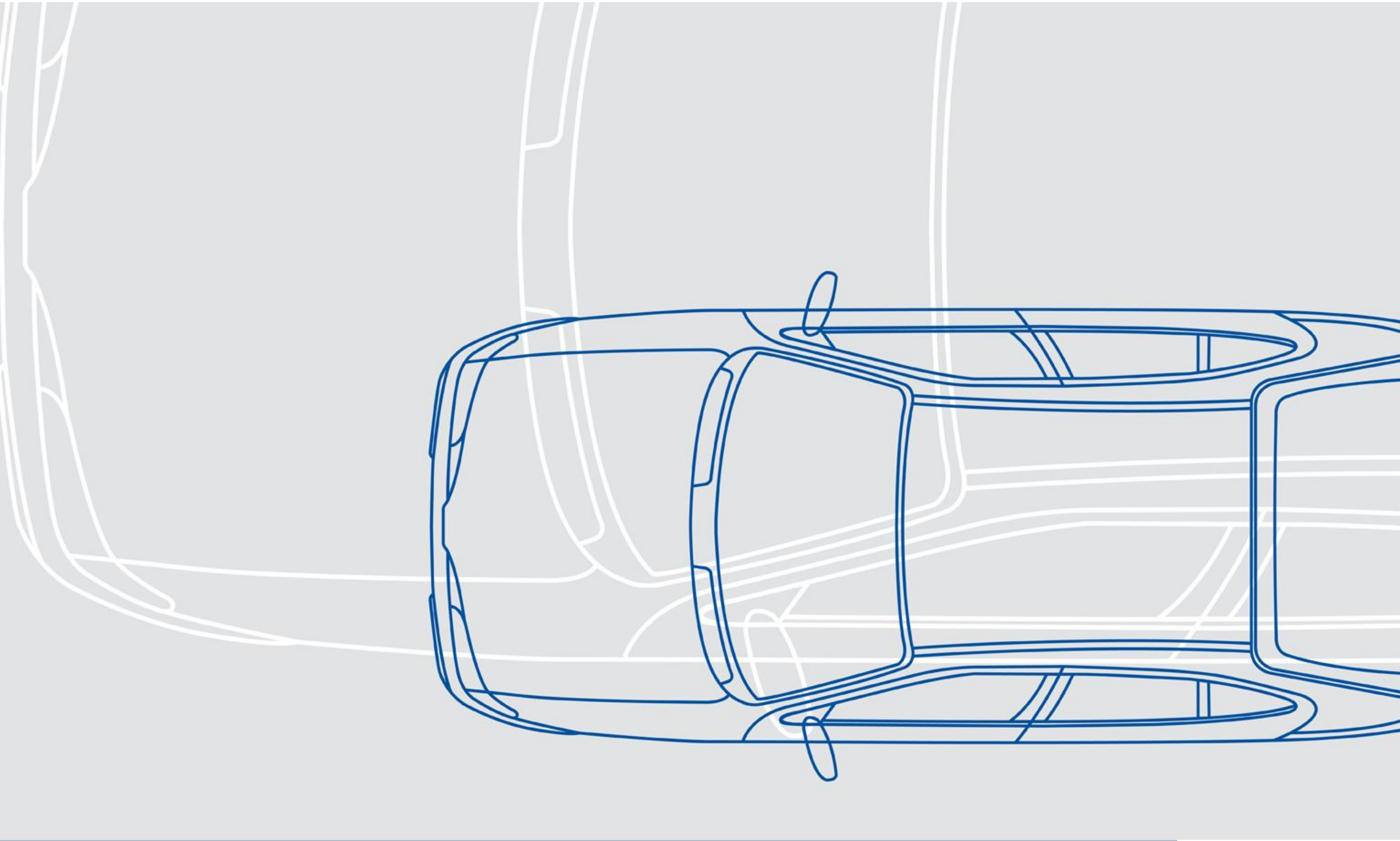
Entwicklung zuverlässiger Software-Systeme, Stuttgart 30.Juni 2011

Tools and Methods for Validation and Verification as requested by ISO26262

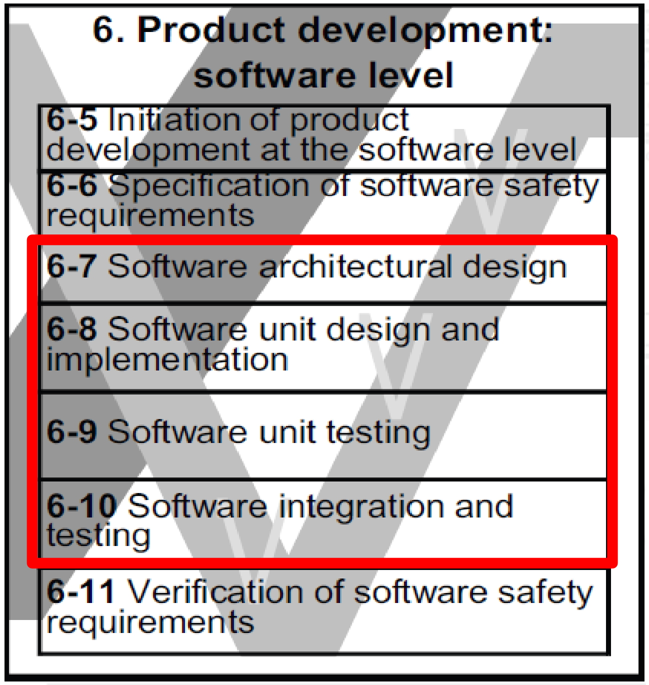


Tools and Methods for Validation and Verification
Introduction ISO26262

"ISO 26262 is the adaptation of IEC 61508 to comply with needs specific to the application sector of E/E systems within road vehicles.
This adaptation applies to all activities during the safety lifecycle of safety-related **systems** comprised of electrical, electronic, and software elements that provide safety-related functions."

- Goal:
  - Functional safety of electrical/electronic/programmable electronic safety-related systems
- Provides
  - measures for an automotive safety lifecycle (management, development, production, operation, service, decommissioning)
  - an automotive specific risk-based approach for determining risk classes (Automotive Safety Integrity Levels, ASILs);
  - requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety being achieved.

Tools and Methods for Validation and Verification
Software development process in ISO26262



6. Product development: software level

6-5 Initiation of product development at the software level

6-6 Specification of software safety requirements

6-7 Software architectural design

6-8 Software unit design and implementation

6-9 Software unit testing

6-10 Software integration and testing

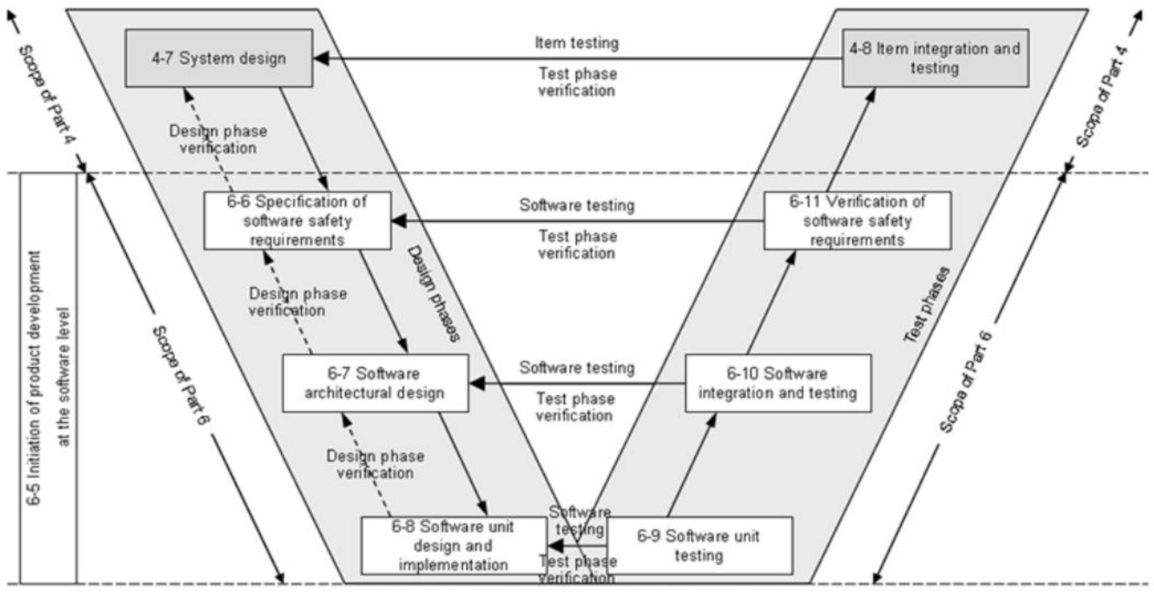
6-11 Verification of software safety requirements

Depending on the ASIL, ISO26262

- gives principles for design
- suggests mechanisms for error detection
- demands methods for the verification
- suggests methods for testing and deriving test cases& metrics cases

on both software unit and architectural level

ISO26262 follows a V-cycle development approach



4-7 System design
Item testing
Test phase verification
4-8 Item integration and testing
Scope of Part 4
Scope of Part 4
Design phase verification
6-6 Specification of software safety requirements
Software testing
Test phase verification
6-11 Verification of software safety requirements
Design phases
Test phases
6-5 Initiation of product development at the software level
Scope of Part 6
Scope of Part 6
Design phase verification
6-7 Software architectural design
Software testing
Test phase verification
6-10 Software integration and testing
Design phase verification
6-8 Software unit design and implementation
Software testing
Test phase verification
6-9 Software unit testing

Tools and Methods for Validation and Verification
ISO26262 Software Unit & Integration Test methods

Methods for software testing at architectural (integration) and unit level
(according to ISO26262 part 6, chapter 9.4.3 and 10.4.3)

- Requirements-based test (=validation)
- Interface test
- Fault injection test
  → *".. Includes injection of arbitrary faults in order to test safety mechanisms (e.g. by corrupting software or hardware components"*
- Resource usage test
  → *".. average/maximum processor performance, min/max execution times, storage usage (stack, program, data), bandwidth of communication links.."*
- Back-to-back comparison test between model and code (if applicable)
  → *".. model that can simulate the functionality of the software components. Here, the model and code are stimulated the same way and results compared with each other"*

ISO26262 Software Unit & Integration Test Requirements

Test Requirements for software testing at architectural und unit level

„The test environment for software integration testing shall correspond as closely as possible to the target environment. <u>If the software unit testing is not carried out in the target environment</u>, the differences in the source and object code, and the differences between the test environment and the target environment, shall be analysed in order to specify additional tests in the target environment during the subsequent test phases." (part 6, chapter 9.4.6)

Test Environment:

- *„The testing of the implementation of the software safety requirements shall be executed on the target hardware." (part 6, chapter 11.4.2)*
- Hardware-in-the-loop, electronic control unit network environments, vehicles (table16)

Conclusion:

→ To optimize the test and minimize the effort, as many of the target specific potential problems should be indentified in early tests.

Tools and Methods for Validation and Verification
MIL, SIL, PIL, HIL

MIL, SIL, PIL and HIL tests are suggested for both software unit test and integration test (part 6, section 9.4.6 and 10.4.8)

- MIL (model in the loop)
  - algorithm in a simulation environment, floating point,
  - usually open loop stimulation (test vectors)
- SIL (software in the loop)
  - algorithm in a programming language, fix point,
  - real time or non real time environment
  - usually open loop stimulation (test vectors)
- PIL (processor in the loop)
  - algorithm executed on target microcontroller
  - real time environment
  - usually open loop stimulation (test vectors)
- HIL (hardware in the loop)
  - algorithm executed on a real time target (µC or RP target)
  - real time environment
  - closed loop stimulation

Tests for

- Requirements & concepts (validation)
- logic

- Code verification (syntax, initialisation, …)
- Arithmetics

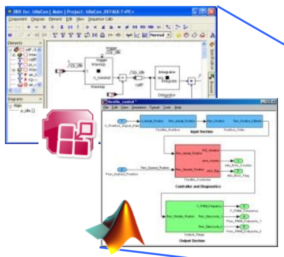
- Resource usage
- (Real?) time behaviour
- (real time) OS
- performance

- Integration test
- Full closed loop
- HW interfaces

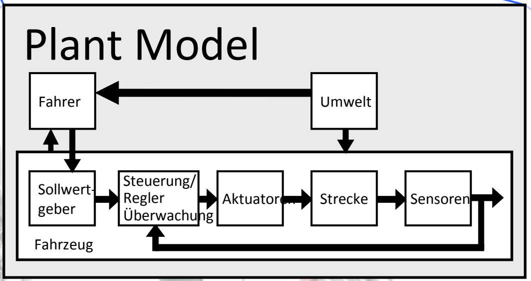
Tools and Methods for Validation and Verification
Virtual Prototyping

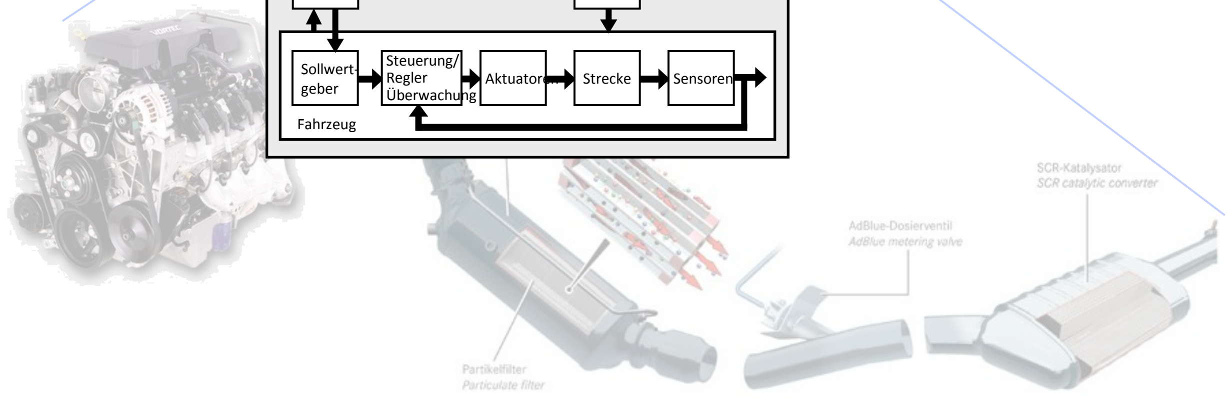
Virtual Prototyping – Model-in-the-Loop (MiL) (often called simulation)

Function models



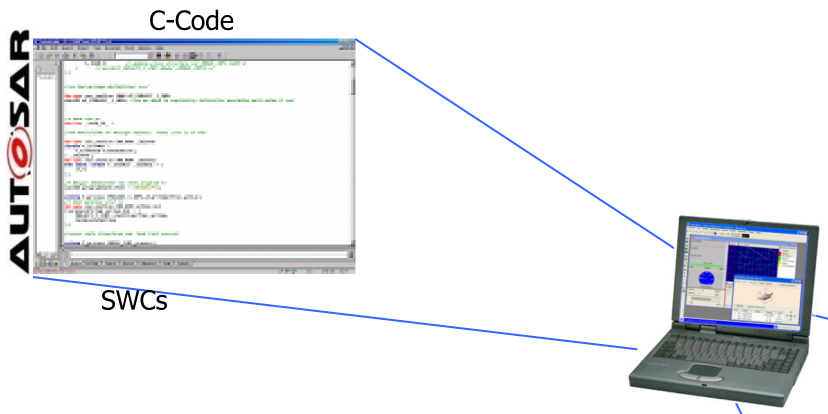
- Floating point, non-optimized, …
- Non-real time
- Plant model might be replaced by test vectors → unit test → open loop
- Several function models → plant model needed → integration test → closed loop



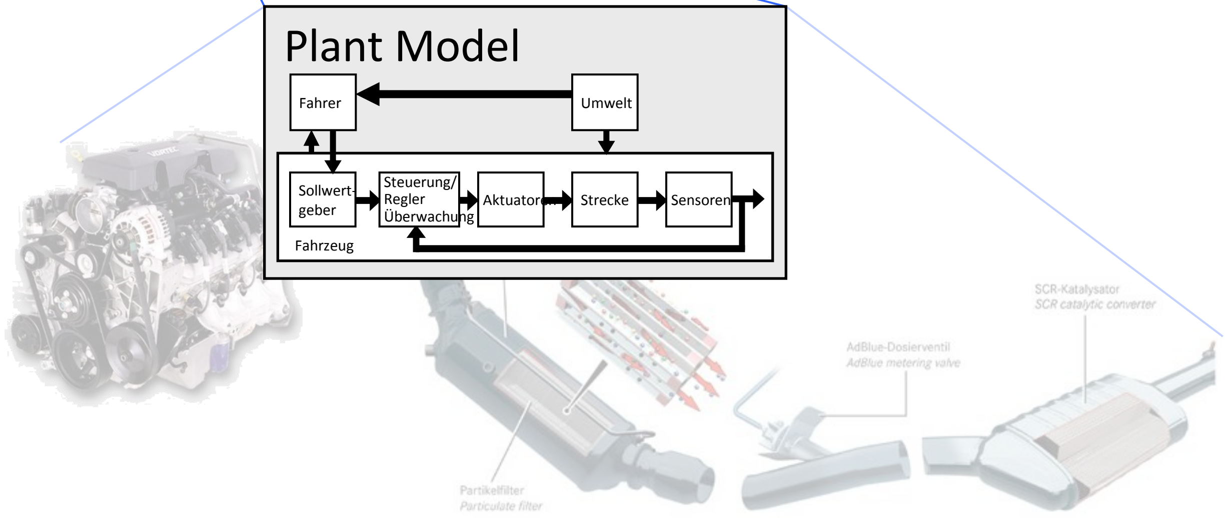


Tools and Methods for Validation and Verification
Virtual Prototyping

Virtual Prototyping – Software-in-the-Loop (SiL)



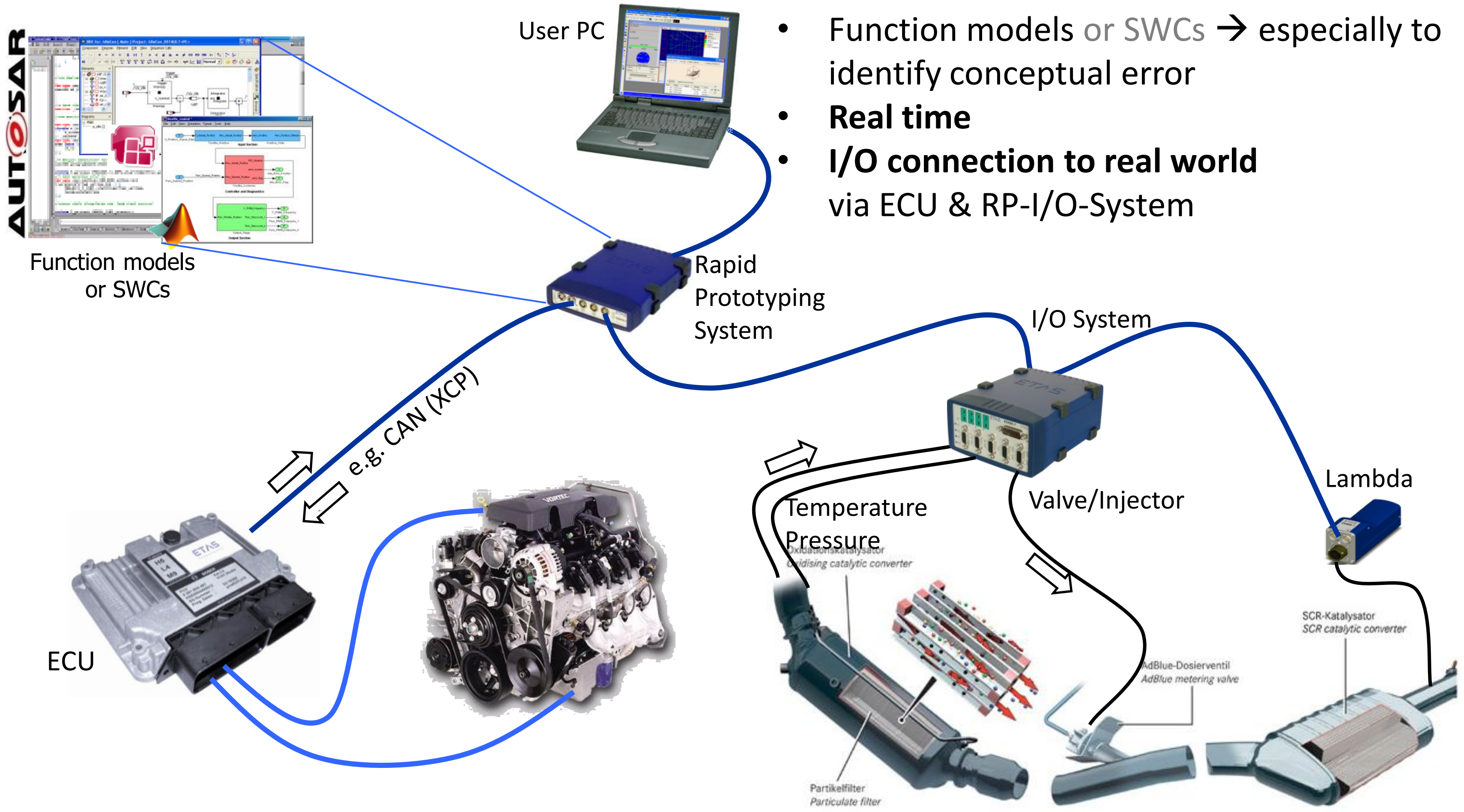
- **ECU ready C-Code**!
- Fix point, optimized, multi-raster, …
- OS/basic SW accesses, …
- Non-real time
- Plant model might be replaced by test vectors → unit test → open loop
- Several function models → plant model needed → integration test → closed loop



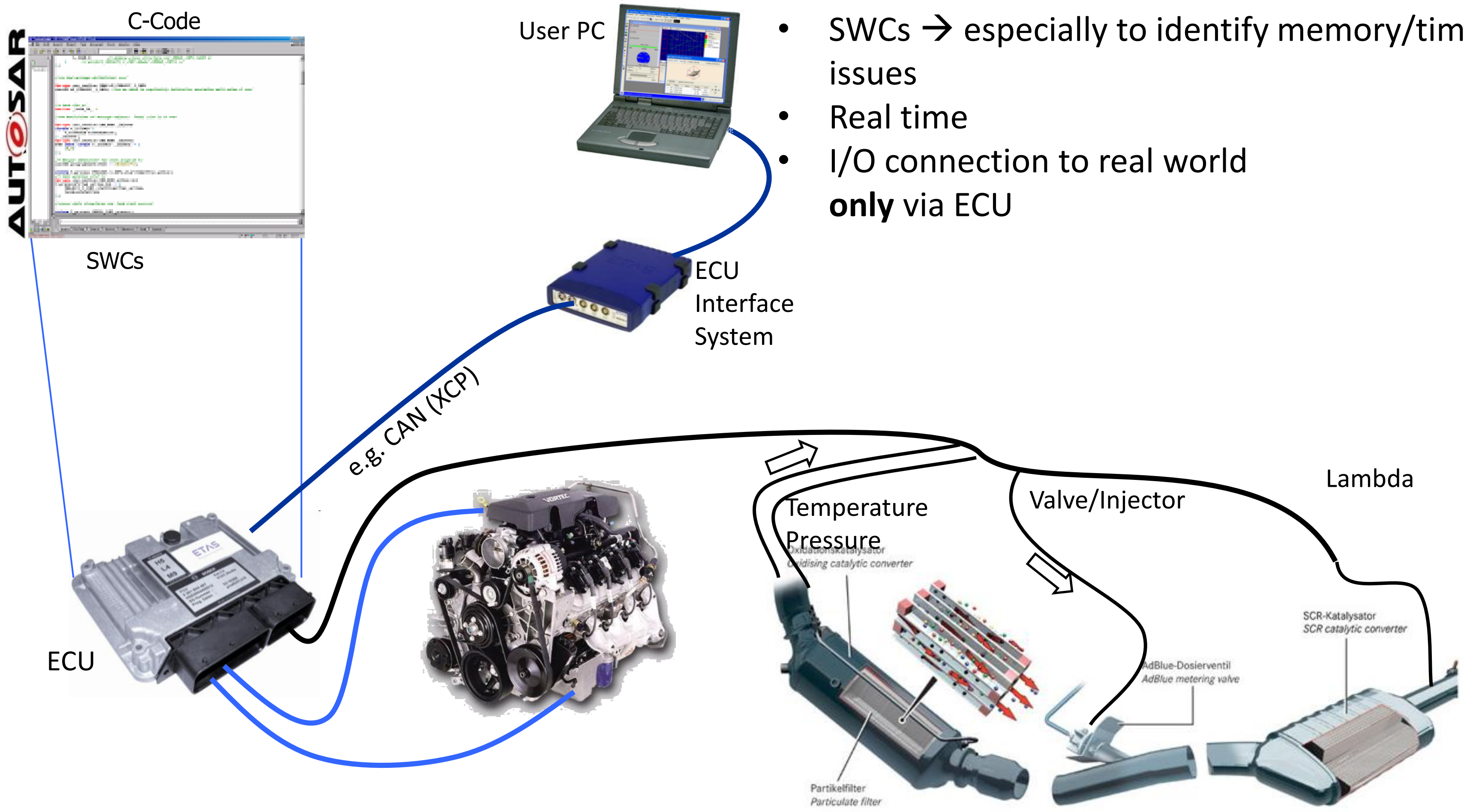
Tools and Methods for Validation and Verification
Rapid Prototyping

Rapid Prototyping – "external" Bypass System (e.g. extension of an existing system)

- Function models or SWCs → especially to identify conceptual error
- **Real time**
- **I/O connection to real world** via ECU & RP-I/O-System



Target Prototyping – “internal” Bypass System

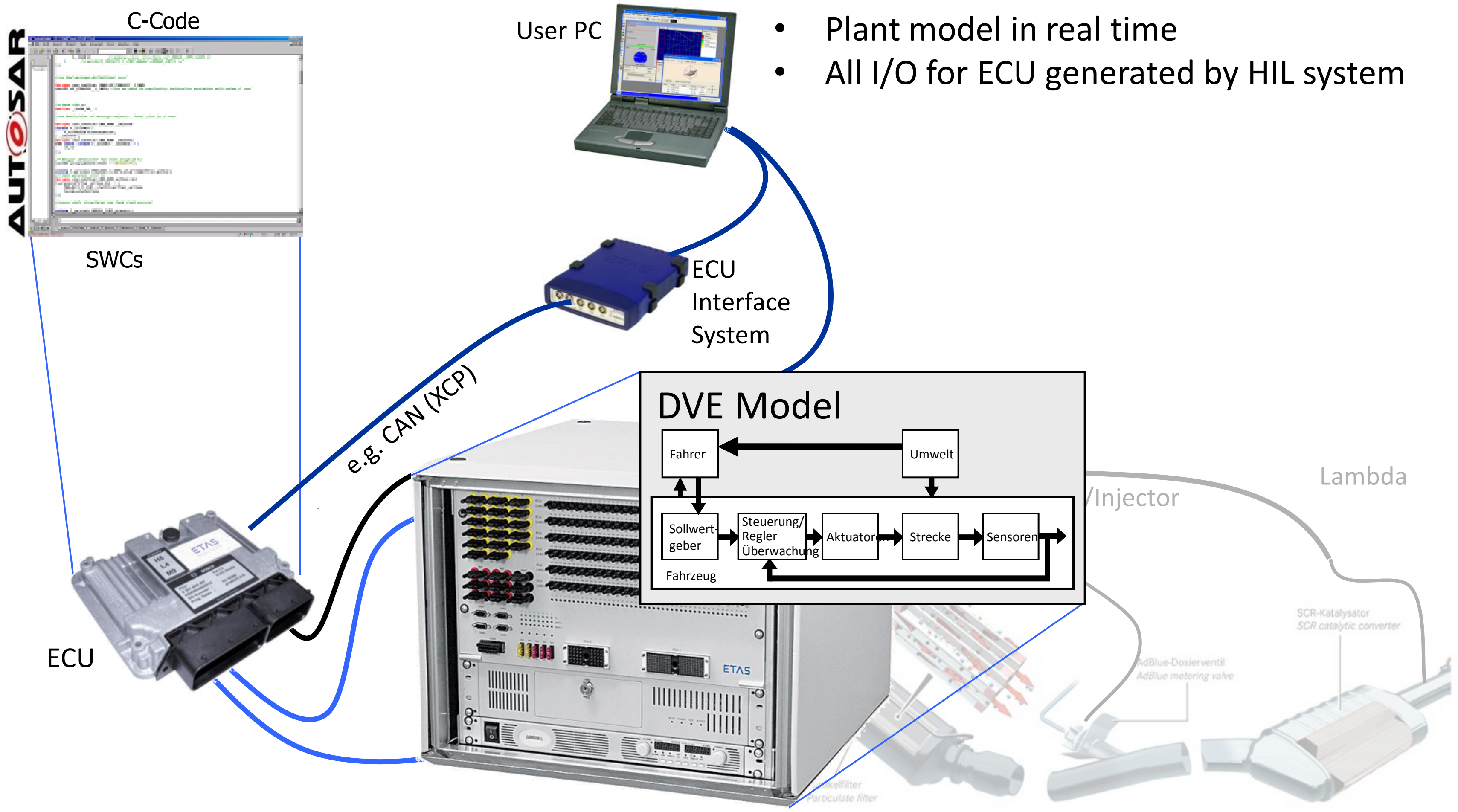


- SWCs → especially to identify memory/timing issues
- Real time
- I/O connection to real world **only** via ECU

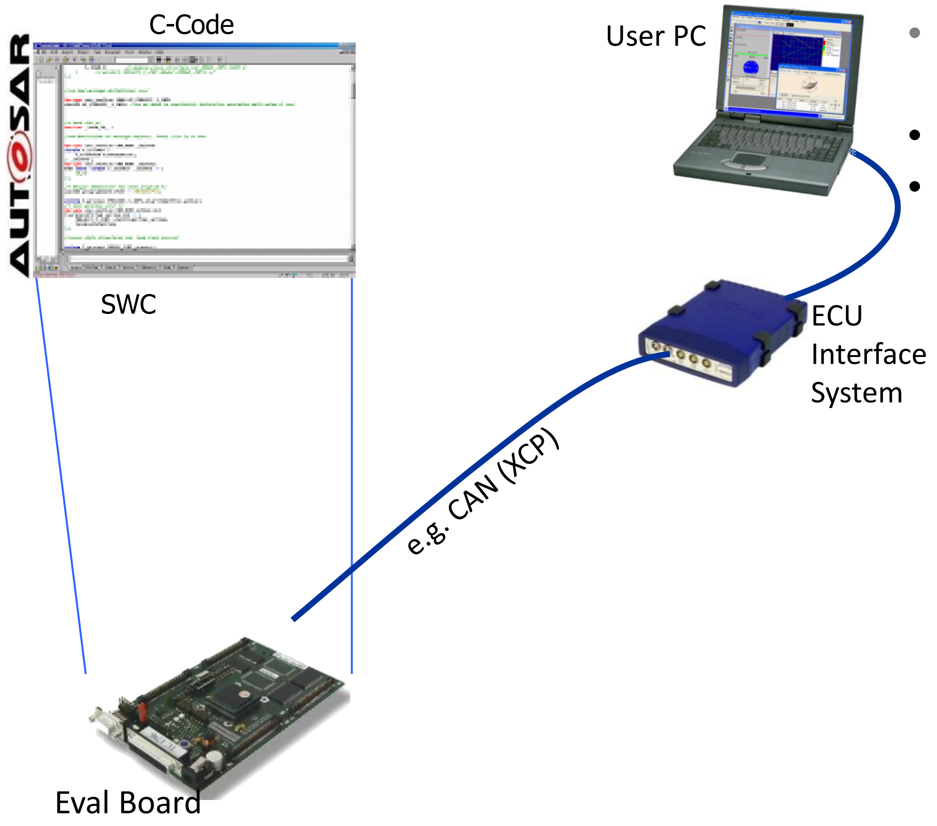
Tools and Methods for Validation and Verification
Hardware-in-the-Loop

Hardware-in-the-Loop – real time replacement for the real world

- Plant model in real time
- All I/O for ECU generated by HIL system



Processor-in-the-Loop – real time with real I/O



- One SWC → especially to identify memory/timing/µC specific issues
- Real time
- Mainly unit tests as only one SWC used → real time test vectors needed

Tools and Methods for Validation and Verification
MIL, SIL, PIL, HIL for ISO26262 requirements

VP

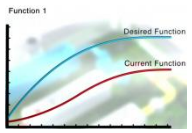
RP

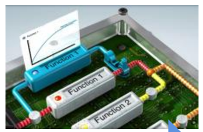
OTP

RP

OTP

	MIL	SIL	PIL	HIL
Requirements-based test	■	■	■	■
Interface test		■	■	■
Fault injection test		■	■	■
Resource usage test			■	■
Back-to-back comparison				■





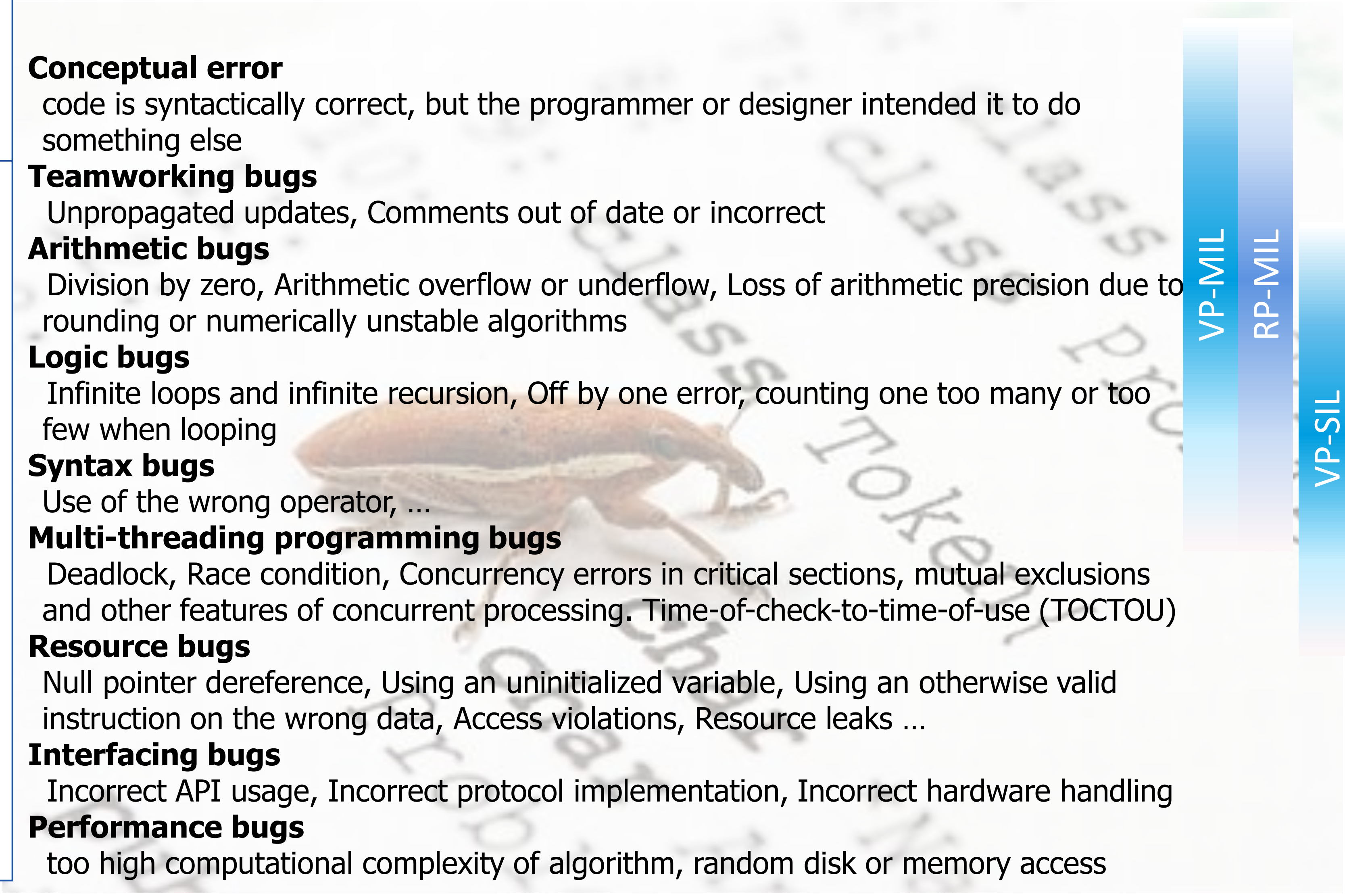
algorithm

implementation

Tools and Methods for Validation and Verification
Software bug types – what could possibly go wrong?

Validation

Verification



Conceptual error
code is syntactically correct, but the programmer or designer intended it to do something else

Teamworking bugs
Unpropagated updates, Comments out of date or incorrect

Arithmetic bugs
Division by zero, Arithmetic overflow or underflow, Loss of arithmetic precision due to rounding or numerically unstable algorithms

Logic bugs
Infinite loops and infinite recursion, Off by one error, counting one too many or too few when looping

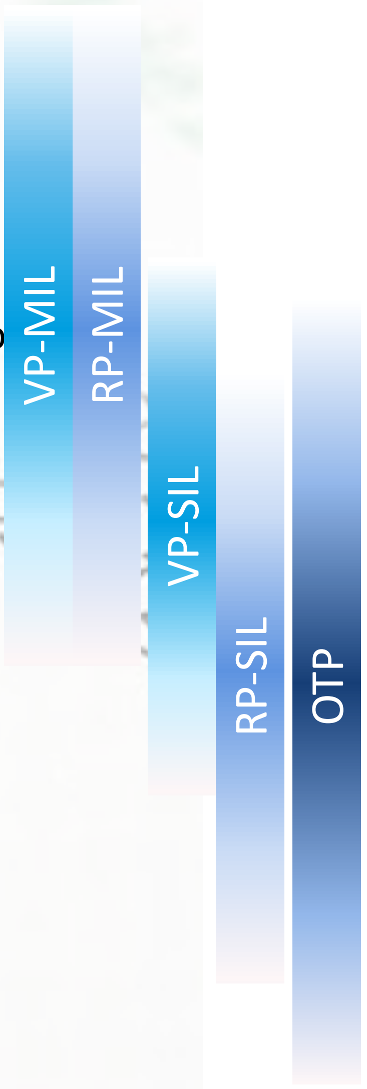
Syntax bugs
Use of the wrong operator, …

Multi-threading programming bugs
Deadlock, Race condition, Concurrency errors in critical sections, mutual exclusions and other features of concurrent processing. Time-of-check-to-time-of-use (TOCTOU)

Resource bugs
Null pointer dereference, Using an uninitialized variable, Using an otherwise valid instruction on the wrong data, Access violations, Resource leaks …

Interfacing bugs
Incorrect API usage, Incorrect protocol implementation, Incorrect hardware handling

Performance bugs
too high computational complexity of algorithm, random disk or memory access



sources: http://en.wikipedia.org/wiki/Software_bug#Common_types_of_computer_bugs; http://www.articlesnatch.com/Article/Software-Bug-And-Their-Common-Types/594429

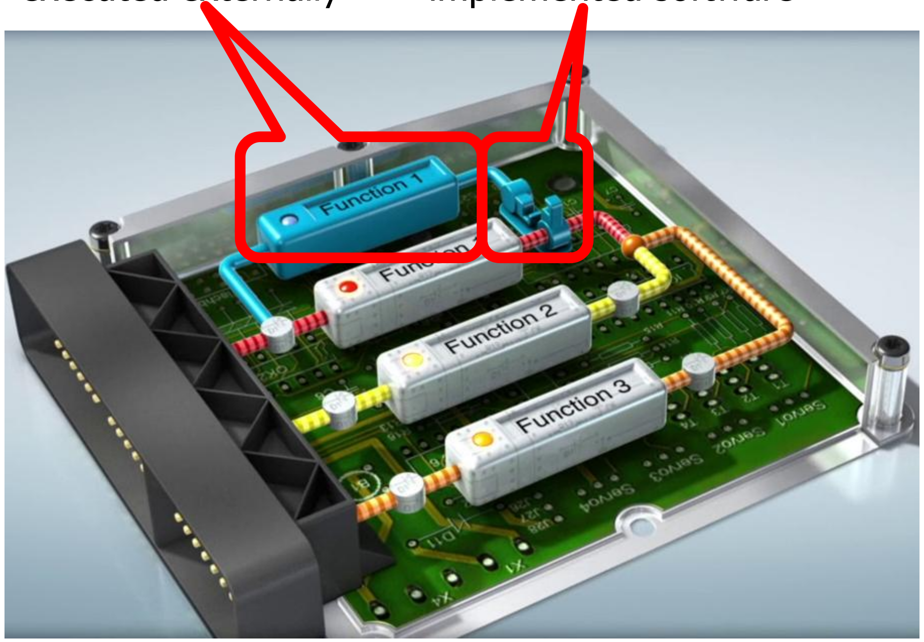
Tools and Methods for Validation and Verification
Prototyping Tools as Environment for Validation : example EHOOKs

ISO26262:
9.4.6/10.4.8: *„The test environment for software integration testing shall correspond as closely as possible to the target environment. ..."*

- ECU values for reading can be accessed via standard measurement
- EHOOKs is a PC software that allows to patch ECU code to access ECU values for writing.
- The module under test can either be executed ECU internally (= on target) or on external (RP) hardware
- The outputs of both implemented ECU module and module under test can be compared with a standard measurement tool (INCA)

The ECU function can be added to the ECU code or executed externally

The output of the simulated ECU function can be compared to the implemented software



Tools and Methods for Validation and Verification
Summary and conclusion

- State of the art prototyping methods are well suited to fulfil the requirements for development and test of safety related software according to ISO26262
- These methods need to be extended to cover new software architectures like AUTOSAR

- Following the MIL -> SIL -> PIL -> HIL approach scan satisfy ISO26262 compliant development
- But advanced tools allow to cover more test details already in earlier stages while reducing the effort and time
- As well as improving the quality of test capabilities, test cases and coverage

Thank you for your attention!