ON THE WAVE OF FUNCTIONAL SAFETY

JAROSLAV KADLEC

2KSYS

2KSYS

- History
  - Founded in 2015 as a start-up company
  - First projects oriented to custom HW design and
  - SW development for microcontrollers
  - 2018 moved to TITC - Technology Innovation Transfer Chamber

- Customers: MSR Engines, JetSurf, Microrisc, IMA …

2KSYS s.r.o.

INTRODUCTION

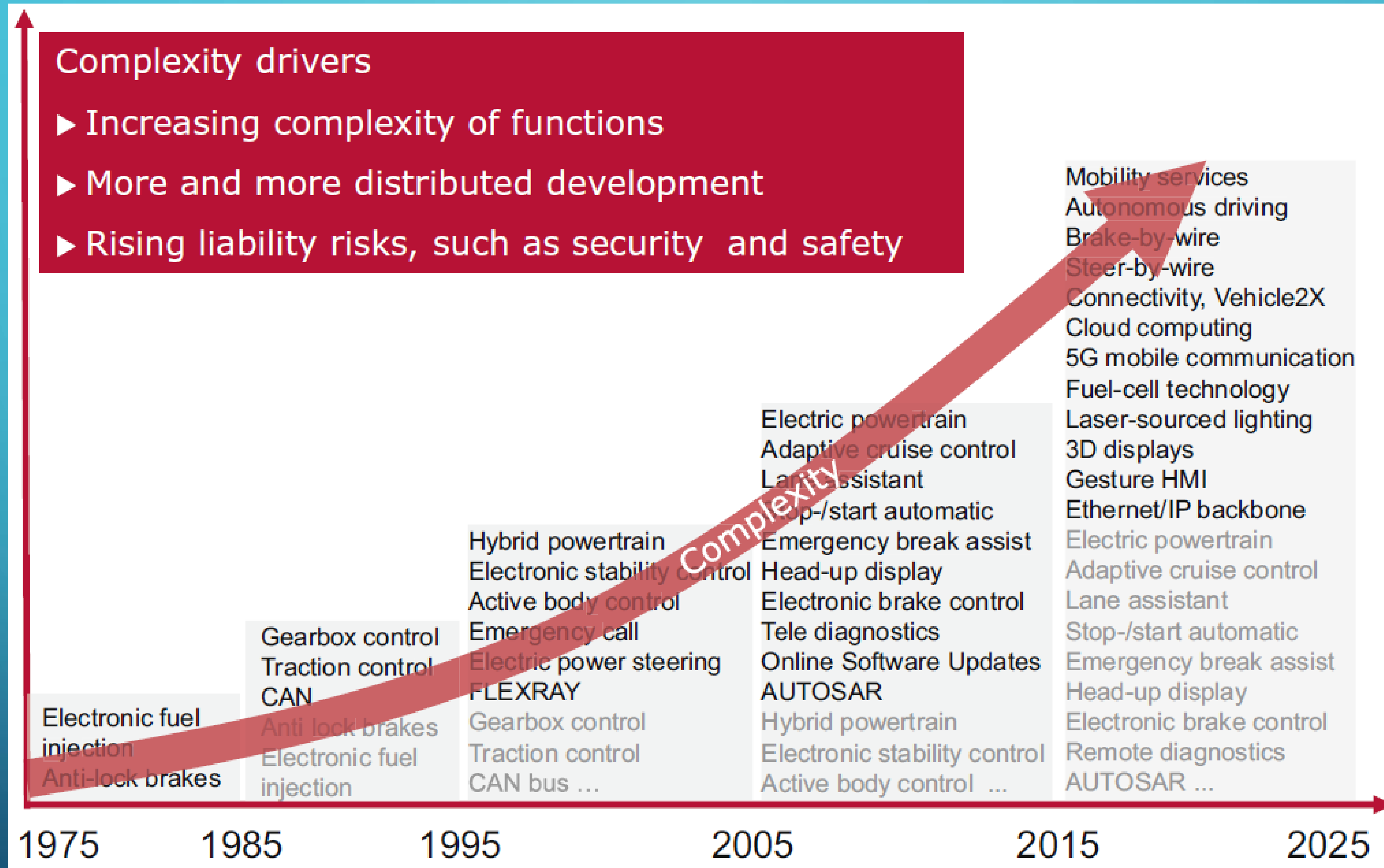
- Functional Safety
  - Electronics and Software Development for JETSURF
  - Automotive vs JETSURF
    - Engine as engine
    - Electronic Fuel Injection
    - The same requirements on Functional Safety for SW and HW
    - Distributed system

2KSYS s.r.o.

HISTORY OF AUTOMOTIVE

- 1970 – 1 ECU, 1 software
- 1990 - Distributed software
- 2000 - Advanced Driver Support, Adaptive Tempomat, Emergency breaking, introduced AUTOSAR
- 2007 - SW size measured in megabytes
- 2010 – C2X communication, autonomous driving
- 2016 - SW size measured in gigabytes
- today
  - 100 milion lines of code
  - Modern vehicles have more than 50 ECUs
    - Premium cars having more than 100 ECUs
  - Some functions, such as engine control or dynamics, are hard real-time functions, with reaction times going down to a few milliseconds
  - Practically all other functions, such as infotainment, demand at least soft real-time behaviors.

HISTORY OF AUTOMOTIVE



Source: Automotive Software Architectures: An Introduction 1st ed. 2017 Edition by Miroslaw Staron

2KSYS s.r.o.

TRENDS IN AUTOMOTIVE

- e-Mobility
- Connectivity and cooperation, IoT, Weareables
- Autonomous functions
- Shared cars
- Big data
- C2X
- Cloud technologies
- Security
- Safety

2KSYS s.r.o.

TRENDS IN AUTOMOTIVE SOFTWARE DEVELOPMENT

- Heterogenity of software
  - from highly safety-critical
  - up to infotainment
- Distribution of development - OEMs and suppliers
- Distribution of software - number of ECUs, need of coordinate SW distribution, OTA SW updates
- Variants and configurations - different requirements on the same car in different countries
- Unit-based cost models - competitive market pushes unit prices of all components down, CI
- Other trends
  - speed of SW development
  - data-driven development
  - ecosystems

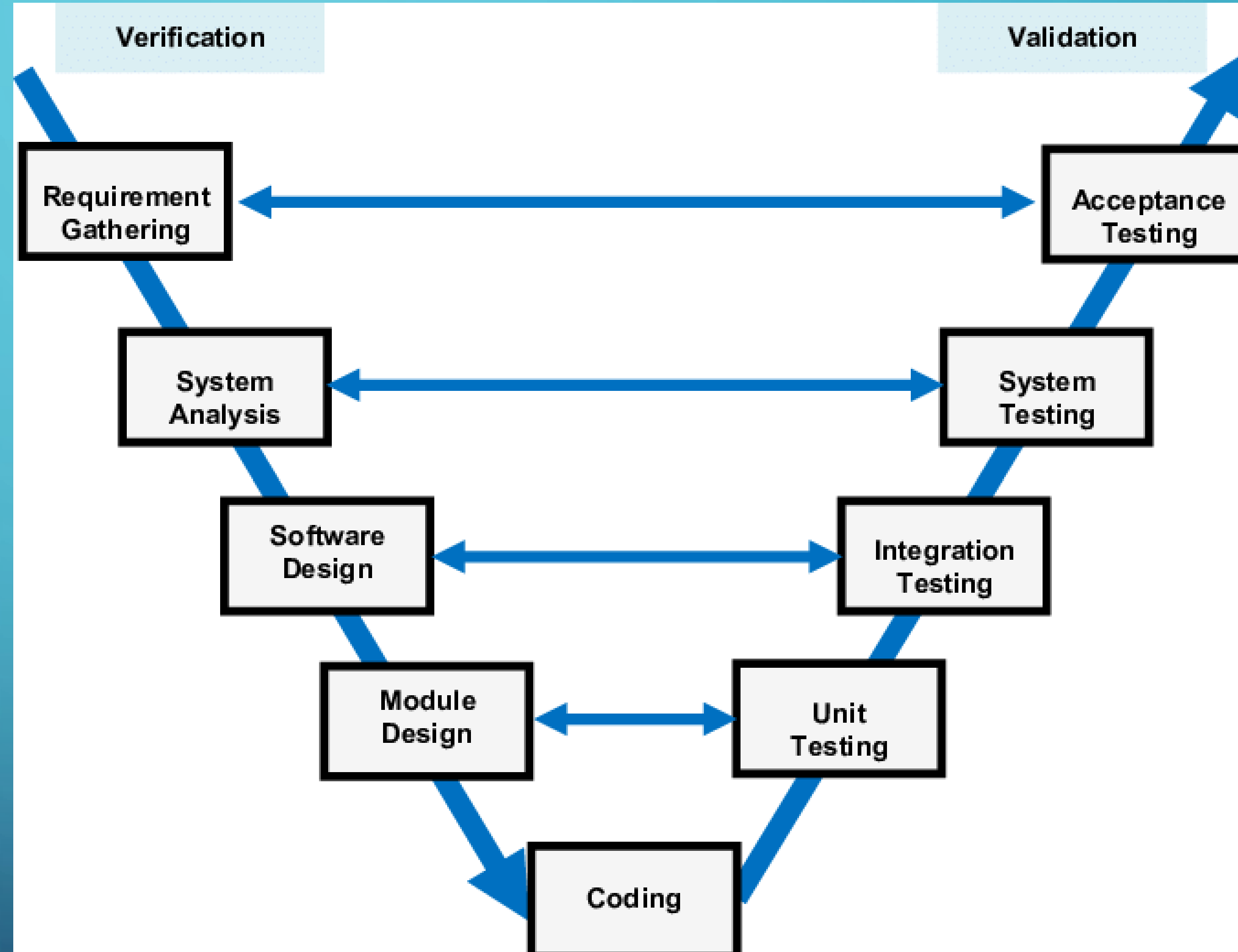
ORGANIZATION OF AUTOMOTIVE SOFTWARE SYSTEMS

- Each automotive area has its own requirements for computational speed, reliability, security, safety, flexibility, and extensibility
- Past
  - Each car manufacturer eveloped its own way of organizing software systems
    - Common signs
      - Similar organization of the electrical and SW systems
      - V development model
- Now
  - High effort in creation Common and standardized architecture
    - This resulted in AUTOSAR

2KSYS s.r.o.

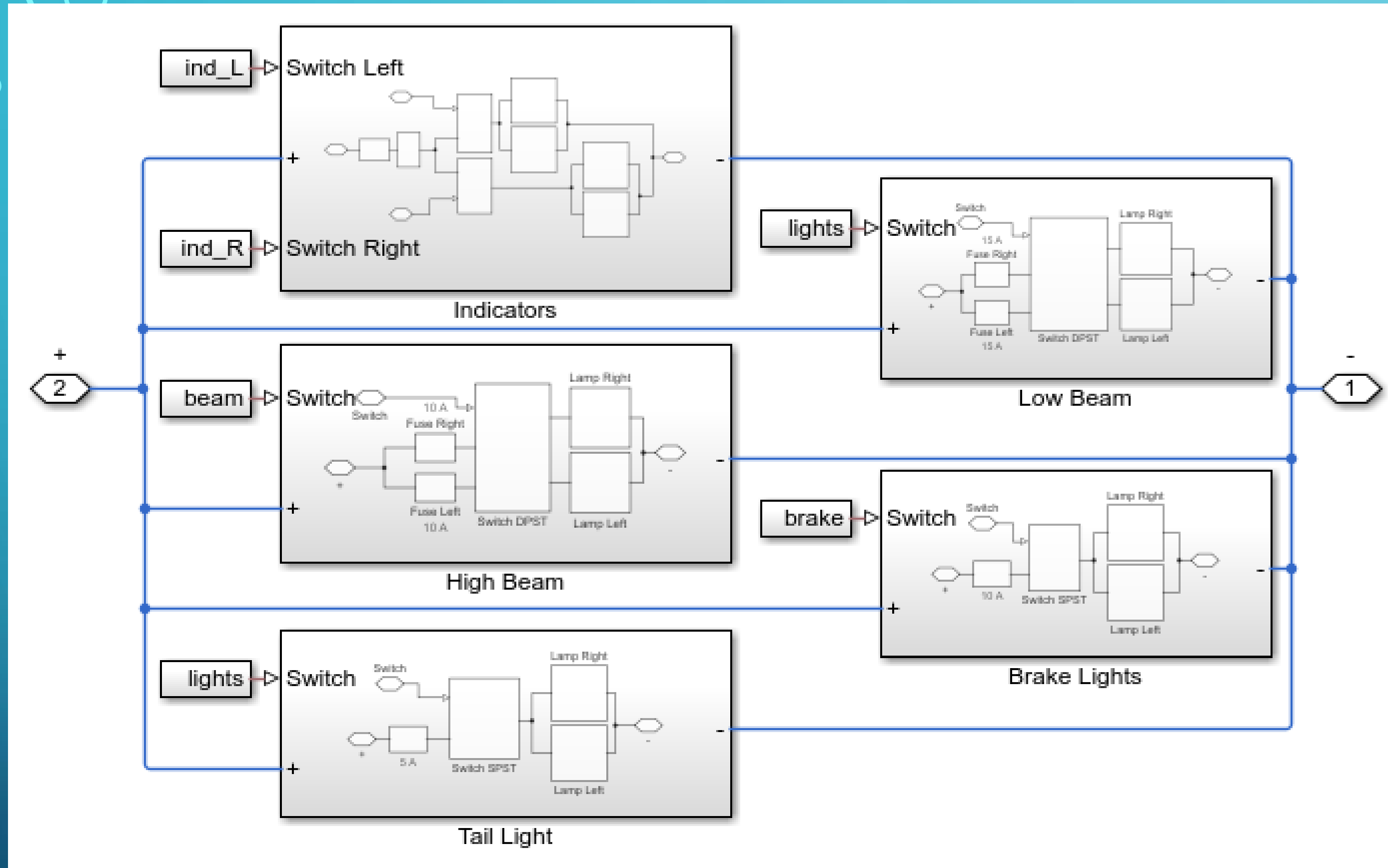
SOFTWARE DEVELOPMENT

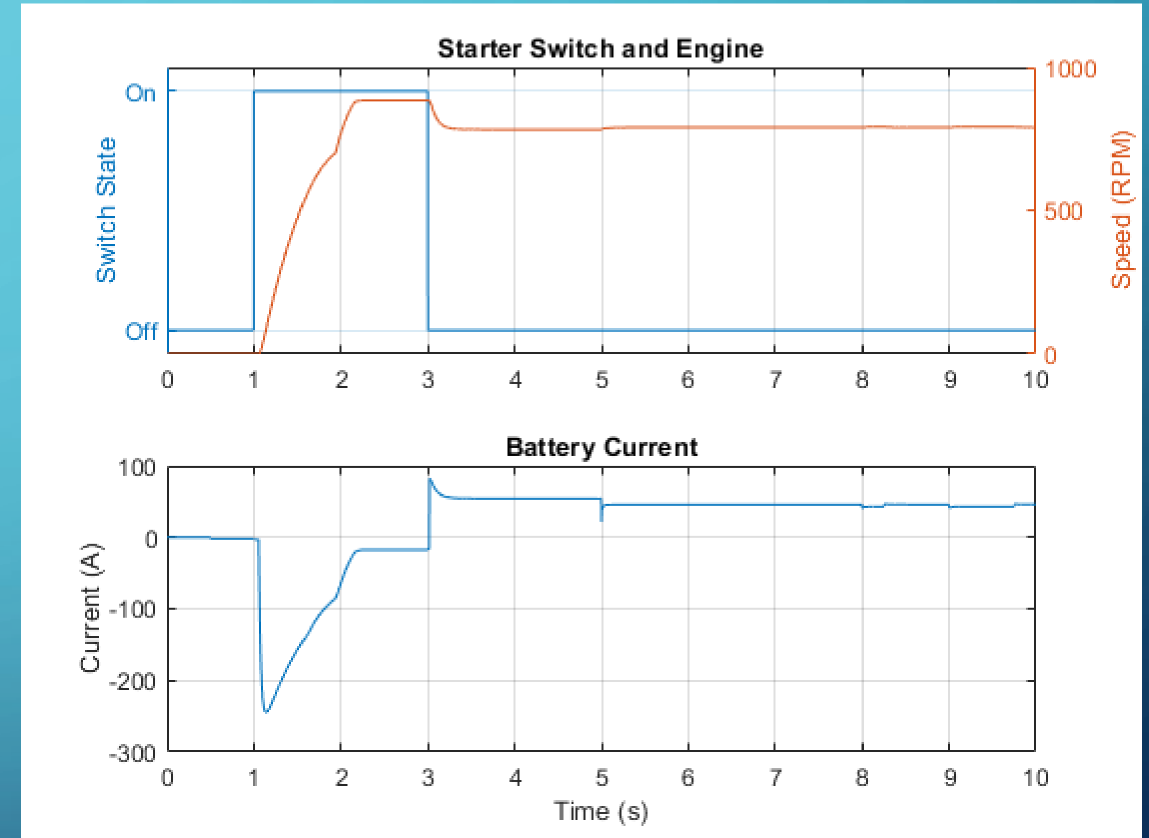
- Requirements engineering
- SW analysis
- SW architecting
- SW design
- Implementation
- Testing
    - Unit tests
    - Component tests
    - System test
    - Functional test
    - Customer test
- Simulink modelling



2KSYS s.r.o.

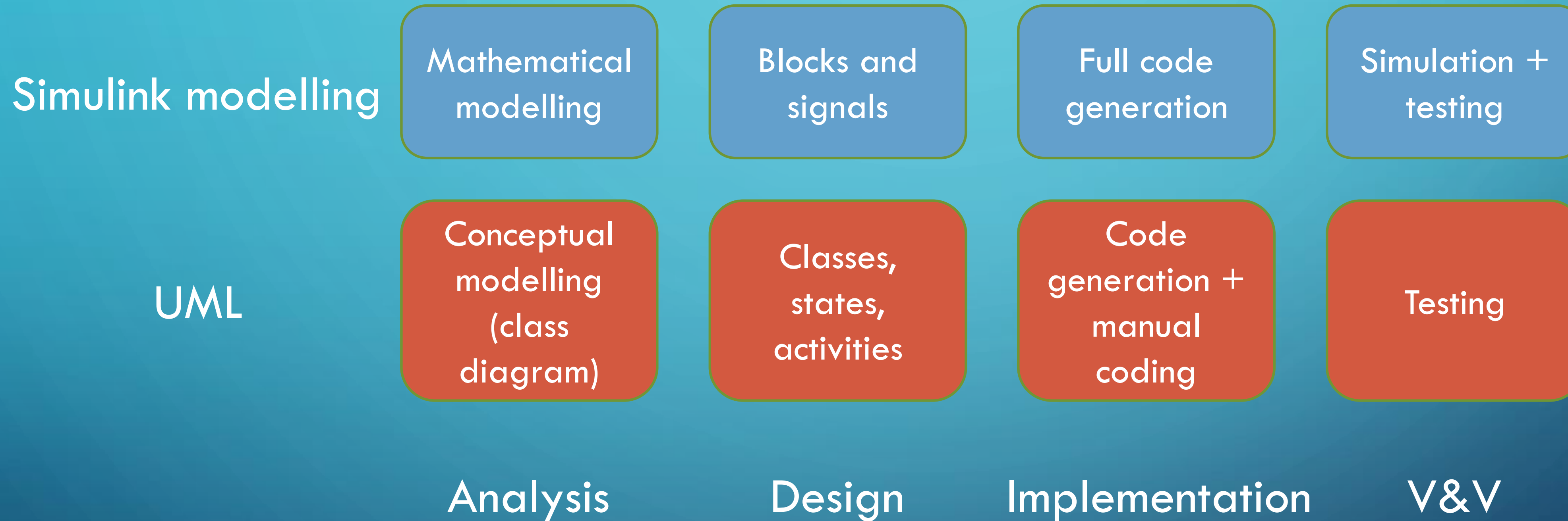
SIMULINK MODELLING





2KSYS s.r.o.

SIMULINK COMPARED TO SYSML/UML



2KSYS s.r.o.

FUNCTIONAL SAFETY OF AUTOMOTIVE SOFTWARE

- Much of this work is based on the ISO 26262 standard that was published in 2011.
  - 2018 update to most road vehicles
- Functional safety in ISO 26262 is defined as “absence of unreasonable risk due to hazards caused by malfunctioning behaviour of E/E systems”.

<table>
<tr><td colspan="4">Part 1 Vocabulary</td></tr>
<tr><td colspan="4">Part 2 Management of functional safety</td></tr>
<tr><td>Part 3 Concept phase</td><td colspan="2">Part 4 Product development on the system level</td><td rowspan="2">Part 7 Production, operation, service and decommissioning</td></tr>
<tr><td>Part 12 Adaptation of ISO 26262 for motorcycles</td><td>Part 5 Product development on the hardware level</td><td>Part 6 Product development on the software level</td></tr>
<tr><td colspan="4">Part 8 Supporting processes</td></tr>
<tr><td colspan="4">Part 9 ASIL-oriented and safety-oriented analyses</td></tr>
<tr><td colspan="4">Part 10 Guideline on ISO 26262 (informative)</td></tr>
<tr><td colspan="4">Part 11 Guideline on application of ISO 26262 to semiconductors (informative)</td></tr>
</table>

2KSYS s.r.o.

SOFTWARE SAFETY REQUIREMENTS

- Correct and safe execution of the intended functionality
- Monitoring that the system maintain a safe state
- Transitioning the system to a degraded state with reduced or no functionality, and keeping the system in that state
- Fault detection and handling hardware faults, including setting diagnostic fault codes
- Self-testing to find faults before they are activated
- Functionality related to production, service and decommissioning, e.g. calibration and deploying airbags during decommissioning

2KSYS s.r.o.

SAFETY CRITICAL CODE - MISRA

- The MISRA rules are often encoded in the C/C++ compilers used in safetycritical systems.
- The MISRA standard was revised in 2008 and later in 2012, leading to the addition of more rules.
- Today, we have over 200 rules, with the majority of them classified as “required”.

2KSYS s.r.o.

SAFETY CRITICAL CODE - NASA'S 10 RULES

1. Restrict all code to very simple control flow constructs, do not use **goto** statements, **setjmp** or **longjmp** constructs, direct or indirect recursion
2. Give all loops a fixed upper bound. It must be trivially possible for a checking tool to prove statically that the loop cannot exceed a preset upper bound on the number of iterations. If a tool cannot prove the loop bound statically, the rule is considered violated.
3. Do not use dynamic memory allocation after initialization.
4. No function should be longer than what can be printed on a single sheet of paper in a standard format with one line per statement and one line per declaration.
   - Typically, this means no more than about 60 lines of code per function.
5. The code's assertion density should average to minimally two assertions per function. Assertions must be used to check for anomalous conditions that should never happen in real-life executions. Assertions must be side effect-free and should be defined as Boolean tests. When an assertion fails, an explicit recovery action must be taken, such as returning an error condition to the caller of the function that executes the failing assertion.

SAFETY CRITICAL CODE - NASA'S 10 RULES

6. Declare all data objects at the smallest possible level of scope.
7. Each calling function must check the return value of non-void functions, and each called function must check the validity of all parameters provided by the caller.
8. The use of the preprocessor must be limited to the inclusion of header files and simple macro definitions. Token pasting, variable argument lists (ellipses), and recursive macro calls are not allowed. All macros must expand into complete syntactic units. The use of conditional compilation directives must be kept to a minimum.
9. The use of pointers must be restricted. Specifically, no more than one level of dereferencing should be used. Pointer dereference operations may not be hidden in macro definitions or inside typedef declarations. Function pointers are not permitted.
10. All code must be compiled, from the first day of development, with all compiler warnings enabled at the most pedantic setting available. All code must compile without warnings. All code must also be checked daily with at least one, but preferably more than one, strong static source code analyzer and should pass all analyses with zero warnings.

2KSYS s.r.o.

MECHANISMS OF DETECTING ERRORS IN ISO 26262

- Range checks of data
  - data read from or written to an interface is within a specified range of values
- Plausibility checks
  - check that can be used on signals between software units
- Detection of data errors
  - error detecting codes
  - checksums
  - redundant data storage
- External monitoring facility:
  - detect faults in execution
  - software executed in a different microcontroller
  - external watchdog

2KSYS s.r.o.

MECHANISMS OF DETECTING ERRORS IN ISO 26262

- Control flow monitoring
  - monitoring the execution flow of a software unit
  - certain faults can be detected, including skipped instructions and software stuck in infinite loops
- Diverse software design:
  - design two different software units monitoring each other

2KSYS s.r.o.

MECHANISMS FOR ERROR HANDLING

- Error recovery mechanism
  - The purpose is to go from a corrupted state back into a state from which normal operation can be continued.
- Graceful degradation
  - This method takes the system from a normal operation to a safe operation when faults are detected (e.g. warning lamp).
- Independent parallel redundancy
  - Quite costly
  - Need redundant hardware
- Correcting codes for data
  - Based on adding redundant data.
  - The more the redundant data that is used, the more the errors that can be corrected.

2KSYS s.r.o.

SAFE, RELIABLE AND ABOVE ALL FUN. IT'S TIME TO ENJOY!

2KSYS s.r.o.

JAROSLAV KADLEC

2KSYS

CZECH REPUBLIC

KADLEC@2KSYS.CZ