Railway control systems: Development of safety-critical software

Istvan Majzik
Budapest University of Technology and Economics
Dept. of Measurement and Information Systems
Contents

- The role of standards
- Development of railway control software
  - Safety lifecycle
  - Roles and competences of personnel
  - Techniques for design and V&V
  - Tools and languages
  - Documentation
- Case study: SAFEDMI
  - Hardware and software architecture
  - Verification techniques
The role of standards for railway control systems

How the development is influenced by the requirements of the standards?
Standards for railway control applications

- **Basic standard:**

- **Specific CENELEC standards derived from IEC 61508:**
  - EN 50126-1:2012 - Railway applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)
  - EN 50129:2003 - Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling
  - EN 50128:2011 - Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems
  - EN 50159:2010 - Railway applications - Communication, signalling and processing systems - Safety-related communication in transmission systems
Relation of railway related standards

- **EN 50126**: The Specification and Demonstration of RAMS
  - Safety Aspects
  - Software Aspects

- **EN 50129**: Safety related electronic systems for signalling
  - Physical Aspects
    - EN 50124: Distances for all EE equipments
    - EN 50122: Fixed installations - Electrical safety, earthing and the return circuit
    - EN 50119: Fixed installations - Electric traction overhead contact lines
    - EN 50123: Fixed installations - D.C. switchgear
    - EN 50155: Electronic equipment used on rolling stock
    - EN 50121: Railway applications - Electromagnetic compatibility

- **EN 50128**: Software for railway control and protection systems
  - Protocol Aspects
    - EN 50159: Safety-related communication in transmission systems

- **EN 50125**: Environmental conditions for equipment
  - Environment Aspects
Railway control software as safety-critical software
Software route map

- **Basic SIL concepts:**
  - **Software SIL shall be identical to the system SIL**
  - **Exception:** Software SIL can be reduced if mechanism exists to **prevent** the failure of a software component from causing the system to go to an unsafe state

- **Reducing software SIL requires:**
  - Analysis of failure modes and effects
  - Analysis of independence between software and the prevention mechanisms
Example: SCADA system architecture

Reducing SW component SIL by the following solutions:

- Processing in two channels
- Comparison of output signals at the I/O
- Comparison of visual output by the operator: Alternating bitmap visualization from the two channels (blinking if different)
- Detection of internal errors before the effects reach the outputs
Recall: Safety integrity requirements

- **Low demand mode** (low frequency of demands):

<table>
<thead>
<tr>
<th>SIL</th>
<th>Average probability of failure to perform the function on demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$10^{-2} \leq PFD &lt; 10^{-1}$</td>
</tr>
<tr>
<td>2</td>
<td>$10^{-3} \leq PFD &lt; 10^{-2}$</td>
</tr>
<tr>
<td>3</td>
<td>$10^{-4} \leq PFD &lt; 10^{-3}$</td>
</tr>
<tr>
<td>4</td>
<td>$10^{-5} \leq PFD &lt; 10^{-4}$</td>
</tr>
</tbody>
</table>

- **High demand mode** (high frequency or continuous demand):

<table>
<thead>
<tr>
<th>SIL</th>
<th>Probability of dangerous failure per hour per safety function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$10^{-6} \leq PFH &lt; 10^{-5}$</td>
</tr>
<tr>
<td>2</td>
<td>$10^{-7} \leq PFH &lt; 10^{-6}$</td>
</tr>
<tr>
<td>3</td>
<td>$10^{-8} \leq PFH &lt; 10^{-7}$</td>
</tr>
<tr>
<td>4</td>
<td>$10^{-9} \leq PFH &lt; 10^{-8}$</td>
</tr>
</tbody>
</table>

(PFH or THR)
Problems in demonstrating software SIL

- **Systematic failures** in complex software:
  - Development of *fault-free software cannot be guaranteed* in case of complex functions
    - Goal: Reducing the number of faults that may cause hazard
  - Target failure measure (hazard rate) *cannot be demonstrated by a quantitative analysis*
    - General techniques do not exist, estimations are questionable

→ **SW safety standards prescribe** methods and techniques for the software development, operation and maintenance:

1. Safety *lifecycle*
2. Competence and independence of *personnel*
3. *Techniques and measures* in all phases of the lifecycle
4. *Documentation*
Safety lifecycle
Software lifecycle

Basic principles:
- Top-down design
- Modularity
- Preparing test specifications together with the design specification
- Verification of each phase
- Validation
- Configuration management and change control
- Clear documentation and traceability
Software quality assurance

- **Software Quality Assurance Plan**
  - Determining all *technical and control activities* in the lifecycle
    - Activities, inputs and outputs (esp. verification and validation)
    - Quantitative *quality metrics*
    - Specification of *its own updating* (frequency, responsibility, methods)
  - Control of *external suppliers*

- **Software configuration management**
  - Configuration control before release for all artifacts
  - Changes require authorization

- **Problem reporting and corrective actions (issue tracking)**
  - “Lifecycle” of problems: From reporting through analysis, design and implementation to validation
  - Preventive actions
Generic software: It can be used and re-used after parameterization with specific data (e.g., station layout)
Parameterization of generic software

System development

- Requirement specification
  - Architecture design
    - Component design
      - Component coding
      - Component test spec.
      - Component testing
    - Component testing
  - Validation test specification
    - Integration test specification
    - Software integration
    - Software validation
    - Operation and maintenance
  - Software assessment

Parameterization

Design for parameterization

V&V of parameterization
Roles and competences in the lifecycle
Roles in the development lifecycle

1. Project Manager (PM)
2. Requirements Manager (RQM)
3. Designer (DES)
4. Implementer (IMP)
5. Tester (TST) – component and overall testing
6. Integrator (INT) – integration testing
7. Verifier (VER) – static verification
8. Validator (VAL) – overall satisfaction of req.s
9. Assessor (ASR) – external reviewer
The preferred organizational structure
Competence of personnel

- Competence shall be demonstrated for each role
  - Training, experience and qualifications

- Example: Competences of an Implementer
  - Shall be competent in engineering appropriate to the application area
  - Shall be competent in the implementation language and supporting tools
  - Shall be capable of applying the specified coding standards and programming styles
  - Shall understand all the constraints imposed by the hardware platform and the operating system
  - Shall understand the relevant parts of the standard
Techniques for design and V&V
Basic approach

- **Goal:** Preventing the introduction of *systematic faults* and controlling the *residual faults*

- **SIL** determines the set of *techniques to be applied* as:
  - **M:** Mandatory
  - **HR:** Highly recommended (rationale behind not using it should be detailed and agreed with the assessor)
  - **R:** Recommended
  - **---:** No recommendation for or against being used
  - **NR:** Not recommended

- **Combinations** of techniques is allowed:
  - E.g., alternative or equivalent techniques are marked

- **Hierarchy of methods** is formed (references to sub-tables)
## Example: Software design and implementation

<table>
<thead>
<tr>
<th>TECHNIQUE/MEASURE</th>
<th>Ref</th>
<th>SIL 0</th>
<th>SIL 1</th>
<th>SIL 2</th>
<th>SIL 3</th>
<th>SIL 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Formal Methods</td>
<td>D.28</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>2. Modelling</td>
<td>Table A.17</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>3. Structured methodology</td>
<td>D.52</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>4. Modular Approach</td>
<td>D.38</td>
<td>HR</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>5. Components</td>
<td>Table A.20</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>6. Design and Coding Standards</td>
<td>Table A.12</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>7. Analysable Programs</td>
<td>D.2</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>8. Strongly Typed Programming Language</td>
<td>D.49</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>9. Structured Programming</td>
<td>D.53</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>10. Programming Language</td>
<td>Table A.15</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>11. Language Subset</td>
<td>D.35</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>12. Object Oriented Programming</td>
<td>Table A.22 D.57</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>13. Procedural programming</td>
<td>D.60</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>14. Metaprogramming</td>
<td>D.59</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

### Requirements:

1. An approved combination of techniques for Software Safety Integrity Levels 3 and 4 is 4, 5, 6, 8 and one from 1 or 2.

2. An approved combination of techniques for Software Safety Integrity Levels 1 and 2 is 3, 4, 5, 6 and one from 8, 9 or 10.

3. Metaprogramming shall be restricted to the production of the code of the software source before compilation.
Example: Software Architecture

Combinations:

- "Approved combinations of techniques for Software SIL 3 and 4 are as follows:
  - 1, 7, 19, 22 and one from 4, 5, 12 or 21; or
  - 1, 4, 19, 22 and one from 2, 5, 12, 15 or 21."

- "Approved combinations of techniques for Software SIL 1 and 2 are as follows:
  - 1, 19, 22 and one from 2, 4, 5, 7, 12, 15 or 21."
Example: Verification and Testing

Requirements for SIL4:
- 5: Mandatory
- 4: Highly recommended
- 3: Recommended
- 2: No recommendation
- 1: Not recommended
Example: Integration and Overall SW Testing

Integration

Functional and Black-box Testing (A14)

Test Case Execution from Cause Consequence Diagrams

Prototyping/Animation

Boundary Value Analysis

Equivalence Classes and Input Partition Testing

Process Simulation

Avalanche/Stress Testing

Response Timing and Memory Constraints

Performance Requirements

Performance Testing (A18)

Performance Testing (A18)

Functional and Black-box Testing (A14)

Modelling (A17)
Specific techniques (examples)

- **Defensive programming**
  - Self-checking anomalous control/data flow and data values during execution (e.g., checking variable ranges, consistency of configuration) and react in a safe manner

- **Safety bag technique**
  - Independent external monitor ensuring that the behaviour is safe

- **Memorizing executed traces**
  - Comparison of program execution with previously documented reference execution in order to detect errors and fail safely

- **Test case execution from error seeding**
  - Inserting errors in order to estimate the number of remaining errors after testing – from the number of inserted and detected errors
### Tools and languages

<table>
<thead>
<tr>
<th>Offset</th>
<th>Format</th>
<th>Search</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td>novcamArea.xLogger.L0DC novcamArea.xLogger.xReg</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td></td>
<td>novcamArea.xLogger.L0DC novcamArea.xLogger.xReg</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td></td>
<td>novcamArea.xLogger.L0DC novcamArea.xLogger.xData(1)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td></td>
<td>novcamArea.xLogger.L0DC novcamArea.xLogger.xData(0)</td>
<td>EVENT_FLT_INJICT_START</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td></td>
<td>novcamArea.xLogger.L0DC novcamArea.xLogger.xData(0)</td>
<td>00000002</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td></td>
<td>novcamArea.xLogger.L0DC novcamArea.xLogger.xData(0)</td>
<td>1000</td>
</tr>
</tbody>
</table>
Tool classes

- **T1**: Generates outputs which *cannot contribute to the executable code* (and data) of the software
  - E.g.: a text editor, a requirement support tool, a configuration control tool

- **T2**: Supports the test or verification of the design or executable code, where *errors in the tool can fail to reveal defects*
  - E.g.: a test coverage measurement tool; a static analysis tool

- **T3**: Generates outputs which *can contribute to the executable code* (including data) of the system
  - E.g.: source code compiler, a data/algorithms compiler
Selection of software tools

- **Justification of the selection of T2 and T3 tools:**
  - Identification of potential failures in the tools output
  - Measures to avoid or handle such failures

- **Evidence in case of T3 tools:**
  - Output of the tool conforms to its specification
  - Or failures in the output are detected

**Sources of evidence:**

- Validation of the output of the tool: Based on the same steps necessary for a manual process as a replacement of the tool
- Validation of the tool: Sufficient test cases and their results
  - History of successful use in similar environments, for similar tasks
- Compliance with the safety integrity levels derived from the risk analysis of the process including the tools
- Diverse redundant code that allows the detection and control of tool failures
The programming language shall

- have a **translator which has been evaluated**, e.g., by a validation suite (test suite)
  - for a specific project: reduced to checking specific suitability
  - for a class of applications: all intended and appropriate use of the tool
- match the **characteristics of the application**,
- contain features that facilitate the **detection of design or programming errors**,
- support features that **match the design method**
### Requirements for languages

<table>
<thead>
<tr>
<th>TECHNIQUE/MEASURE</th>
<th>Ref</th>
<th>SIL 0</th>
<th>SIL 1</th>
<th>SIL 2</th>
<th>SIL 3</th>
<th>SIL 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ADA</td>
<td>D.54</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>2. MODULA-2</td>
<td>D.54</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>3. PASCAL</td>
<td>D.54</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>4. C or C++</td>
<td>D.54</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>D.35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. PL/M</td>
<td>D.54</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>6. BASIC</td>
<td>D.54</td>
<td>R</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>7. Assembler</td>
<td>D.54</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>8. C#</td>
<td>D.54</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>D.35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. JAVA</td>
<td>D.54</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>D.35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Statement List</td>
<td>D.54</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

- **Coding standards** (subsets of languages) are defined
  - “Dangerous” constructs are excluded (e.g., function pointers)
  - Static checking can be used to verify the subset
Interesting facts

- Boeing 777: Approx. 35 languages are used
  - Mostly Ada with assembler (e.g., cabin management system)
  - Onboard extinguishers in PLM
  - Seatback entertainment system in C++ with MFC

- European Space Agency:
  - Mandates Ada for mission critical systems

- Honeywell: Aircraft navigation data loader in C

- Lockheed: F-22 Advanced Tactical Fighter program in Ada 83 with a small amount in assembly

- GM trucks vehicle controllers mostly in Modula-GM (Modula-GM is a variant of Modula-2)

- TGV France: Braking and switching system in Ada

- Westinghouse: Automatic Train Protection (ATP) systems in Pascal
Restrictions using pre-existing software

- The following information about the pre-existing software shall clearly be identified and documented:
  - the requirements that it is intended to fulfil
  - the assumptions about the environment
  - interfaces with other parts of the software
  - Precise and complete description for the system integrator

- The pre-existing software shall be included in the validation process of the whole software

- For SIL 3 or SIL 4 the following precautions shall be taken:
  - analysis of its possible failures and their consequences
  - a strategy to detect failures and to protect the system from these
    - e.g., wrapper code to detect failures and isolate the unit
  - verification and validation of the following:
    - that it fulfils the allocated requirements
    - that its failures are detected and the system is protected
    - that the assumptions about the environment are fulfilled
**Specification of interfaces**

- **Pre/post conditions**
- **Data** from and to the interfaces
  - All **boundary values** for all specified data,
  - All **equivalence classes** for all specified data and each function
  - Unused or forbidden equivalence classes
- **Behaviour when the boundary value is exceeded**
- **Behaviour when the value is at the boundary**
- **For time-critical input and output data:**
  - Time constraints and requirements for correct operation
  - Management of exceptions
- **Allocated memory for the interface buffers**
  - The mechanisms to detect that the memory cannot be allocated or all buffers are full
- **Existence of synchronization mechanisms** between functions
Documentation
Documents in the software lifecycle

Software Planning Phase
- Software Quality Assurance Plan
- Software Configuration Management Plan
- Software Verification Plan
- Software Validation Plan
- Software Maintenance Plan

Software Development Phase (external)
- System Requirements Specification
- System Safety Requirements Specification
- System Architecture Description
- System Safety Plan Plan

Software Requirements Phase (7.2)
- Software Requirements Specification
- Overall Software Test Specification
- Software Requirements Verification Report

Software Arch. & Design Phase (7.3)
- Software Architecture Specification
- Software Design Specification
- Software Interface Specification
- Software Integration Test Specification
- Software/Hardware Integration Test Specification
- Software Architecture and Design Verification Report

Software Component Design Phase (7.4)
- Software Component Design Specification
- Software Component Test Specification
- Software Component Design Verification Report

Software Component Implementation Phase (7.5)
- Software Source Code & Supporting Documentation

Software Validation Phase (7.7)
- Overall Software Test Report
- Software Validation Report

Software Integration Phase (7.6)
- Software Integration Test Report
- Software/Hardware Integration Test Report
- Software Integration Verification Report

Software Maintenance Phase (9.2)
- Software Maintenance Records
- Software Change Records

Software Assessment Phase
- Software Assessment Plan
- Software Assessment Report
## Doc. control

- **Writing**
- **First check:** Verifier
- **Second check:** Validator
- **Third check:** Assessor

<table>
<thead>
<tr>
<th>PHASE</th>
<th>DOCUMENTATION</th>
<th>Written by</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; check</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; check</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Planning</strong></td>
<td>1. Software Quality Assurance Plan</td>
<td></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>2. Software Quality Assurance Verification Report</td>
<td><strong>VER</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. Software Configuration Management Plan</td>
<td></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>4. Software Verification Plan</td>
<td><strong>VER</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. Software Validation Plan</td>
<td><strong>VAL</strong></td>
<td><strong>VER</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Software requirements</strong></td>
<td>6. Software Requirements Specification</td>
<td><strong>REQ</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>7. Overall Software Test Specification</td>
<td><strong>TST</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>8. Software Requirements Verification Report</td>
<td><strong>VER</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Architecture and design</strong></td>
<td>9. Software Architecture Specification</td>
<td><strong>DES</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>10. Software Design Specification</td>
<td><strong>DES</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>11. Software Interface Specifications</td>
<td><strong>DES</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>12. Software Integration Test Specification</td>
<td><strong>INT</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>13. Software/Hardware Integration Test Specification</td>
<td><strong>INT</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>14. Software Architecture and Design Verification Report</td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Component design</strong></td>
<td>15. Software Component Design Specification</td>
<td><strong>DES</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>16. Software Component Test Specification</td>
<td><strong>TST</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>17. Software Component Design Verification Report</td>
<td><strong>VER</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Component implementation and testing</strong></td>
<td>18. Software Source Code and Supporting Documentation</td>
<td><strong>IMP</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>19. Software Source Code Verification Report</td>
<td><strong>VER</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20. Software Component Test Report</td>
<td><strong>TST</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td><strong>Integration</strong></td>
<td>21. Software Integration Test Report</td>
<td><strong>INT</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>22. Software/Hardware Integration Test Report</td>
<td><strong>INT</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>23. Software Integration Verification Report</td>
<td><strong>VER</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Overall software testing / Final validation</strong></td>
<td>24. Overall Software Test Report</td>
<td><strong>TST</strong></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
<tr>
<td></td>
<td>25. Software Validation Report</td>
<td><strong>VAL</strong></td>
<td><strong>VER</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>26. Tools Validation Report</td>
<td></td>
<td><strong>VER</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>27. Release Note</td>
<td></td>
<td><strong>VER</strong></td>
<td><strong>VAL</strong></td>
</tr>
</tbody>
</table>
Case study: SAFEDMI

Development of a safe driver-machine interface for ERTMS train control
What is ERTMS?

- **European Rail Traffic Management System**
  - Single Europe-wide standard for **train control and command** systems

- **Main components:**
  - European Train Control System (ETCS): standard for **in-cab train control**
  - GSM-R: the GSM mobile **communications standard** for railway operations (from/to control centers)

- **Equipment used:**
  - **On-board equipment**: e.g., EVC European Vital Computer for on-board train control
  - **Infrastructure equipment**: e.g., balise, an electronic transponder placed between the rails to give the exact location of a train
Development of a safe DMI

Main characteristics:

- Safety-critical functions
  - Information visualization (speedometer, odometer, ...)
  - Processing driver commands
  - Data transfer to EVC

- Safe wireless communication
  - System configuration
  - Diagnostics
  - Software update
Requirements

- **Safety:**
  - Safety Integrity Level: SIL 2
  - Tolerable Hazard Rate: \(10^{-7} \leq \text{THR} < 10^{-6}\) hazardous failures per hours
  - CENELEC standards: EN 50129 and EN 50128

- **Reliability:**
  - Mean Time To Failure: \(\text{MTTF} > 5000\) hours
    (5000 hours: \(\sim 7\) months)

- **Availability:**
  - \(A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}, \quad A > 0.9952\)
  - Faulty state: shall be less than 42 hours per year
  - MTTR < 24 hours if MTTF = 5000 hours
Fail-safe operation

Safe operation even in case of faults

Fail-stop behaviour

- Stopping (switch-off) is a safe state
- In case of a detected error the system has to be stopped
- Detecting errors is the main concern

Fail-operational behaviour

- Stopping (switch-off) is not a safe state
- Service is needed even in case of a detected error
  - full service
  - degraded (but safe) service
- Fault tolerance is required
Fail-safety concerns

Safety in case of single random hardware faults

Fault handling

Composite fail-safety

- Each function is implemented by at least 2 independent components
- Agreement between the independent components is needed to continue the operation

Reactive fail-safety

- Each function is equipped with an independent error detection
- The effects of detected errors can be handled

Inherent fail-safety

- All failure modes are safe
- "Inherent safe" system
The SAFEDMI hardware concept

- Single electronic structure based on **reactive fail-safety**
- Generic (off-the-shelf) hardware components are used
- Most of the safety mechanisms are **based on software implemented error detection and error handling**
The SAFEDMI hardware architecture

Commercial hardware components:
The SAFEDMI fault handling

- Operational modes:
  - Startup, Normal, Configuration and Safe (stopped) modes
  - Suspect state to implement controlled restart/stop after error: counting occurrences of errors in a given time period; forcing to Safe state (stop) in a given limit is exceeded
Error detection in Startup mode

Detection of permanent hardware faults by thorough self-testing

- Memory testing:
  - March algorithms (for stuck-at and coupling faults): writing and reading back regular 1 and 0 patterns stepwise

- CPU testing:
  - External watchdog circuit: Basic functionality (starting, heartbeat)
  - Self-test of functions: Core functionality $\rightarrow$ complex functionality (instruction decoding, register decoding, internal buses, arithmetic and logic unit)

- Integrity of software (in EEPROM):
  - Error detection codes

- Device testing (speaker, keyboard etc.):
  - Operator assistance is needed
Error detection in Normal/Config mode

- **Hardware devices:**
  - Scheduled low-overhead memory, video page and CPU tests
  - Acceptance checks for I/O

- **Communication and configuration functions:**
  - Assertions for data acceptance / credibility checks of internal data
  - Error detection and correction codes for messages

- **Operation mode control and driver input processing:**
  - Control flow monitoring (based on the program control flow graph)
  - Time-out checking for operations
  - Acknowledgement procedure: the driver shall confirm risky operations

- **Visualization of train data (bitmap computations):**
  - Duplicated computation and comparison of the results
  - Visual comparison by the driver (periodic change of bitmaps)
Testing the DMI
Main test groups:

- ERTMS functions
  - Interactions with the driver
  - Interactions with the EVC
- Internal safety mechanisms
- Wireless communications
Testing the ERTMS functions

- Sequences of test inputs: DMI inputs + workload
- Test output: DMI display + Diagnostic device

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Expected Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><strong>Driver:</strong> give traction to the train</td>
<td><strong>SAFEDMI:</strong> the current train speed increases.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>SAFEDMI:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The text message “Entry in Full Supervision Mode” is shown and a sound is produced.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- the FS mode icon is shown in area B7;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- in area A2 the distance to target is shown;</td>
</tr>
<tr>
<td>2.</td>
<td>None</td>
<td><strong>SAFEDMI:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- In area A1 the warning to avoid brake intervention is displayed and sound is produced;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- In area E1 the icon (Brake applied) is shown;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- In area C9 the icon (Service brake intervention or emergency brake intervention) is shown.</td>
</tr>
<tr>
<td>3.</td>
<td><strong>Driver:</strong> give traction to the train until the current train speed overcomes the permitted speed.</td>
<td><strong>SAFEDMI:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- In area A1 the warning to avoid brake intervention is displayed and sound is produced;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- In area E1 the icon (Brake applied) is shown;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- In area C9 the icon (Service brake intervention or emergency brake intervention) is shown.</td>
</tr>
</tbody>
</table>
Simulating the workload:
- signals from balises on a given route
- control messages from the railway regulation control center
Plus: Diagnostic device
Output of the diagnostic device

<table>
<thead>
<tr>
<th>Offset</th>
<th>Format</th>
<th>Remark</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[00]</td>
<td>DDEC</td>
<td>novramArea.xLogger.ulNumReg</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Format</th>
<th>Remark</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[00]</td>
<td>DDEC</td>
<td>novramArea.xLogger.ulNextIdx</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offset</th>
<th>Format</th>
<th>Remark</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[00]</td>
<td>URNV</td>
<td>novramArea.xLogger.xData[0].ulTimestamp</td>
<td>2</td>
</tr>
<tr>
<td>[04]</td>
<td>URNV</td>
<td>novramArea.xLogger.xData[0].ulParam1</td>
<td>EVENT_PLT_INJECT_START</td>
</tr>
<tr>
<td>[08]</td>
<td>LHEX</td>
<td>novramArea.xLogger.xData[0].ulParam2</td>
<td>0000000Z</td>
</tr>
<tr>
<td>[0C]</td>
<td>LDEC</td>
<td>novramArea.xLogger.xData[0].ulParam2</td>
<td>1000</td>
</tr>
</tbody>
</table>
Robustness testing

- Focus: Exceptional and extreme inputs, overload

- Testing behaviour on the driver interface:
  - Handling buttons: pressing more buttons simultaneously, ...
  - Input fields: empty, full, invalid characters, ...

- Testing behaviour on the EVC interface:
  - Invalid messages: empty, garbage, invalid fields, flooding, ...
Testing the internal mechanisms

- **Operational modes** and the corresponding functions
  - Activation of operational modes, configuration, disconnection from the environment
  - Coverage of the state machine of the operational modes
  - Coverage of the state machine of error counting

- **Performance**: Testing deadlines in case of maximum workload (specified on the EVC interface)

- **Handling of buttons**: Blocked buttons, safety acknowledgements, ordering of events

- **Handling temperature sensors**: Startup and operational temperature conditions (tested in climate test chamber)
Systematic testing

- Testing the operational modes:
  - Covering each state and each state transition

State machine of the operational modes

State machine of error counting
Testing the internal safety functions

- **Targeted fault injection**: Testing the implementation of the software based error detection and error handling mechanisms
  - Test goals:
    - The injected errors are **detected** by the implemented mechanisms
    - The proper error handling is **triggered**
  - Tested error detection mechanisms:
    - Control flow checking, data acceptance checking, duplicated execution and comparison, time-out checking

- **Random fault injection**: Evaluation of error detection coverage
  - Collecting data for coverage statistics

- Checking hardware self-tests in specific configurations
  - Hardware checks (RAM, ROM, video page)
  - I/O device checks (cabin, LCD, temperature)
Software based fault injection
Collecting diagnostic data

```
<table>
<thead>
<tr>
<th>Offset</th>
<th>Format</th>
<th>Remark</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[00]uLocalTime</td>
<td>UCV</td>
<td>uLocalTime</td>
<td></td>
</tr>
<tr>
<td>[00]uLastTime</td>
<td>UCV</td>
<td>uLastTime</td>
<td></td>
</tr>
<tr>
<td>[00]ubFaultStep</td>
<td>UBC</td>
<td>ubFaultStep</td>
<td></td>
</tr>
<tr>
<td>[00]ulTotalFaultCoulter</td>
<td>UBC</td>
<td>ulTotalFaultCoulter</td>
<td></td>
</tr>
<tr>
<td>[00]ulFaultCounter[0]</td>
<td>UBC</td>
<td>ulFaultCounter[0]</td>
<td></td>
</tr>
<tr>
<td>[00]faultParam[0]</td>
<td>UBC</td>
<td>faultParam[0]</td>
<td></td>
</tr>
<tr>
<td>[00]ubFltSequenceId</td>
<td>UBC</td>
<td>ubFltSequenceId</td>
<td></td>
</tr>
<tr>
<td>[00]ulFaultPeriod</td>
<td>UCV</td>
<td>ulFaultPeriod</td>
<td></td>
</tr>
</tbody>
</table>
```

Sample file path: `C:\SafeDmi\diag\FaultInjector.dat`
Testing the wireless communication

- **Scenario based** testing: Communication scenarios
- **Normal** operation:
  - Protocol testing: Establishing connection, message processing, closing the connection
- **Operation in case of transmission errors**:
  - Error detection mechanisms (EDC, ECC)
  - Closing the connection in case of too frequent errors
Wrapper configuration for testing

Session control
- CIS (installed on DMI)

System under test
- DMI
- IUT
- Wrapper

Bridge device
- BD

Test control
- SAVS

Connections:
- CIS/DMI broadcast
- Control Data
- Perf. Obs. Data
- DMI/BD session setup
- Session signaling
- Session data

Diagram showing the connections and flow between the components.
Summary

- The role of standards
- Development of railway control software
  - Safety lifecycle
  - Roles and competences
  - Techniques for design and V&V
  - Tools and languages
  - Documentation
- Case study: SAFEDMI
  - Hardware and software architecture
  - Verification techniques