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
  - 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
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):

  - **SIL**
  - **Average probability of failure to perform the function on demand**

    | SIL | Probability of failure to perform the function on demand |
    |-----|----------------------------------------------------------|
    | 1   | $10^{-2} \leq PFD < 10^{-1}$                           |
    | 2   | $10^{-3} \leq PFD < 10^{-2}$                           |
    | 3   | $10^{-4} \leq PFD < 10^{-3}$                           |
    | 4   | $10^{-5} \leq PFD < 10^{-4}$                           |

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

  - **SIL**
  - **Probability of dangerous failure per hour per safety function**

    | SIL | Probability of dangerous failure per hour per safety function |
    |-----|---------------------------------------------------------------|
    | 1   | $10^{-6} \leq PFH < 10^{-5}$                                  |
    | 2   | $10^{-7} \leq PFH < 10^{-6}$                                  |
    | 3   | $10^{-8} \leq PFH < 10^{-7}$                                  |
    | 4   | $10^{-9} \leq PFH < 10^{-8}$                                  |

  (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
Development of generic software

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 testing

Component test spec.

Integration test specification

Validation test specification

Software integration

Software validation

Software assessment

Operation and maintenance

V&V of parameterization

Parameterization

Design for parameterization
Roles and competences in the lifecycle
Roles in the development lifecycle

- Project Manager (PM)
- Requirements Manager (RQM)
- Designer (DES)
- Implementer (IMP)
- Tester (TST) – component and overall testing
- Integrator (INT) – integration testing
- Verifier (VER) – static verification
- Validator (VAL) – overall satisfaction of req.s
- Assessor (ASR) – external reviewer
The preferred organizational structure
Competences

- 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, 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</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>D.57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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."

<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. Defensive Programming</td>
<td>D.14</td>
<td>-</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>2. Fault Detection &amp; Diagnosis</td>
<td>D.26</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>3. Error Correcting Codes</td>
<td>D.19</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4. Error Detecting Codes</td>
<td>D.19</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>5. Failure Assertion Programming</td>
<td>D.24</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>6. Safety Bag Techniques</td>
<td>D.47</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>7. Diverse Programming</td>
<td>D.16</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>8. Recovery Block</td>
<td>D.44</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>9. Backward Recovery</td>
<td>D.5</td>
<td>-</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>10. Forward Recovery</td>
<td>D.30</td>
<td>-</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>11. Retry Fault Recovery Mechanisms</td>
<td>D.46</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>12. Memorising Executed Cases</td>
<td>D.36</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>13. Artificial Intelligence – Fault Correction</td>
<td>D.1</td>
<td>-</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>14. Dynamic Reconfiguration of software</td>
<td>D.17</td>
<td>-</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>15. Software Error Effect Analysis</td>
<td>D.25</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>16. Graceful Degradation</td>
<td>D.31</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>17. Information Hiding</td>
<td>D.33</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>18. Information Encapsulation</td>
<td>D.33</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>19. Fully Defined Interface</td>
<td>D.38</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>20. Formal Methods</td>
<td>D.28</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>21. Modelling</td>
<td>Table A.17</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>22. 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>23. Modelling supported by computer aided design and specification tools</td>
<td>Table A.17</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
</tbody>
</table>
Example: Verification and Testing

Requirements for SIL4:
- 5: Mandatory
- 4: Highly recommended
- 3: Recommended
- 2: No recommendation
- 1: Not recommended

- Formal Proof
  - Boundary Value Analysis
    - Checklists
    - Control Flow Analysis
    - Data Flow Analysis
    - Error Guessing
    - Walkthroughs/Design Reviews
  - Test Case Execution from Boundary Value Analysis
    - Test Case Execution from Error Guessing
    - Test Case Execution from Error Seeding
    - Performance Modelling
    - Equivalence Classes and Input Partition Testing
    - Structure-Based Testing

- Static Analysis (A15)
  - Data Flow Analysis
  - Error Guessing
  - Walkthroughs/Design Reviews

- Dynamic Analysis and Testing (A13)
  - Test Case Execution from Error Seeding
  - Performance Modelling
  - Equivalence Classes and Input Partition Testing
  - Structure-Based Testing

- Metrics

- Traceability

- Software Error Effect Analysis
  - 1. Statement
  - 2. Branch
  - 3. Compound Condition
  - 4. Data flow
  - 5. Path

- Functional/ Black-box Testing (A14)

- Performance Testing (A18)

- Interface Testing
Example: Integration and Overall SW Testing

Integration

Functional and Black-box Testing (A14)

Performance Testing (A18)

Overall Software Testing

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
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
Programming languages

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

- 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
  - 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

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

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

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

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

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

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

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

8. Software Assessment Phase
   - Software Assessment Plan
   - Software Assessment Report

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

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

11. Software Component Testing Phase (7.5)
    - Software Component Test Report
    - Software Source Code Verification 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>1st check</th>
<th>2nd check</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Planning</strong></td>
<td></td>
<td></td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>1. Software Quality Assurance Plan</td>
<td></td>
<td>a</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>2. Software Quality Assurance Verification Report</td>
<td></td>
<td>VER</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>3. Software Configuration Management Plan</td>
<td></td>
<td>see B.10</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>4. Software Verification Plan</td>
<td></td>
<td>VER</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>5. Software Validation Plan</td>
<td></td>
<td>VAL</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td><strong>Software requirements</strong></td>
<td></td>
<td></td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>6. Software Requirements Specification</td>
<td></td>
<td>REQ</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>7. Overall Software Test Specification</td>
<td></td>
<td>TST</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>8. Software Requirements Verification Report</td>
<td></td>
<td>VER</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td><strong>Architecture and design</strong></td>
<td></td>
<td></td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>9. Software Architecture Specification</td>
<td></td>
<td>DES</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>10. Software Design Specification</td>
<td></td>
<td>DES</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>11. Software Interface Specifications</td>
<td></td>
<td>DES</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>12. Software Integration Test Specification</td>
<td></td>
<td>INT</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>13. Software/Hardware Integration Test Specification</td>
<td></td>
<td>INT</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td><strong>Component design</strong></td>
<td></td>
<td></td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>15. Software Component Design Specification</td>
<td></td>
<td>DES</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>16. Software Component Test Specification</td>
<td></td>
<td>TST</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>17. Software Component Design Verification Report</td>
<td></td>
<td>VER</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td><strong>Component implementation and testing</strong></td>
<td></td>
<td></td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>18. Software Source Code and Supporting Documentation</td>
<td></td>
<td>IMP</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>20. Software Component Test Report</td>
<td></td>
<td>TST</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td><strong>Integration</strong></td>
<td></td>
<td></td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>21. Software Integration Test Report</td>
<td></td>
<td>INT</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>22. Software/Hardware Integration Test Report</td>
<td></td>
<td>INT</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>23. Software Integration Verification Report</td>
<td></td>
<td>VER</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td><strong>Overall software testing / Final validation</strong></td>
<td></td>
<td></td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>24. Overall Software Test Report</td>
<td></td>
<td>TST</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>25. Software Validation Report</td>
<td></td>
<td>VAL</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>26. Tools Validation Report</td>
<td></td>
<td>a</td>
<td>VER</td>
<td>VAL</td>
</tr>
<tr>
<td>27. Release Note</td>
<td></td>
<td>a</td>
<td>VER</td>
<td>VAL</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: ~ 7 months)

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

Fail-safe operation

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

Safe operation even in case of faults
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.
Detection of permanent hardware faults by thorough self-testing

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

- CPU testing:
  - External watchdog circuit: Basic functionality (starting, heartbeat)
  - Self-test: Core functionality → 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:
  - Data acceptance / credibility checks for 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. Driver: give traction to the train</td>
<td>SAFEDMI: the current train speed increases.</td>
<td></td>
</tr>
</tbody>
</table>
| 2. None | SAFEDMI:  
  - The text message “Entry in Full Supervision Mode” is shown and a sound is produced.  
  - the FS mode icon is shown in area B7;  
  - in area A2 the distance to target is shown; | |
| 3. Driver: give traction to the train until the current train speed overcomes the permitted speed. | SAFEDMI:  
  - In area A1 the warning to avoid brake intervention is displayed and sound is produced;  
  - In area E1 the icon (Brake applied) is shown;  
  - In area C9 the icon (Service brake intervention or emergency brake intervention) is shown. | |
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

### Logger

<table>
<thead>
<tr>
<th>Offset</th>
<th>Format</th>
<th>Remark</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[00]novramArea.xLogger.ulNumReg</td>
<td>DBC</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]novramArea.xLogger.ulNextIdx</td>
<td>DBC</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]novramArea.xLogger.xData[1]</td>
<td>LCNV</td>
<td>novramArea.xLogger.xData[0].ulTimestamp</td>
<td>2</td>
</tr>
<tr>
<td>[04]novramArea.xLogger.LDSC</td>
<td>Evento</td>
<td>novramArea.xLogger.xData[0].ulParam1</td>
<td>EVENT_PLT_INJECT_START</td>
</tr>
<tr>
<td>[08]novramArea.xLogger.LHEX</td>
<td>novramArea.xLogger.xData[0].ulParam1</td>
<td>0000000Z</td>
<td></td>
</tr>
<tr>
<td>[0C]novramArea.xLogger.LDEC</td>
<td>novramArea.xLogger.xData[0].ulParam2</td>
<td>1000</td>
<td></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 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

![Fault Injector interface showing various data fields and values for diagnostic purposes.](image)

- **DMIF:LocalTime**
  - Offset: 00
  - Format: LocalTime
  - Value: (value)

- **DMIF:LastTime**
  - Offset: 00
  - Format: LastTime
  - Value: (value)

- **DMIF:ubFaultStep**
  - Offset: 00
  - Format: ubFaultStep
  - Value: (value)

- **DMIF:ulTotalFaultCoulter**
  - Offset: 00
  - Format: ulTotalFaultCoulter
  - Value: (value)

- **DMIF:ulFaultCounter[0]**
  - Offset: 00
  - Format: ulFaultCounter[0]
  - Value: (value)

- **DMIF:FaultParam[0]**
  - Offset: 00
  - Format: FaultParam[0]
  - Value: (value)

- **DMIF:ubFltSequenceId**
  - Offset: 00
  - Format: ubFltSequenceId
  - Value: (value)

- **DMIF:ulFaultPeriod**
  - Offset: 00
  - Format: ulFaultPeriod
  - Value: (value)

- **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
- Bridge device
  - BD
- Test control
  - SAVS

Connections:
- DMI broadcast
- Control Data
- Perf. Obs. Data
- DMI/BD session setup
- Session signaling
- Session data
Evaluation of the DMI
Goals and challenges of the evaluation

Evaluation techniques

- DMI architecture
  - hazardous failure rate
  - reliability
  - availability
  Challenge: On-line tests and checks

- Wireless communication
  - performance: throughput, delay
  - error rate
  - connection management
  Challenge: Safe protocol stack with several layers

- Detection codes
  - detection quality
  - residual errors
  Challenge: Inherent complexity of computations
Evaluation of the DMI architecture

- Model based evaluation approach:
  - Construction of an analytical dependability model representing:
    - fault activation, error propagation processes,
    - error detection and error handling mechanisms
  - Stochastic Activity Network formalism (~ stochastic Petri Nets)
  - Sub-models assigned to architectural components:
    - Resources with fault activation and periodic tests
    - Propagation from active/passive resources to tasks
    - Tasks with on-line error detection techniques
    - Operational mode changes according to events and detected errors

- Analysis results:
  - Availability and safety (SIL 2) requirements are satisfied
  - Sensitivity analysis was performed to find optimization possibilities
Evaluation of the DMI architecture

UML based architecture model

Analysis subnets

Dependability model construction tool

System level dependability model

Dependability measures, sensitivity results
Results of the dependability analysis

- **MTTF (Mean time to failure)**
  - MTTF = 47 000 hours
  - Availability is computed on the basis of MTTR

- **MTTH (Mean time to hazard)**
  - Focusing on hazardous failures
  - MTTH = 1 482 000 hours

- **Hazardous failure rate**
  - Computed as 1/MTTH
  - 6.7 * 10^{-7} per hour → satisfies SIL2

- **Sensitivity analysis w.r.t. hazardous failure rate**
Example: Efficiency of control flow checking

- If the coverage falls below 50% then the SIL2 requirement is not satisfied (HR > 10^{-6})
Example: Efficiency of duplicated execution

- SIL 2 requirement is not satisfied if the duplicated execution and comparison is replaced with a less efficient error detection technique (HR > 10^{-6})
Summary of the evaluation activities

- Experimental analysis of **schedulability** and **real-time** properties
- Fault injection based experimental evaluation of **error detection**
- Model based analysis of **reliability**, **availability** and **hazardous failure rate**
- Evaluation of the **detection property of codes**
- Evaluation of the **performance** and **dependability** properties of the wireless communication
- Model based evaluation of the **effect of DMI failures** on **QoS** of the train control system
- Evaluation of **wireless DMI-EVC communication**
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: testing and evaluation