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

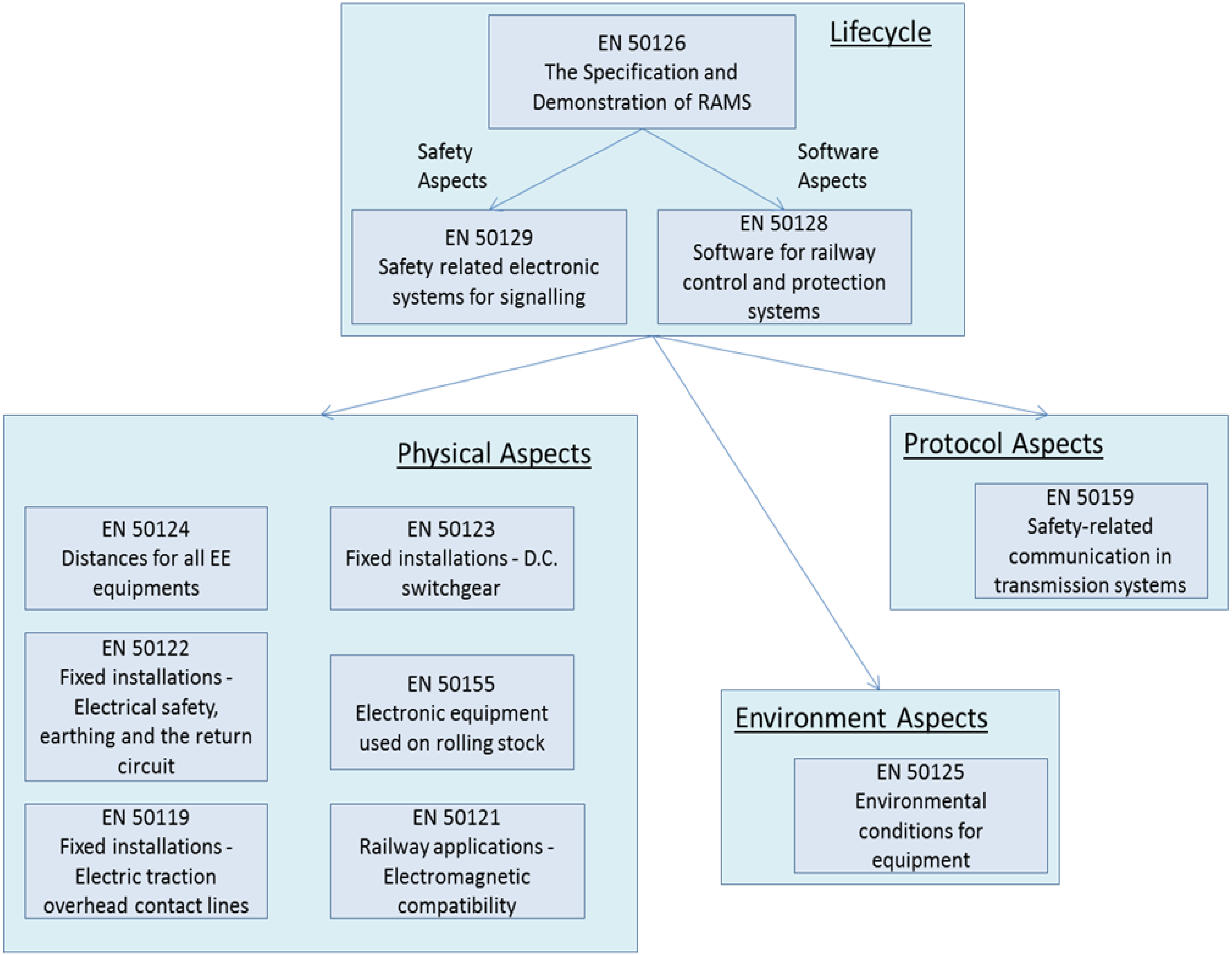
The role of standards for railway control systems

How the development is influenced by the requirements of the standards?

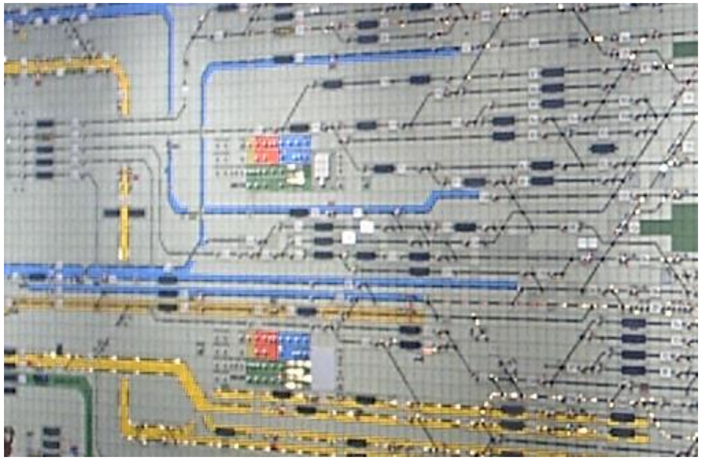
Standards for railway control applications

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

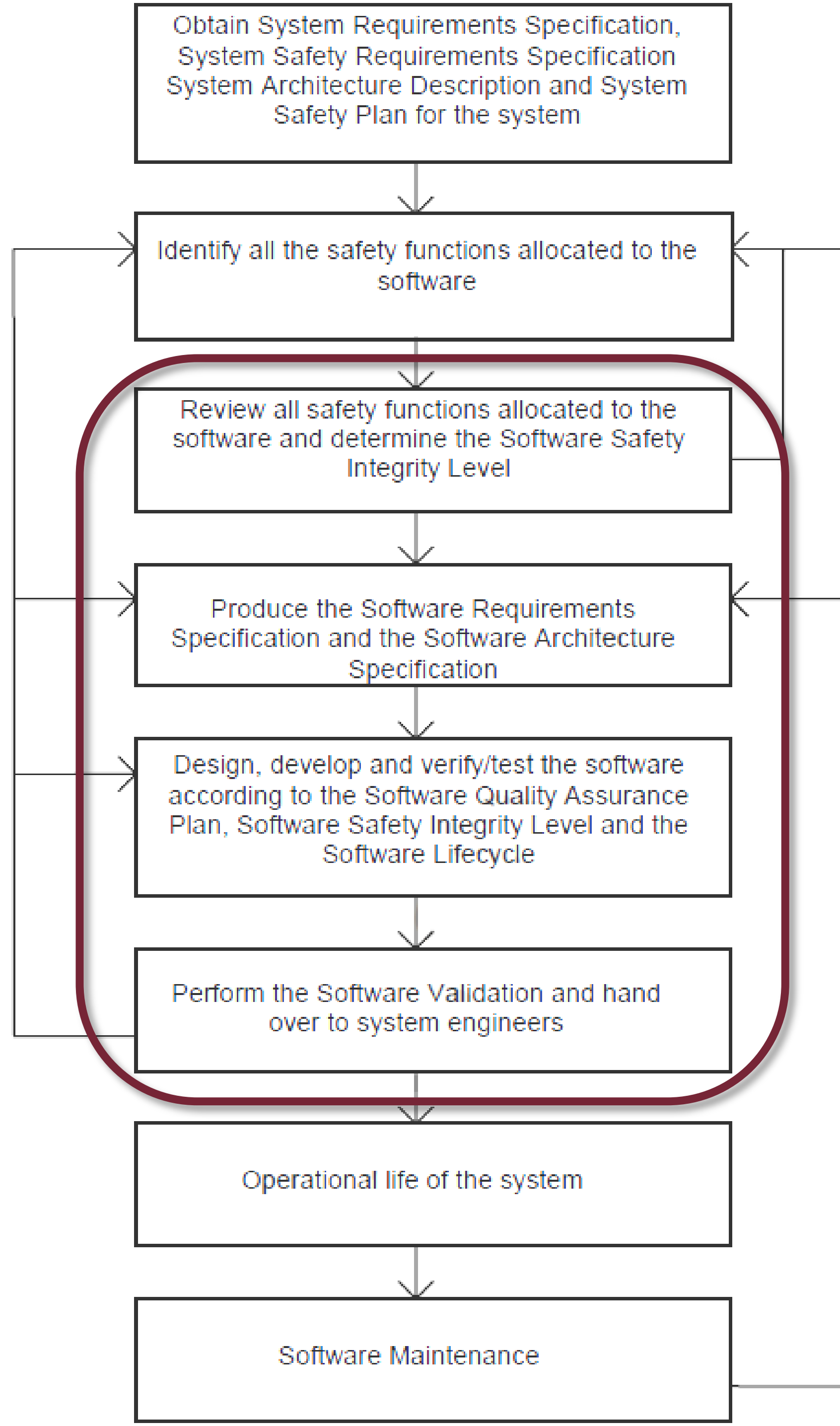


Railway control software as safety-critical software



Software route map

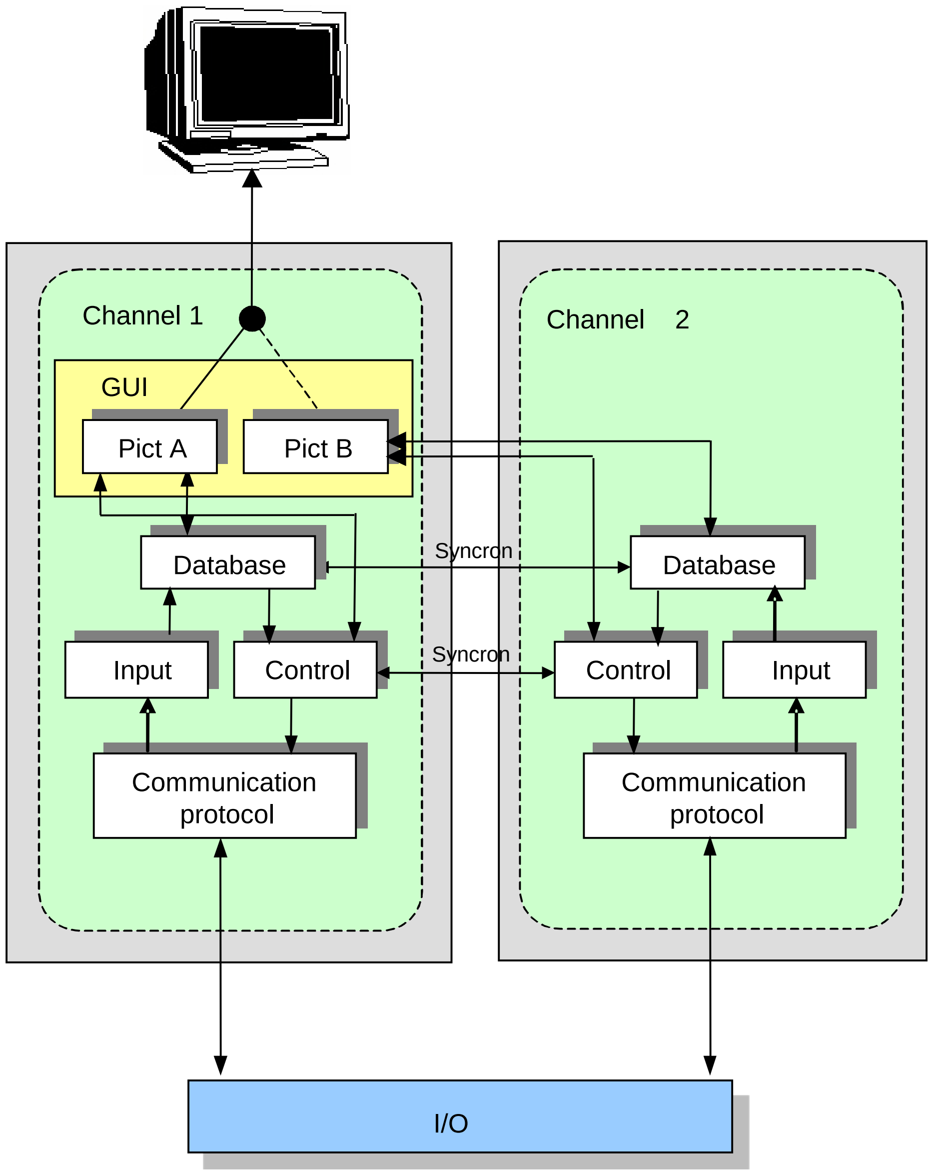
- Basic SIL concepts:
  - Software SIL shall be identical to the system SIL
  - Exception: Software SIL can be reduced if mechanism exists to prevent the failure of a software component from causing the system to go to an unsafe state
- Reducing software SIL requires:
  - Analysis of failure modes and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

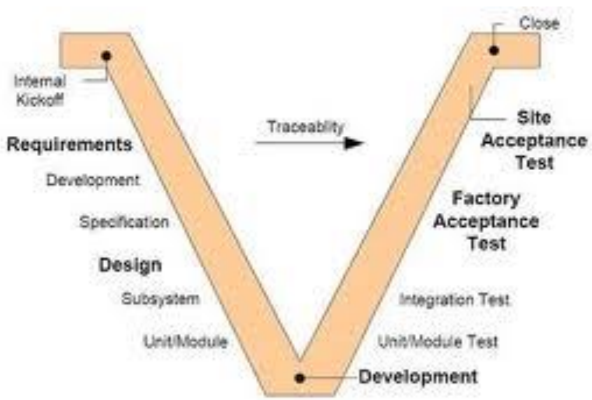
Problems in demonstrating software SIL

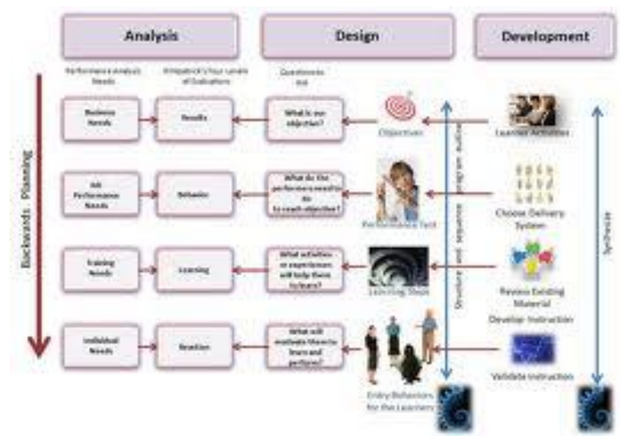
- Systematic failures in complex software:
  - Development of fault-free software cannot be guaranteed in case of complex functions
    - Goal: Reducing the number of faults that may cause hazard
  - Target failure measure (hazard rate) cannot be demonstrated by a quantitative analysis
    - General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

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

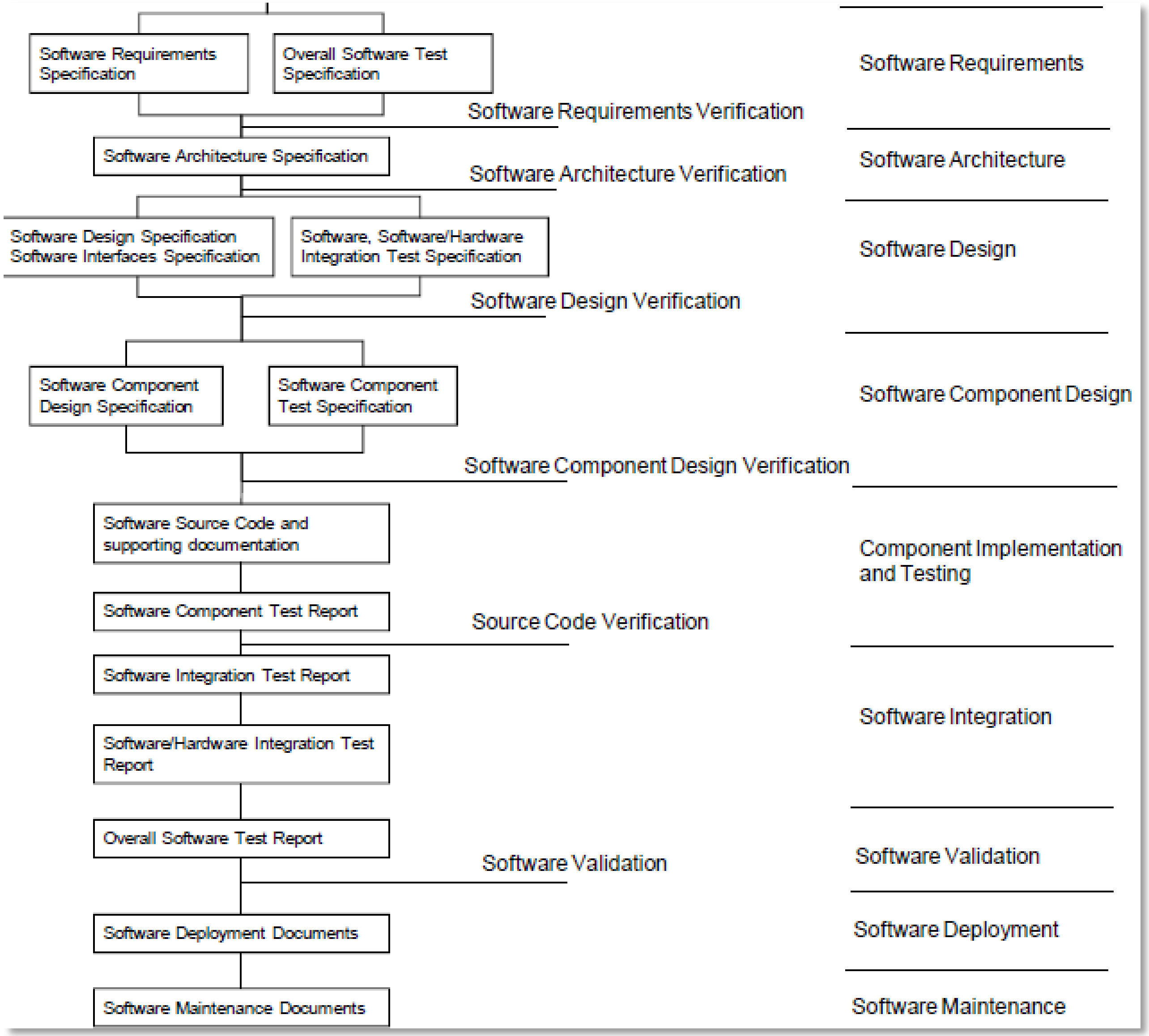
1. Safety lifecycle
2. Competence and independence of personnel
3. Techniques and measures in all phases of the lifecycle
4. Documentation

Safety lifecycle





Software lifecycle



Basic principles:

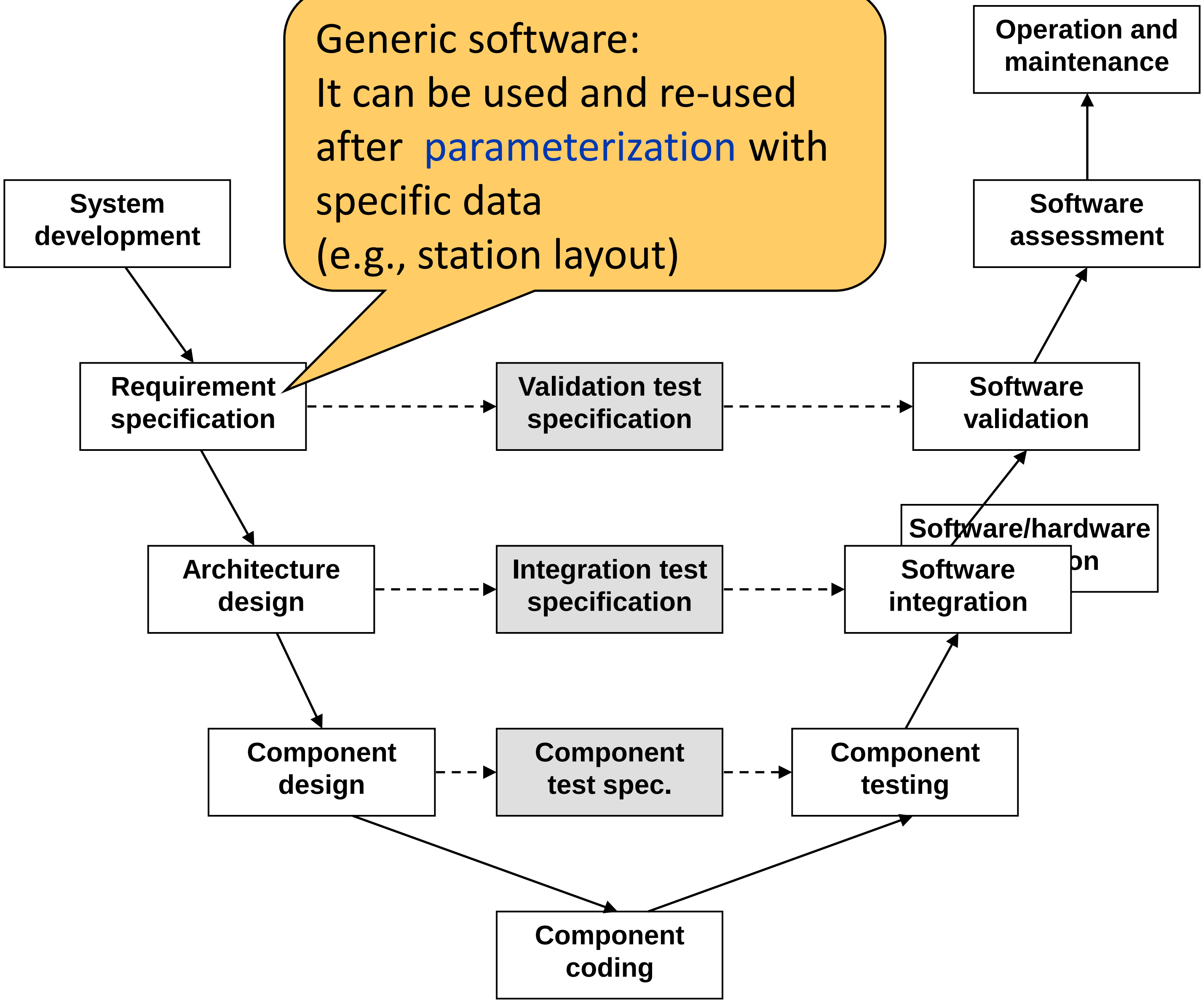
- Top-down design
- Modularity
- Preparing test specifications together with the design specification
- Verification of each phase
- Validation
- Configuration management and change control
- Clear documentation and traceability

Software quality assurance

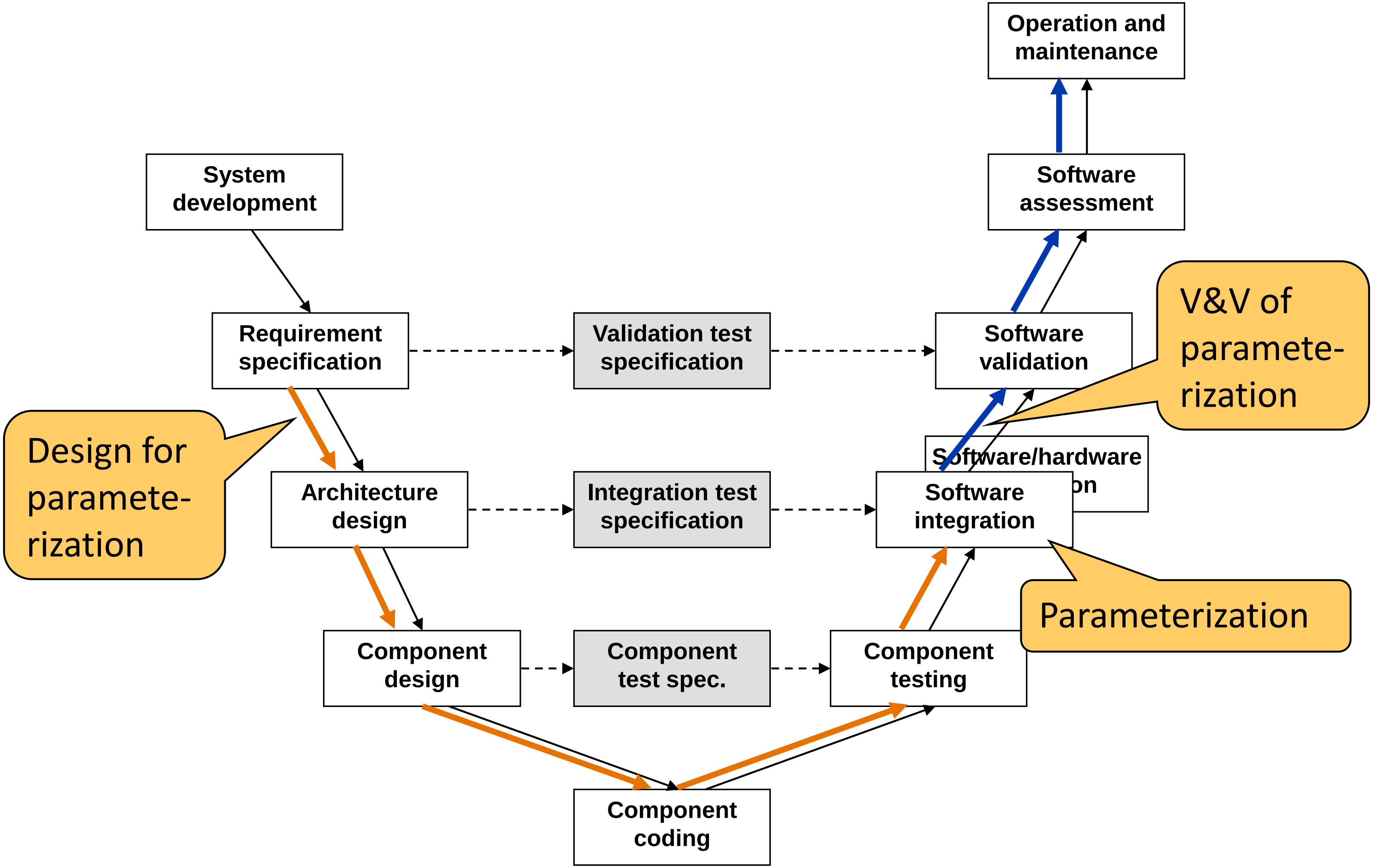
- Software Quality Assurance Plan
  - Determining all technical and control activities in the lifecycle
    - Activities, inputs and outputs (esp. verification and validation)
    - Quantitative quality metrics
    - Specification of its own updating (frequency, responsibility, methods)
  - Control of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

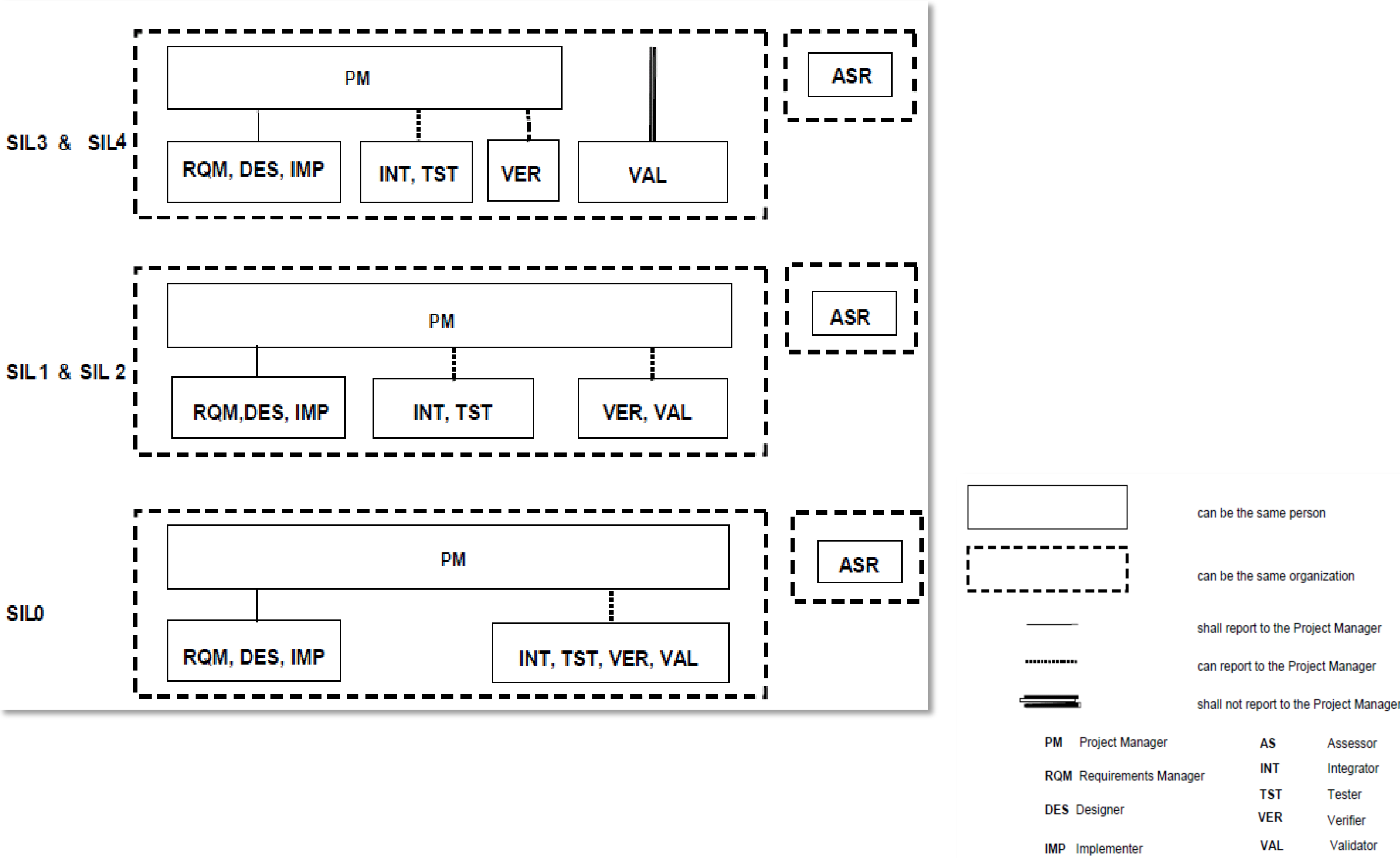


Roles and competences in the lifecycle

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

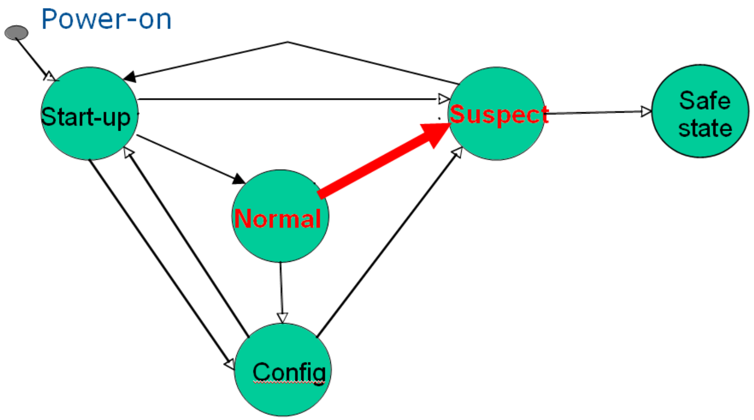
The preferred organizational structure



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

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

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."

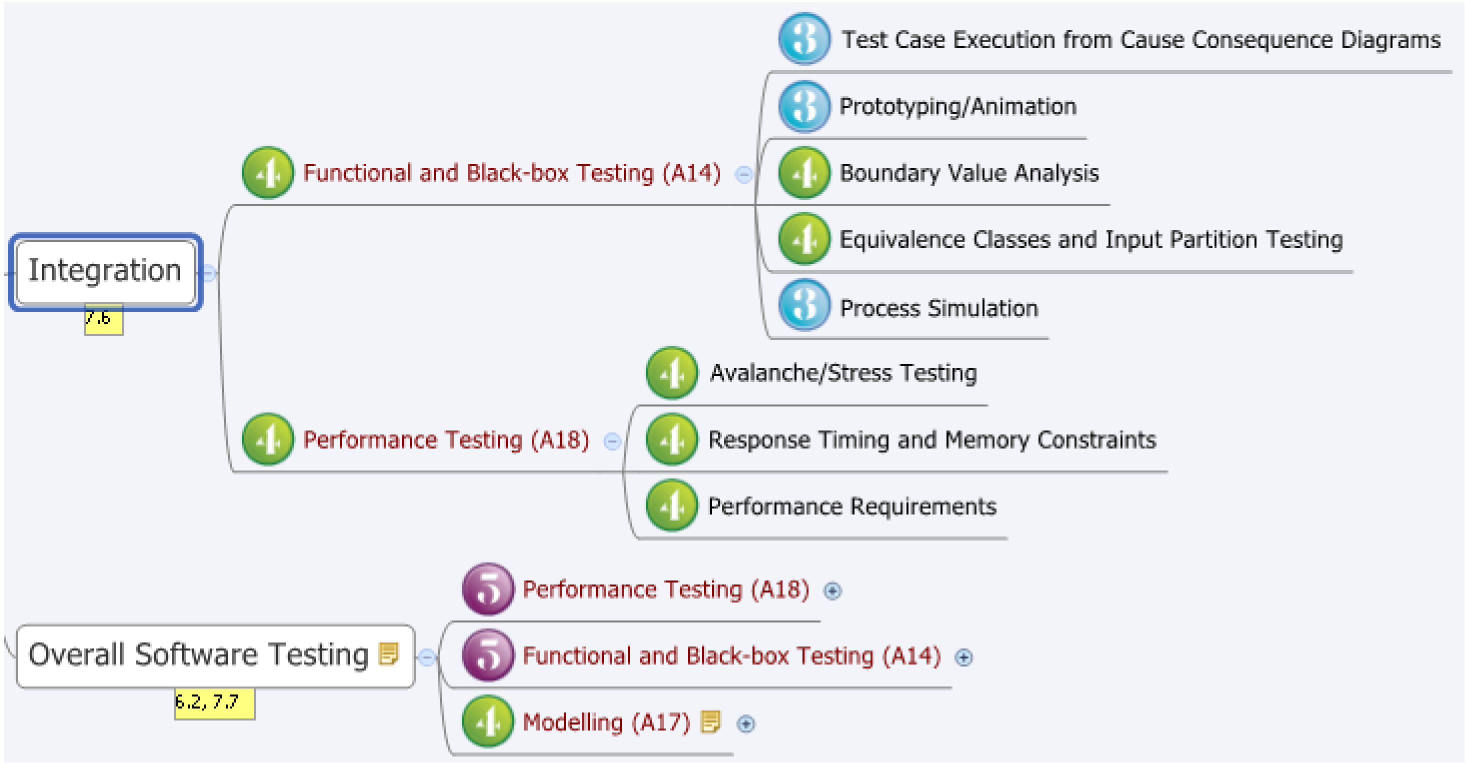
TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Defensive Programming	D.14	-	HR	HR	HR	HR
2. Fault Detection & Diagnosis	D.26	-	R	R	HR	HR
3. Error Correcting Codes	D.19	-	-	-	-	-
4. Error Detecting Codes	D.19	-	R	R	HR	HR
5. Failure Assertion Programming	D.24	-	R	R	HR	HR
6. Safety Bag Techniques	D.47	-	R	R	R	R
7. Diverse Programming	D.16	-	R	R	HR	HR
8. Recovery Block	D.44	-	R	R	R	R
9. Backward Recovery	D.5	-	NR	NR	NR	NR
10. Forward Recovery	D.30	-	NR	NR	NR	NR
11. Retry Fault Recovery Mechanisms	D.46	-	R	R	R	R
12. Memorising Executed Cases	D.36	-	R	R	HR	HR
13. Artificial Intelligence – Fault Correction	D.1	-	NR	NR	NR	NR
14. Dynamic Reconfiguration of software	D.17	-	NR	NR	NR	NR
15. Software Error Effect Analysis	D.25	-	R	R	HR	HR
16. Graceful Degradation	D.31	-	R	R	HR	HR
17. Information Hiding	D.33	-	-	-	-	-
18. Information Encapsulation	D.33	R	HR	HR	HR	HR
19. Fully Defined Interface	D.38	HR	HR	HR	M	M
20. Formal Methods	D.28	-	R	R	HR	HR
21. Modelling	Table A.17	R	R	R	HR	HR
22. Structured Methodology	D.52	R	HR	HR	HR	HR
23. Modelling supported by computer aided design and specification tools	Table A.17	R	R	R	HR	HR

Example: Verification and Testing

Requirements for SIL4:

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

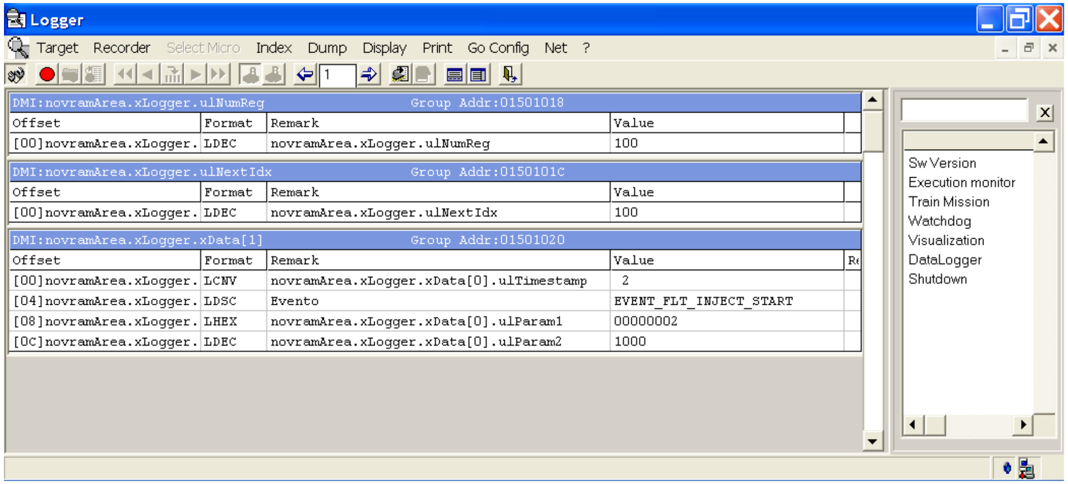
Example: Integration and Overall SW Testing



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

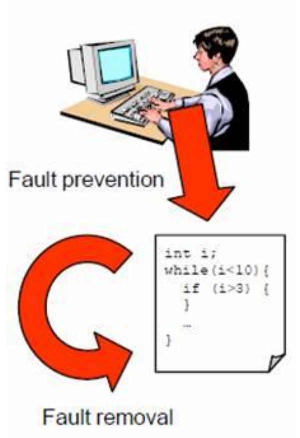
- T1: Generates outputs which cannot contribute to the executable code (and data) of the software
  - E.g.: a text editor, a requirement support tool, a configuration control tool

- 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

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

- 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

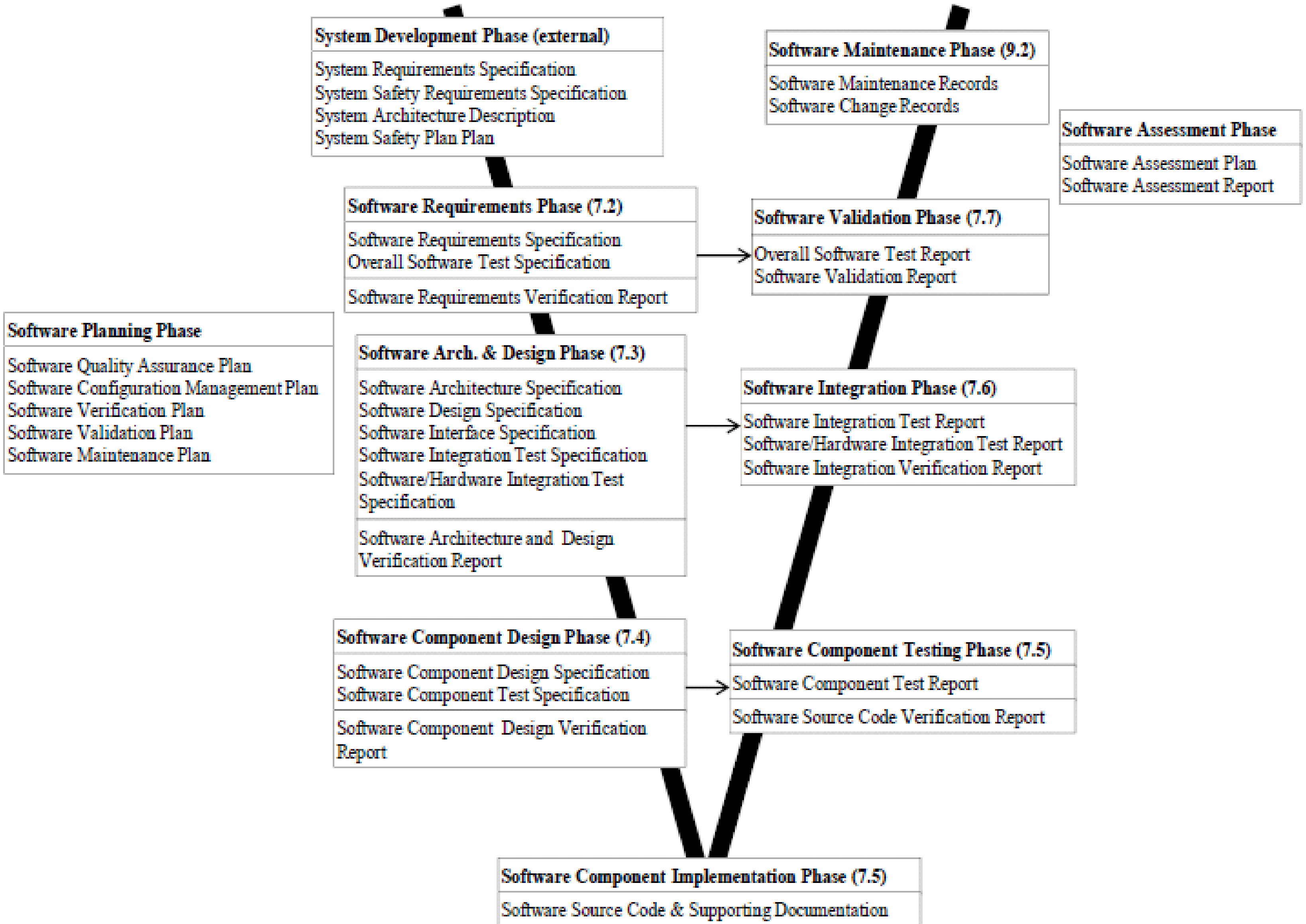
  → Precise and complete description for the system integrator
- The pre-existing software shall be included in the validation process of the whole software
- For SIL 3 or SIL 4 the following precautions shall be taken:
  - analysis of its possible failures and their consequences
  - a strategy to detect 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



Doc. control

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

PHASE	DOCUMENTATION	Written by	1st check	2nd check
Planning	1. Software Quality Assurance Plan	[a]	VER	VAL
	2. Software Quality Assurance Verification Report	VER		VAL
	3. Software Configuration Management Plan	see B.10	VER	VAL
	4. Software Verification Plan	VER		VAL
	5. Software Validation Plan	VAL	VER	
Software requirements	6. Software Requirements Specification	REQ	VER	VAL
	7. Overall Software Test Specification	TST	VER	VAL
	8. Software Requirements Verification Report	VER		VAL
Architecture and design	9. Software Architecture Specification	DES	VER	VAL
	10. Software Design Specification	DES	VER	VAL
	11. Software Interface Specifications	DES	VER	VAL
	12. Software Integration Test Specification	INT	VER	VAL
	13. Software/Hardware Integration Test Specification	INT	VER	VAL
	14. Software Architecture and Design Verification Report	VER		VAL
Component design	15. Software Component Design Specification	DES	VER	VAL
	16. Software Component Test Specification	TST	VER	VAL
	17. Software Component Design Verification Report	VER		
Component implementation and testing	18. Software Source Code and Supporting Documentation	IMP	VER	VAL
	19. Software Source Code Verification Report	VER		VAL
	20. Software Component Test Report	TST	VER	VAL
Integration	21. Software Integration Test Report	INT	VER	VAL
	22. Software/Hardware Integration Test Report	INT	VER	VAL
	23. Software Integration Verification Report	VER		
Overall software testing / Final validation	24. Overall Software Test Report	TST	VER	VAL
	25. Software Validation Report	VAL	VER	
	26. Tools Validation Report	[a]	VER	
	27. Release Note	[a]	VER	VAL

Case study: SAFEDMI

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

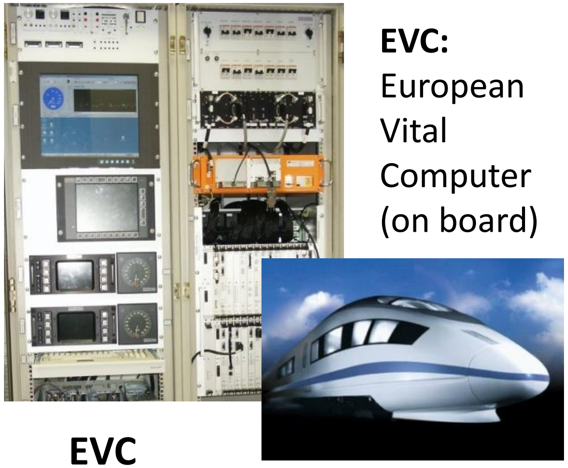
What is ERTMS?

- 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

Train driver

DMI



EVC

EVC: European Vital Computer (on board)

Maintenance centre

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} <= THR < 10^{-6} hazardous failures per hours
  - CENELEC standards: EN 50129 and EN 50128
- Reliability:
  - Mean Time To Failure: MTTF > 5000 hours (5000 hours: ~ 7 months)
- Availability:
  - A = MTTF / (MTTF+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

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

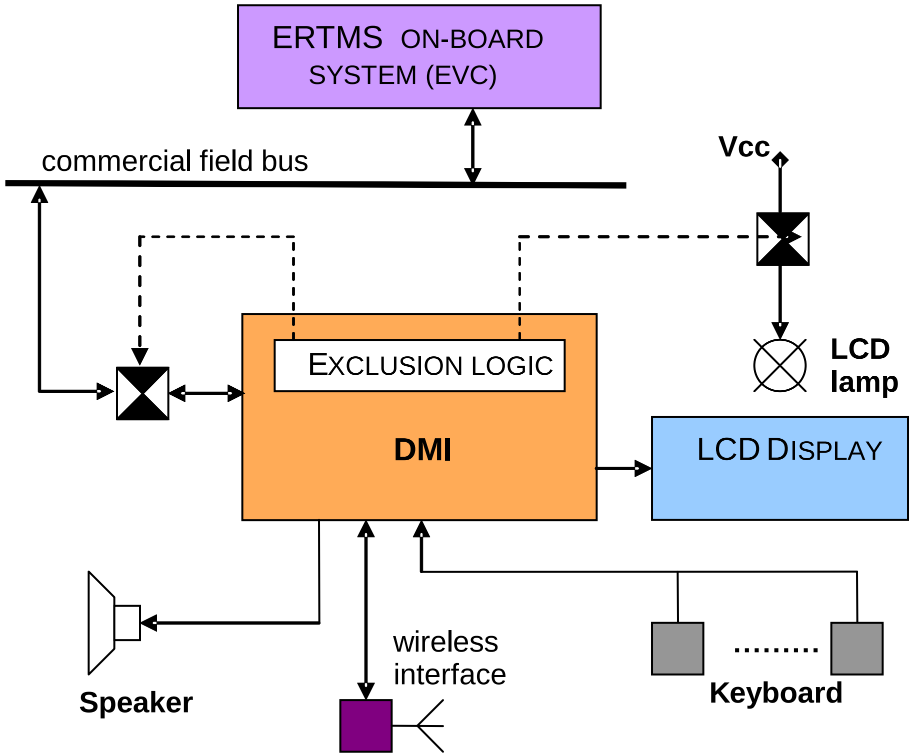
- Each function is equipped with an independent error detection
- The effects of detected errors can be handled

Inherent fail-safety

- All failure modes are safe
- „Inherent safe" system

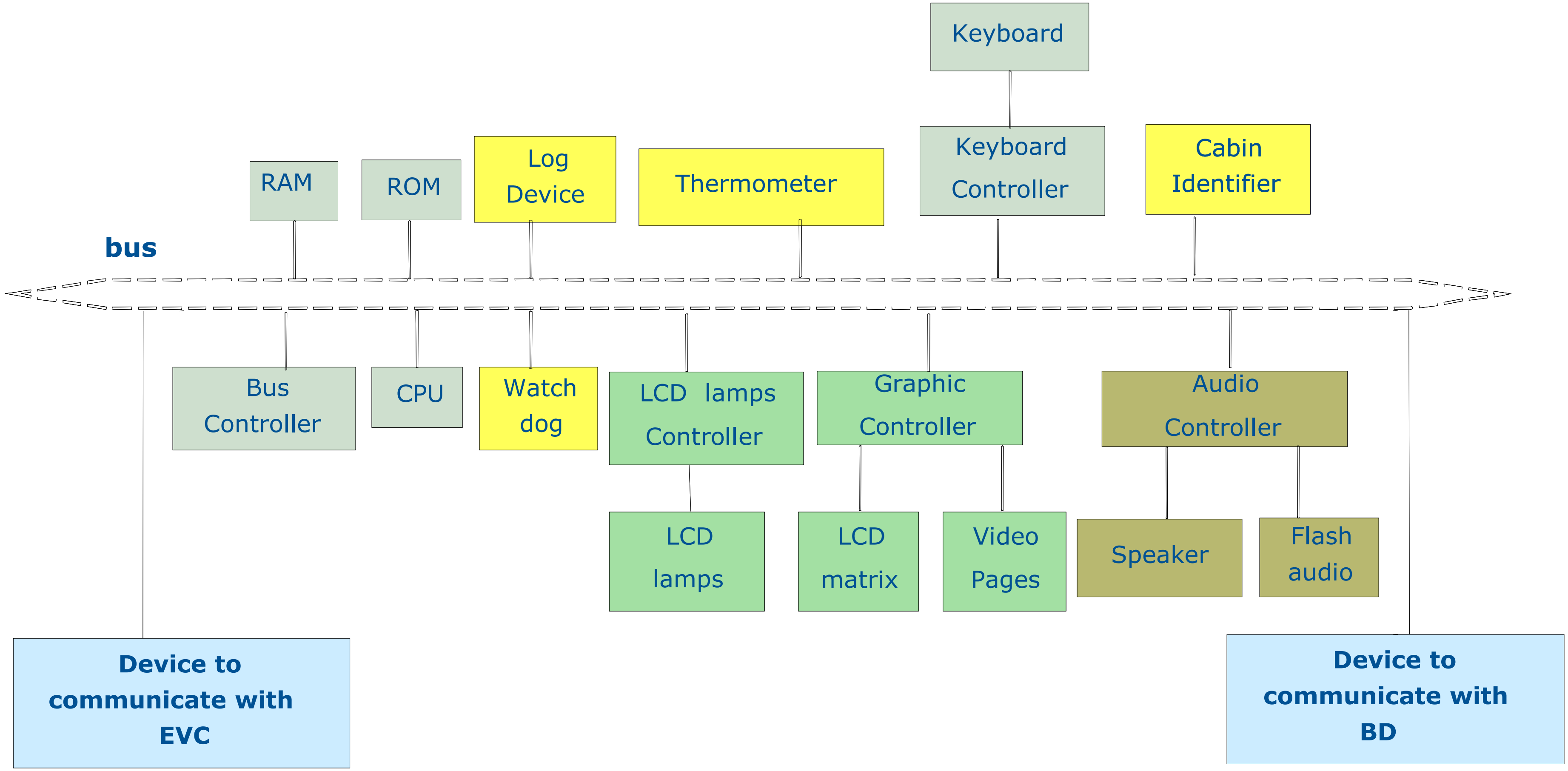
The SAFEDMI hardware concept

- 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



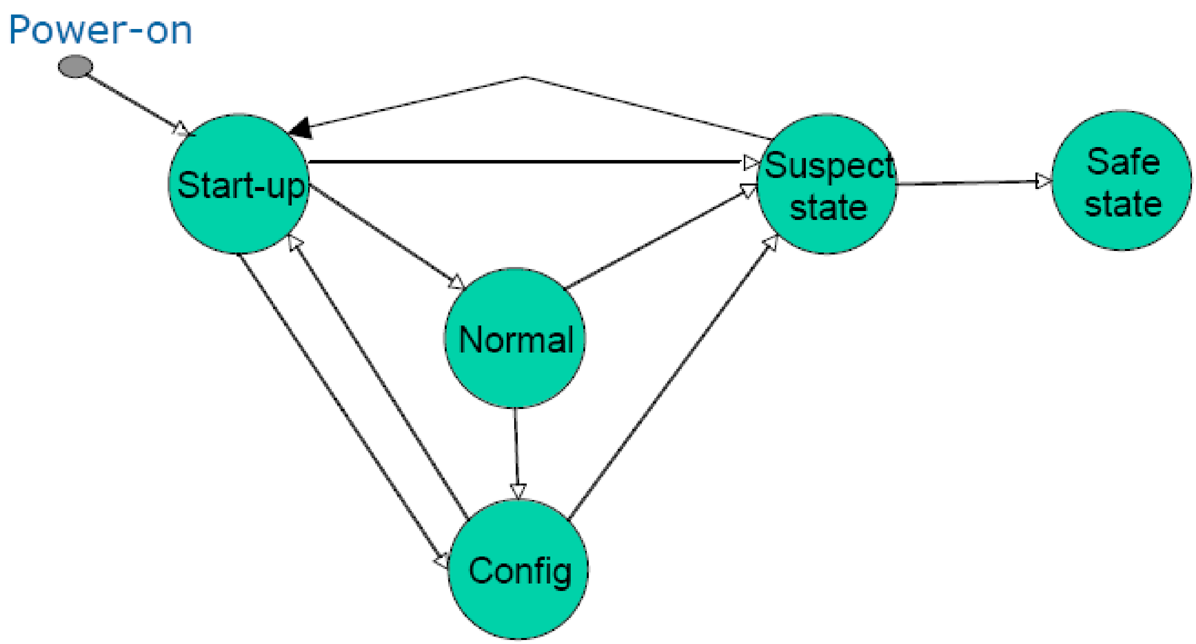
The SAFEDMI hardware architecture

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 → 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

Testing goals

Driver

DMI

EVC

EVC:
European
Vital
Computer
(on board)

Maintenance centre

Main test groups:

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

Testing the ERTMS functions

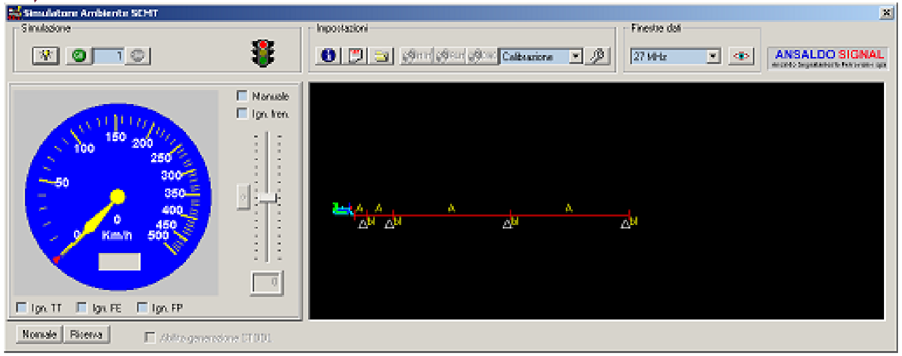
- Sequences of test inputs: DMI inputs + workload
- Test output: DMI display + Diagnostic device

Step	Action	Expected Event
1.	**Driver:** give traction to the train	**SAFEDMI:** the current train speed increases.
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.

Test environment

Track simulator

ERTMS OnBoard equipment

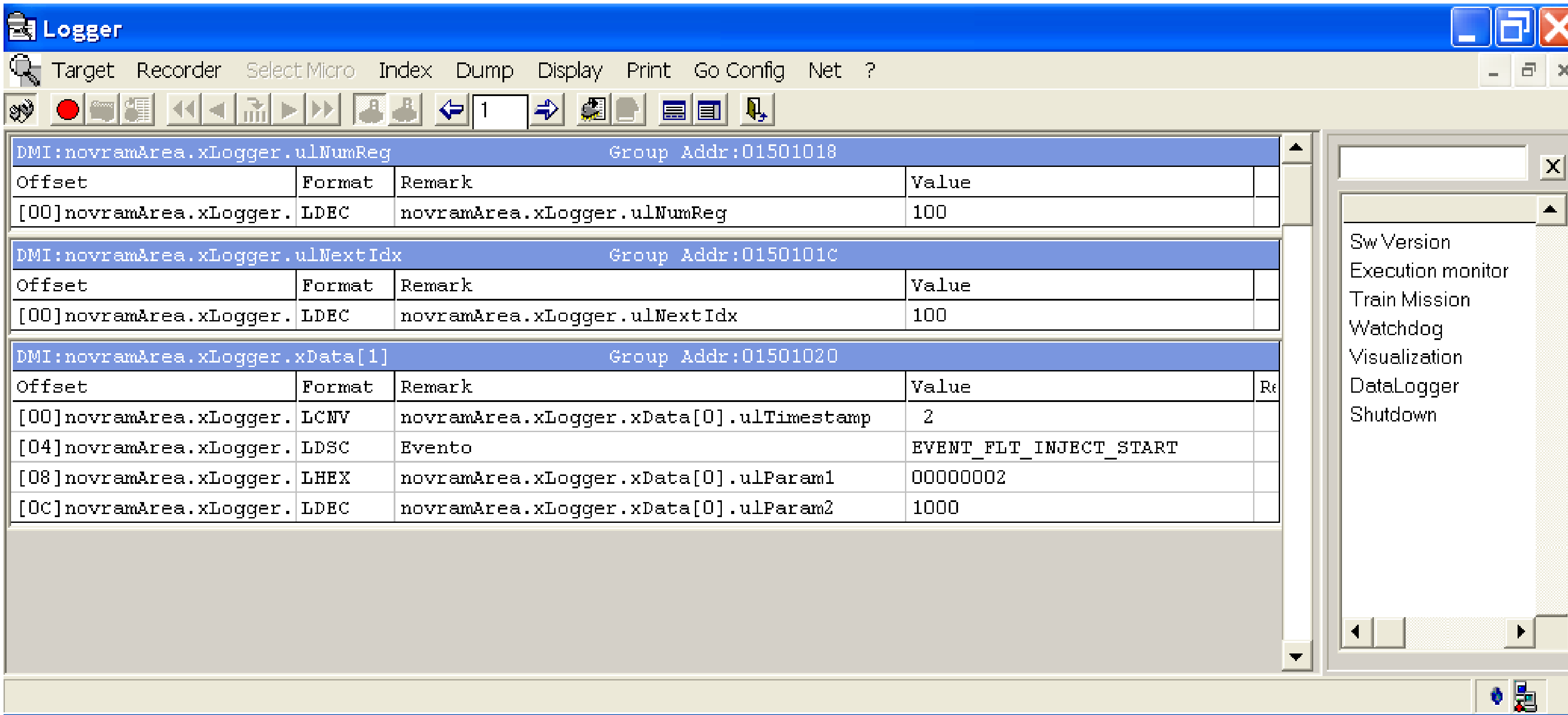


Simulating the workload:

- signals from balises on a given route
- control messages from the railway regulation control center

Plus: Diagnostic device

Output of the diagnostic device



Robustness testing

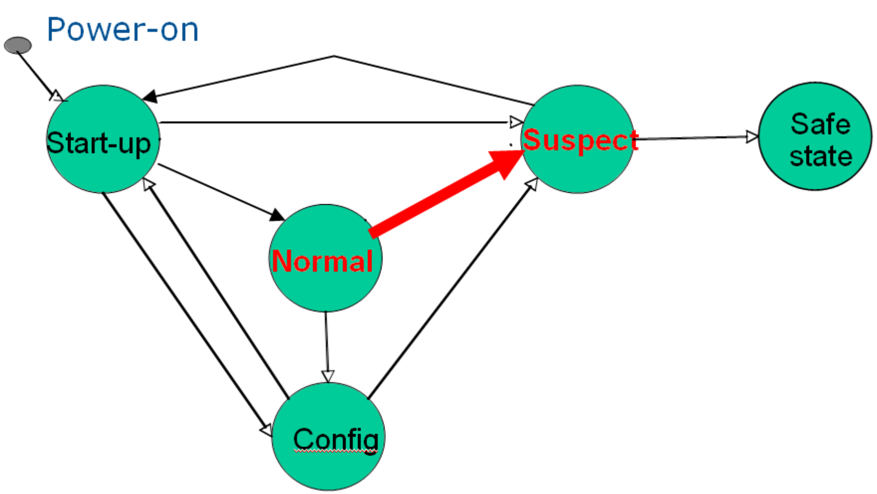
Driver DMI EVC

- 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, …

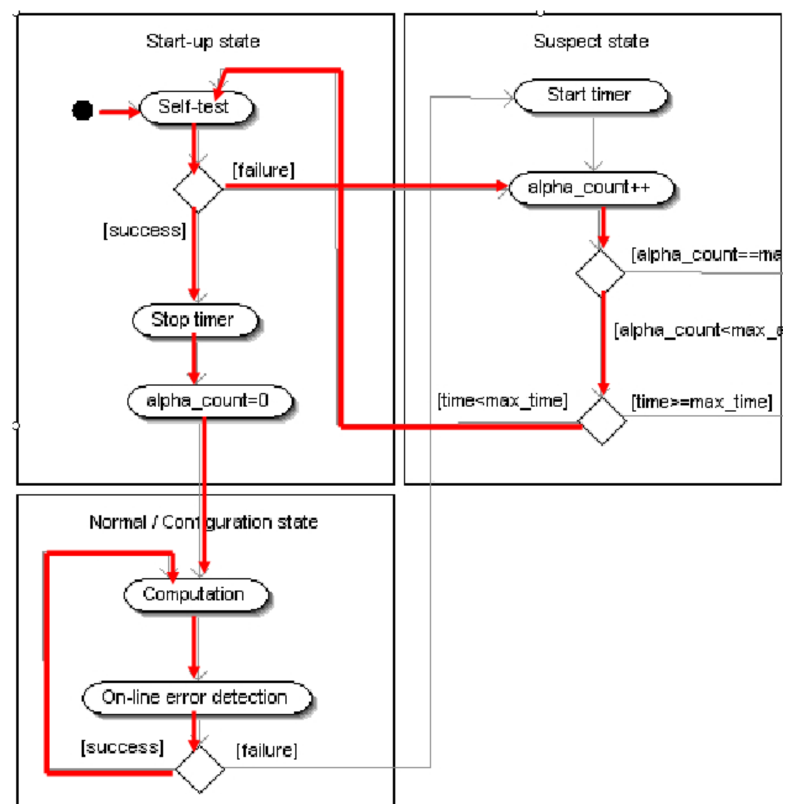
Testing the internal mechanisms

- 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)

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



State machine of the operational modes

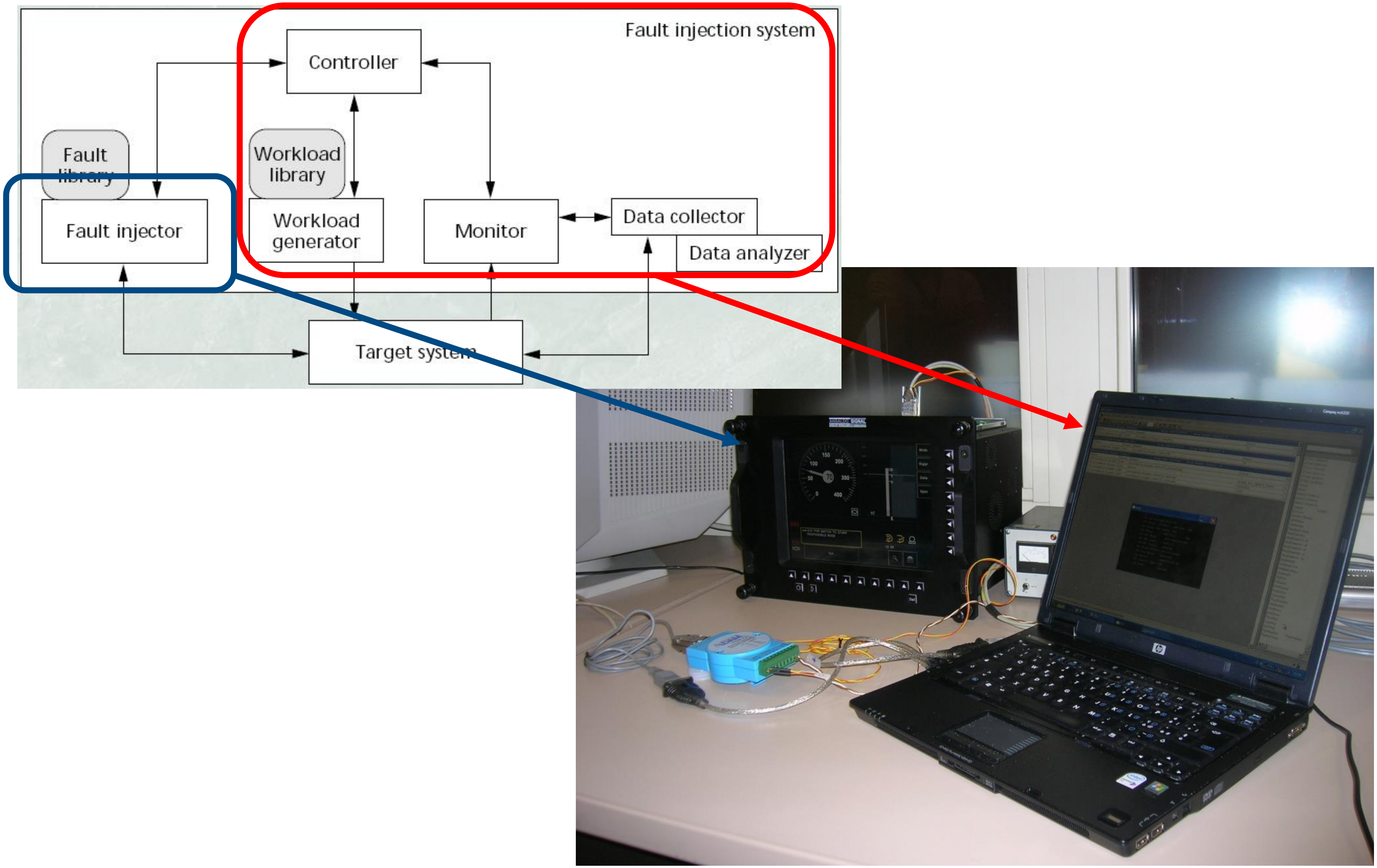


State machine of error counting

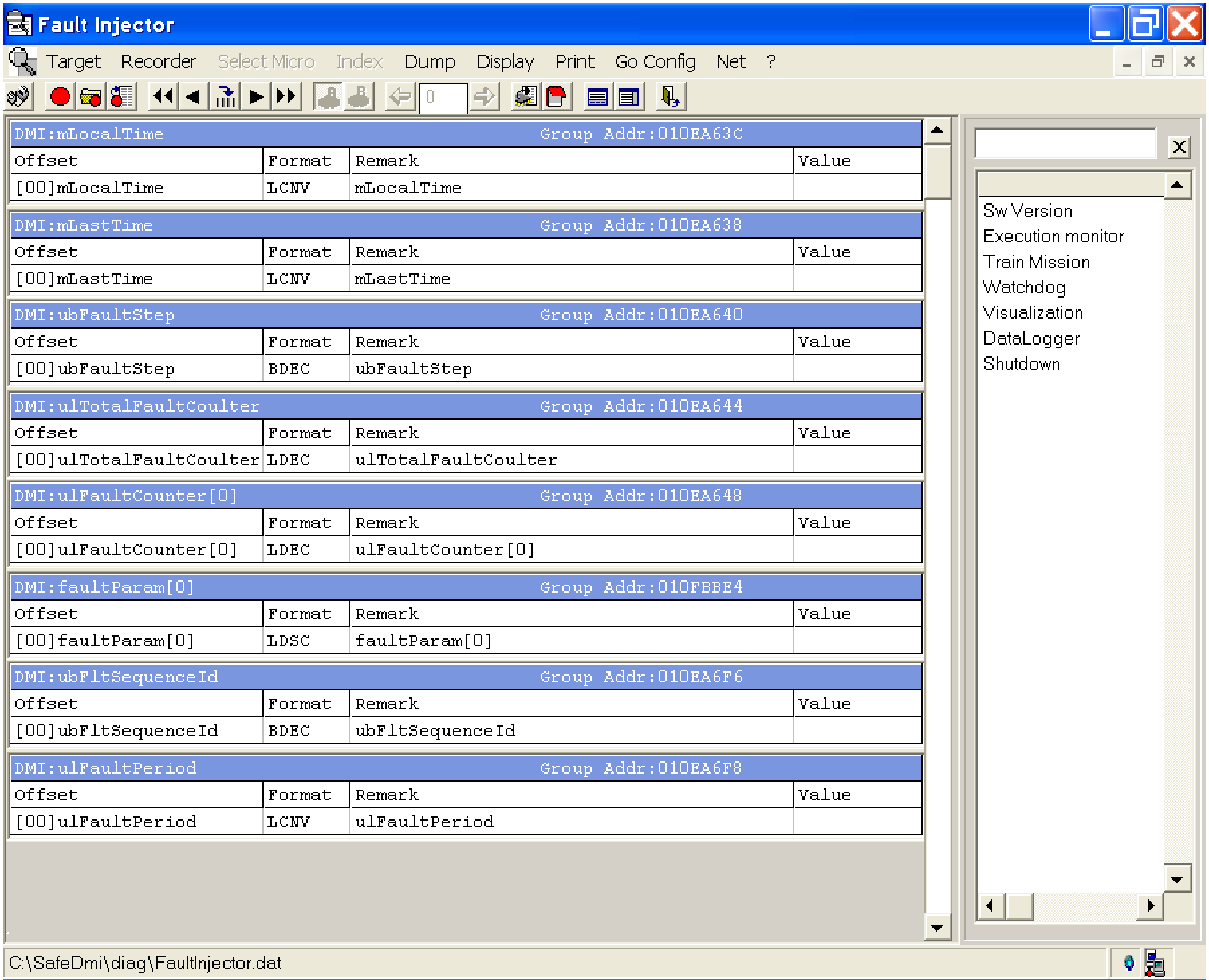
Testing the internal safety functions

- **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

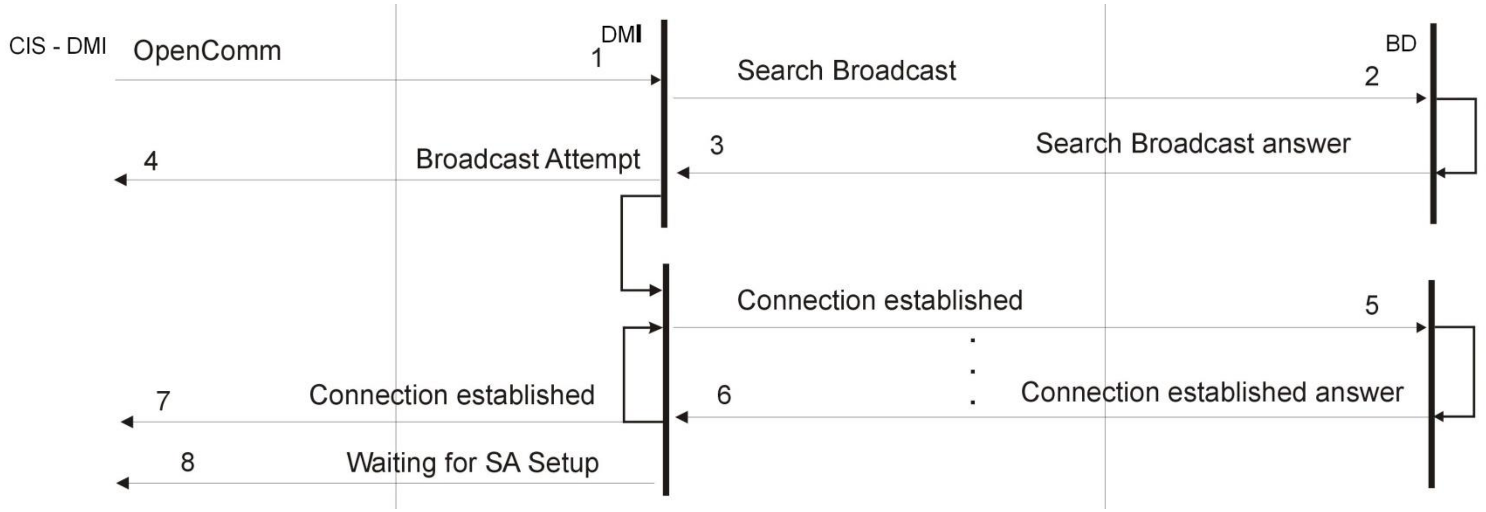


Collecting diagnostic data

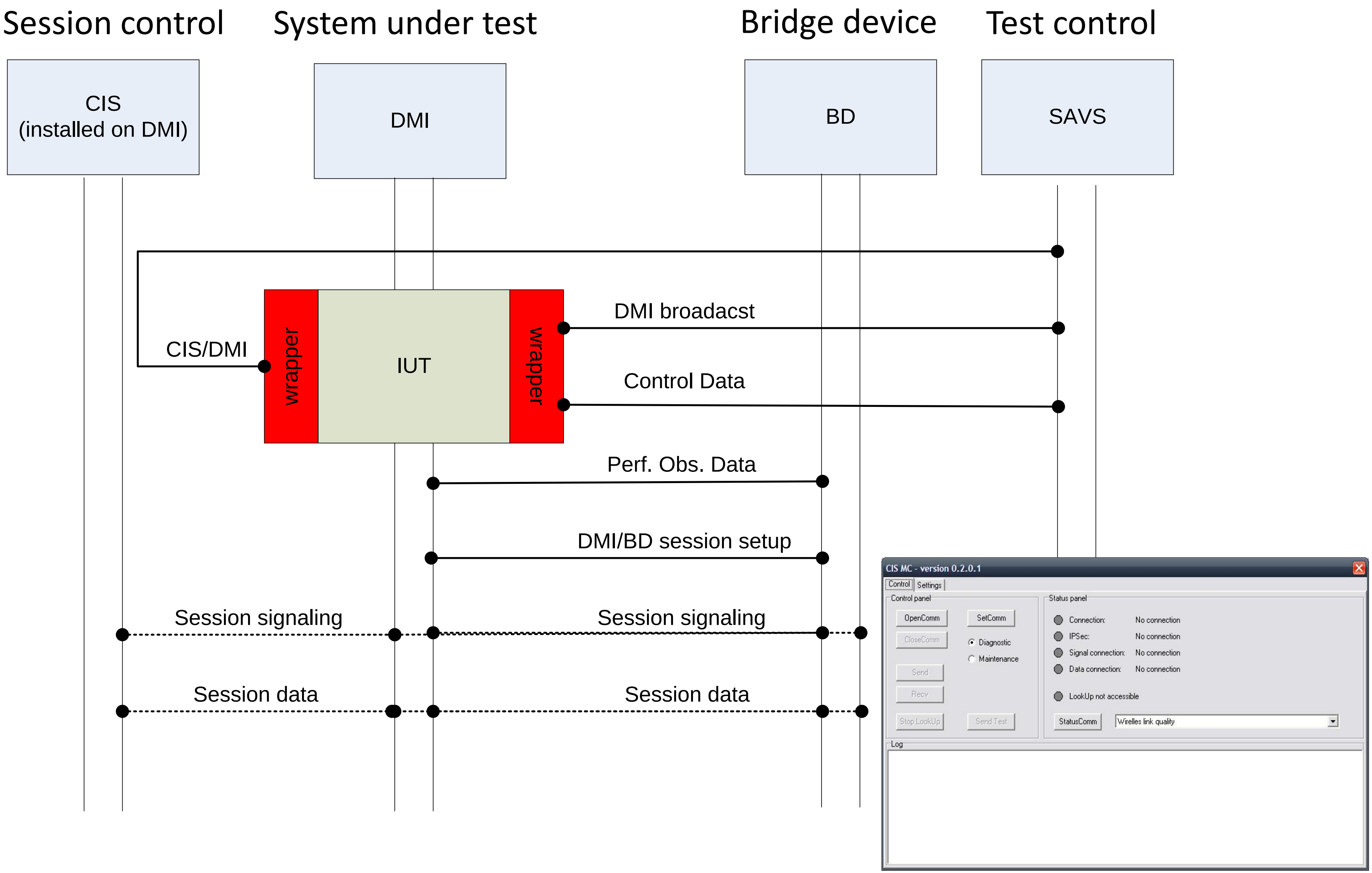


Testing the wireless communication

- 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



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