately!

OK CANCEL

Question: Do we dare putting software in direct control of people’s life?

Examples

Reasons for Failures

63%

60% Root cause analysis of software failures in 90 healthcare companies
50%
40%
30%
20%
10%
Implementation Architecture Design Requirements Other

Source: Fraunhofer Institute for Experimental Software Engineering 2007

Complexity

DAIMLER

Current Situation
Trends in Automotive Electric/Electronics (E/E)
- Increasing functionality and complexity of software-based car functions
- Increasing risks from systematic faults and random hardware faults
- Most of the new car functions are safety-related

Active Light Functions
Pro-SAFE
Night View Assist
Distronic PLUS
Active Airbags
Adaptive Brake Lights
Remote Boot Closing
Adaptive Body Control
Brake Assist
Hi! Start Assist

Source: © Courtesy of Daimler; Presentation given at Automotive Electronics and Electrical Systems Forum 2008, May 6, 2008, Stuttgart, Germany
§ 823 Abs. 1 BGB:
"Anyone who injures intentionally or negligently the life, body, health, liberty, property or any other right of another person, is obliged to compensate for the resulting damages."

§ 1 Abs. 1 ProdhaftG:
"If someone is killed, his body or health injured or an item damaged by a defect in a product, the manufacturer of the product is obliged to replace the resulting damages."

**Definitions**

**Safety**
... is the absence of unacceptable (unreasonable) risks that can cause harm achieved through a planned strategy

**Functional Safety**
... is part of the overall safety that depends on a system or equipment operating correctly in response to its inputs.  
... is achieved when every specified safety function is carried out and the level of performance required of each safety function is met  
... is not to provide the perfect car, but a safe car.

**Functional Safety Management**
... is the management (plan, do, act, check) of all activities necessary to reach functional safety.

**Existing Standards**

IEC 61508  
Functional safety of electrical / electronic / programmable electronic safety-related systems

- IEC 61511  
Automation
- EN 50261  
Gas Measuring
- IEC 61513  
Medical
- EN 62061  
Nuclear
- ISO 13849  
Automation

- DO 178B  
Aviation
- IEC 62304  
Medical
- EN 50126  
Rail
- EN 50128  
Rail
- EN 50129  
Rail
- IEC 62308  
Medical
- ISO 26262  
Automotive
- ISO 13849  
Manufacturing
- EN 62061  
Manufacturing
- EN 500271  
EN 500402  
EN 500126  
EN 500128  
EN 500129
Scope of ISO 26262

Why not using IEC 61508?

Lessons learnt from application of IEC 61508 in automotive industry:
- Not adapted to real-time and integrated embedded systems
- Not adapted to automotive development and life cycles
- No requirements for manufacturer / supplier relationship
- No 'consumer-goods' orientation
- ...

Companies had to solve these issues themselves until introduction of ISO 26262.
Safety Lifecycle Overview

- Concept
- Development
- Production

Source: ISO 26262-2:2011

Concept Phase
- Focus on entire system
- Risks
- Safety Goals and Requirements
- Safety functions

Product Development
- System, Hardware and Software
- Safety validation and assessment
- Production and Operation (Planning)

Product Development at the System Level

Source: ISO 26262-2:2011
**Product Development**

**Product Development at the Hardware Level**

- System design
- Hardware design
- Verification of hardware adherence
- Hardware integration and testing
- Hardware integration and decommissioning

Source: ISO 26262-5:2011

**Product Development at the Software Level**

- System design
- Software design and architecture
- Software development and integration
- Software testing
- Tool phase validation

Source: ISO 26262-6:2011

---

**Safety Lifecycle Overview**

- Production
- Installation
- Operation
- Maintenance and reparation
- Disassembly

Source: ISO 26262-2:2011

---

**Contents**

- Who is Method Park?
- Why do we need Safety Standards?
- Process and Safety demands in Automotive
- **Hazard Analysis and Risk Assessment**
- Functional and Technical Development
- Software Process in detail
- Tool Qualification
- Summary
Hazard Analysis and Risk Assessment

Risk reduction to an acceptable level

Source: IEC 61508-5:2010

Situation analysis and hazard identification

- List of driving and operating situations
  - Estimation of the probability of Exposure
- Detailing failure modes leading to hazards in specific situations
  - Estimation of Controllability
- Evaluating consequences of the hazards
  - Estimation of potential Severity

- Respect only the plain item (do not take risk-reducing measures into account!)
- Involve persons with good knowledge and domain experience

Severity

Measure of the extent of harm to an individual in a specific situation

<table>
<thead>
<tr>
<th>Class</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>No injuries</td>
<td>Light and moderate injuries</td>
<td>Severe and life-threatening injuries (survival probable)</td>
<td>Life-threatening injuries (survival uncertain), fatal injuries</td>
</tr>
</tbody>
</table>

Source: ISO 26262-3:2011
Exposure
State of being in an operational situation that can be hazardous if coincident with the failure mode under analysis

<table>
<thead>
<tr>
<th>Class</th>
<th>E0</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Incre-dible</td>
<td>Very low probability</td>
<td>Low probability</td>
<td>Medium probability</td>
<td>High probability</td>
</tr>
<tr>
<td>Time</td>
<td>Not specified</td>
<td>Less than 1% of average operating time</td>
<td>1% - 10% of average operating time</td>
<td>&gt; 10% of average operating time</td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td>Situations that occur less than once a year for the great majority of drivers</td>
<td>Situations that occur once or more often for an average driver</td>
<td>All situations that occur during almost every drive on average</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: ISO 26262-3:2011

Controllability
Avoidance of the specified harm or damage through the timely reactions of the persons involved

<table>
<thead>
<tr>
<th>Class</th>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Controllable in general</td>
<td>Simply controllable</td>
<td>Normally controllable</td>
<td>Difficult to control or uncontrollable</td>
</tr>
<tr>
<td>Definition</td>
<td>99% or more of all drivers or other traffic participants are usually able to avoid a specific harm.</td>
<td>90% or more of all drivers or other traffic participants are usually able to avoid a specific harm.</td>
<td>Less than 90% of all drivers or other traffic participants are usually able, or barely able, to avoid a specific harm.</td>
<td></td>
</tr>
</tbody>
</table>

Source: ISO 26262-3:2011

Safety Goals
- top-level safety requirements as a result of the hazard analysis and risk assessment
- assigned to each identified hazard rated with an ASIL A-D
- lead to item characteristics needed to avert hazards or to reduce risks associated with the hazards to an acceptable level
- are assigned to a safe state that must be reached in case of appearance
- indicate the maximum fault tolerance time within the safe state must be reached

fault tolerance time = fault recognition time + fault reaction time

Combinations of Severity, Exposure and Controllability result in the applicable ASIL.

The ASIL's influence the development process of the items.

QM = Quality Management
No specific ISO 26262 requirement has to be observed

If S0 or E0 or C0 is set, no ASIL is required (QM).
Hazard Analysis and Risk Assessment

Example for Safety Goals: Park Brake System

<table>
<thead>
<tr>
<th>ID</th>
<th>Safety Goal</th>
<th>ASIL</th>
<th>Safe State</th>
<th>FTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>Avoidance of unintended maximum brake force build up at one or several wheels during drive and in all environmental conditions</td>
<td>D</td>
<td>Brake released</td>
<td>50 ms</td>
</tr>
<tr>
<td>G2</td>
<td>Guarantee the specified parking brake function in use case situation &quot;parking on slope&quot; in all environmental conditions</td>
<td>A</td>
<td>Brake closed</td>
<td>500 ms</td>
</tr>
<tr>
<td>G3</td>
<td>Avoidance of unintended release of the parking brake in use case situation &quot;parking on slope&quot; in all environmental conditions</td>
<td>C</td>
<td>Brake closed</td>
<td>500 ms</td>
</tr>
</tbody>
</table>

Functional Safety Concept

Safety Goals and Functional Safety Requirements

ASIL Decomposition

Source: ISO 26262-9:2011

Contents

- Who is Method Park?
- Why do we need Safety Standards?
- Process and Safety demands in Automotive
- Hazard Analysis and Risk Assessment
- Functional and Technical Development
- Software Process in detail
- Tool Qualification
- Summary
Architectures

Example: Three channel structure 2o03

```
Sensor
Input Circuit | Central Processing Unit | Output Circuit 1 | Output Circuit 2
Sensor
Input Circuit | Central Processing Unit | Output Circuit 1 | Output Circuit 2
Sensor
Input Circuit | Central Processing Unit | Output Circuit 1 | Output Circuit 2
```

Actuator

Contents

- Who is Method Park?
- Why do we need Safety Standards?
- Process and Safety demands in Automotive
- Hazard Analysis and Risk Assessment
- Functional and Technical Development
- Software Process in detail
- Tool Qualification
- Summary

Product Development at Hardware & Software Level

Important part: Hardware-Software Interface Specification (HSI)

Source: ISO 26262-4:2011

How to understand the standard tables

For each method, the degree of recommendation to use corresponding methods depends on the ASIL and is categorized as follows:

- "++" The method is highly recommended for this ASIL
- "+" The method is recommended for this ASIL
- "o" The method has no recommendation for or against its usage for this ASIL
Initiation of Product Development at the Software Level

Topics to be covered by modeling and coding guidelines

<table>
<thead>
<tr>
<th>Topics</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Enforcement of low complexity</td>
<td>++ ++ ++ ++</td>
</tr>
<tr>
<td>1b Use of language subsets</td>
<td>++ ++ ++ ++</td>
</tr>
<tr>
<td>1c Enforcement of strong typing</td>
<td>++ ++ ++ ++</td>
</tr>
<tr>
<td>1d Use of defensive implementation techniques</td>
<td>0 + ++ ++</td>
</tr>
<tr>
<td>1e Use of established design principles</td>
<td>+ + + ++</td>
</tr>
<tr>
<td>1f Use of unambiguous graphical representation</td>
<td>+ ++ ++ ++</td>
</tr>
<tr>
<td>1g Use of style guides</td>
<td>+ ++ ++ ++</td>
</tr>
<tr>
<td>1h Use of naming conventions</td>
<td>++ ++ ++ ++</td>
</tr>
</tbody>
</table>

Source: ISO 26262-6:2011

Specification of Software Safety Requirements

Goals

- Derive Software Safety Requirements from and ensure consistency with
  - System Design
  - Technical Safety Concept
- Detail the hardware-software interface requirements

Methods for specifying Safety Requirements

- Safety requirements shall be specified by an appropriate combination of natural language and methods listed in the table
- For higher level safety requirements (e.g. functional and technical safety requirements) natural language is more appropriate while for lower level safety requirements (e.g. software and hardware safety requirements) notations listed in the table are more appropriate

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Informal notations for requirements specification</td>
<td>++ ++ ++ ++</td>
</tr>
<tr>
<td>1b Semi-formal notations for requirements specification</td>
<td>+ + ++ ++</td>
</tr>
<tr>
<td>1c Formal notations for requirements specification</td>
<td>+ + ++ ++</td>
</tr>
</tbody>
</table>

Source: ISO 26262-8:2011

Methods for the verification of Safety Requirements

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Verification by walk-through</td>
<td>++ + 0 0</td>
</tr>
<tr>
<td>1b Verification by inspection</td>
<td>+ ++ ++ ++</td>
</tr>
<tr>
<td>1c Semi-formal verification (e.g. executable models)</td>
<td>+ + ++ ++</td>
</tr>
<tr>
<td>1d Formal verification</td>
<td>0 + ++ +</td>
</tr>
</tbody>
</table>

Source: ISO 26262-8:2011
Software Architectural Design

Goals

- Develop an Architecture that implements the Software Safety Requirements
  - Static and dynamic interfaces
  - Safety-related and non safety related requirements
- Verify the Software Architecture
  - Compliance with the requirements
  - Compatibility with hardware
  - Respect of design principles and standards

Source: ISO 26262-6:2011

Software Architectural Design

Principles for software architectural design

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>Hierarchical structure of software components&lt;br&gt;++ ++ ++ ++</td>
</tr>
<tr>
<td>1b</td>
<td>Restricted size of software components&lt;br&gt;++ ++ ++ ++</td>
</tr>
<tr>
<td>1c</td>
<td>Restricted size of interfaces&lt;br&gt;+ + + +</td>
</tr>
<tr>
<td>1d</td>
<td>High cohesion within each software component&lt;br&gt;++ ++ ++ ++</td>
</tr>
<tr>
<td>1e</td>
<td>Restricted coupling between software components&lt;br&gt;+ ++ ++ ++</td>
</tr>
<tr>
<td>1f</td>
<td>Appropriate scheduling properties&lt;br&gt;++ ++ ++ ++</td>
</tr>
<tr>
<td>1g</td>
<td>Restricted use of interrupts&lt;br&gt;+ + + +</td>
</tr>
</tbody>
</table>

Source: ISO 26262-6:2011

Software Architectural Design

Methods for the verification of the software architectural design

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>Walk-through of the design&lt;br&gt;++ + o o</td>
</tr>
<tr>
<td>1b</td>
<td>Inspection of the design&lt;br&gt;+ ++ ++ ++</td>
</tr>
<tr>
<td>1c</td>
<td>Simulation of dynamic parts of the design&lt;br&gt;+ + + +</td>
</tr>
<tr>
<td>1d</td>
<td>Prototype generation&lt;br&gt;o o + +</td>
</tr>
<tr>
<td>1e</td>
<td>Formal verification&lt;br&gt;o o + +</td>
</tr>
<tr>
<td>1f</td>
<td>Control flow analysis&lt;br&gt;+ + ++ ++</td>
</tr>
<tr>
<td>1g</td>
<td>Data flow analysis&lt;br&gt;+ + ++ ++</td>
</tr>
</tbody>
</table>

Error handling

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range checks of input and output data&lt;br&gt;++ ++ ++ ++</td>
<td></td>
</tr>
<tr>
<td>Plausibility check&lt;br&gt;+ + + ++</td>
<td></td>
</tr>
<tr>
<td>Detection of data errors&lt;br&gt;+ + + +</td>
<td></td>
</tr>
<tr>
<td>External monitoring facility&lt;br&gt;0 + + ++</td>
<td></td>
</tr>
<tr>
<td>Control flow monitoring&lt;br&gt;o + ++ ++</td>
<td></td>
</tr>
<tr>
<td>Diverse software design&lt;br&gt;0 0 + ++</td>
<td></td>
</tr>
</tbody>
</table>

Error detection

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static recovery mechanism&lt;br&gt;+ + + +</td>
<td></td>
</tr>
<tr>
<td>Graceful degradation&lt;br&gt;+ + ++ ++</td>
<td></td>
</tr>
<tr>
<td>Independent parallel redundancy&lt;br&gt;0 0 + ++</td>
<td></td>
</tr>
<tr>
<td>Correcting codes for data&lt;br&gt;+ + + +</td>
<td></td>
</tr>
</tbody>
</table>

Source: ISO 26262-6:2011
Software Unit Design and Implementation

Goals

- Specify SW Units based on:
  - SW Architecture
  - SW Safety Requirements
- Implement the SW Units
- Verify SW Units
  - Code reviews / inspections

Source: ISO 26262-6:2011

Design principles for software unit design and implementation

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a</td>
<td>++</td>
</tr>
<tr>
<td>1b</td>
<td>+</td>
</tr>
<tr>
<td>1c</td>
<td>++</td>
</tr>
<tr>
<td>1d</td>
<td>++</td>
</tr>
<tr>
<td>1e</td>
<td>++</td>
</tr>
<tr>
<td>1f</td>
<td>O</td>
</tr>
<tr>
<td>1g</td>
<td>+</td>
</tr>
<tr>
<td>1h</td>
<td>+</td>
</tr>
<tr>
<td>1i</td>
<td>++</td>
</tr>
<tr>
<td>1j</td>
<td>+</td>
</tr>
</tbody>
</table>

Source: ISO 26262-6:2011

Example: MISRA C

- Programming standard developed by Motor Industry Software Reliability Association
- Avoidance of runtime errors due to unsafe C constructs
- The respect of MISRA C shall be demonstrated → static code analysis

Infos: www.misra.org

Software Unit Testing

Goals

- Demonstrate that the software units fulfil the Software Unit Specifications
- Verify absence of undesired functionalities

Source: ISO 26262-6:2011
Software Unit Testing

The software unit testing methods shall be applied to demonstrate that the software units achieve:

- Compliance with the software unit design specification
- Compliance with the specification of the hardware-software interface
- Correct implementation of the functionality
- Absence of unintended functionality
- Robustness
- Sufficiency of the resources to support the functionality

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements-based test</td>
<td>++</td>
</tr>
<tr>
<td>Interface test</td>
<td>++</td>
</tr>
<tr>
<td>Fault injection test</td>
<td>+</td>
</tr>
<tr>
<td>Resource usage test</td>
<td>+</td>
</tr>
<tr>
<td>Back-to-back comparison test</td>
<td>+</td>
</tr>
</tbody>
</table>

Source: ISO 26262-6:2011

Software Unit Testing

Methods for deriving test cases for software unit testing

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis of requirements</td>
<td>++</td>
</tr>
<tr>
<td>Generation and analysis of equivalence classes</td>
<td>++</td>
</tr>
<tr>
<td>Analysis of boundary values</td>
<td>+</td>
</tr>
<tr>
<td>Error guessing</td>
<td>+</td>
</tr>
</tbody>
</table>

Source: ISO 26262-6:2011

Software Integration and Testing

Goals

- Integrate SW components
- Integration sequence
- Testing of interfaces between components/units
- Verify correct implementation of the SW Architecture

Source: ISO 26262-6:2011

Structural coverage metrics at the software unit level

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement coverage</td>
<td>++</td>
</tr>
<tr>
<td>Branch coverage</td>
<td>+</td>
</tr>
<tr>
<td>MC/DC (Modified Condition/Decision Coverage)</td>
<td>+</td>
</tr>
</tbody>
</table>

Source: ISO 26262-6:2011
The software integration test methods shall be applied to demonstrate that both the software components and the embedded software achieve:

- Compliance with the software architectural design
- Compliance with the specification of the hardware-software interface
- Correct implementation of the functionality
- Robustness and sufficiency of the resources to support the functionality

### Structural coverage metrics at the software architectural level

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a</td>
<td>++</td>
</tr>
<tr>
<td>1b</td>
<td>++</td>
</tr>
<tr>
<td>1c</td>
<td>++</td>
</tr>
<tr>
<td>1d</td>
<td>+</td>
</tr>
<tr>
<td>1e</td>
<td>+</td>
</tr>
</tbody>
</table>

**Source:** ISO 26262-6:2011

### Verification of Software Safety Requirements

**Goals**

- Verify that the embedded software fulfils the Software Safety Requirements in the target environment

**Methods**

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a</td>
<td>++</td>
</tr>
<tr>
<td>1b</td>
<td>++</td>
</tr>
<tr>
<td>1c</td>
<td>++</td>
</tr>
</tbody>
</table>

**Source:** ISO 26262-6:2011
Functional Safety Assessment

What shall be provided to support the Safety Case?

Verify Component
Design Architecture
Design Function
Product Definition
Create Hardware & Software

Safety Case - Arguments

Qualification of Software Tools

To determine the required level of confidence in a software tool, perform a use case analysis:

- Evaluate if a malfunctioning software tool and its erroneous output can lead to the violation of any safety requirement allocated to the safety-related item or element to be developed
- Establish probability of preventing or detecting such errors in its output
  - Considers measures internal to the software tool (e.g. monitoring)
  - Measures external to the software tool implemented in the development process for the safety-related item or element (e.g. guidelines, tests, reviews)

Qualification of Software Tools

Tool Impact (TI)
Possibility that a safety requirement, allocated to the safety-related item or element, is violated if the software tool is malfunctioning or producing erroneous output
TI1 – no such possibility
TI2 – all other cases

Tool error Detection (TD)
Probability of preventing or detecting that the software tool is malfunctioning or producing erroneous output
TD1 – high degree of confidence for prevention or detection
TD2 – medium degree of confidence for prevention or detection
TD3 – all other cases
Qualification of Software Tools

**Tool Confidence Level (TCL)**
Based on the values determined for the classes of TI and TD

<table>
<thead>
<tr>
<th></th>
<th>TD1</th>
<th>TD2</th>
<th>TD3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI1</td>
<td>TCL1</td>
<td>TCL1</td>
<td>TCL1</td>
</tr>
<tr>
<td>TI2</td>
<td>TCL1</td>
<td>TCL2</td>
<td>TCL3</td>
</tr>
</tbody>
</table>

Source: ISO 26262-8:2011

Qualification methods:

<table>
<thead>
<tr>
<th>Qualification methods of software tools classified TCL3</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Increased confidence from use</td>
<td>++</td>
</tr>
<tr>
<td>1b Evaluation of the tool development process</td>
<td>++</td>
</tr>
<tr>
<td>1c Validation of the software tool</td>
<td>+</td>
</tr>
<tr>
<td>1d Development in accordance with a safety standard</td>
<td>+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Qualification methods of software tools classified TCL2</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Increased confidence from use</td>
<td>++</td>
</tr>
<tr>
<td>1b Evaluation of the tool development process</td>
<td>++</td>
</tr>
<tr>
<td>1c Validation of the software tool</td>
<td>+</td>
</tr>
<tr>
<td>1d Development in accordance with a safety standard</td>
<td>+</td>
</tr>
</tbody>
</table>

Source: ISO 26262-8:2011

Contents

- Who is Method Park?
- Why do we need Safety Standards?
- Process and Safety demands in Automotive
- Hazard Analysis and Risk Assessment
- Functional and Technical Development
- Software Process in detail
- Tool Qualification
- Summary

Summary

- Today’s electronic systems are too complex to understand all potential hazards
- An approach for Functional Safety is needed to avoid severe injuries and damages in human lives and property
- A standardized way to show that your product is safe is needed – best practice yet not fully established – guidance needed
Thank you!

Bernhard Sechser
Principal Consultant SPICE & Safety
Method Park Consulting GmbH
Wetterkreuz 19a
91058 Erlangen
Germany

Phone: +49 9131 97206-427
Mobile: +49 173 3882055
Bernhard.Sechser@methodpark.com
http://www.xing.com/profile/Bernhard_Sechser
http://www.methodpark.com