# Project „Safety Function for Temperature Monitoring System “

# Software architecture specification

**Modification history:**

| Version | Date | Modification | Creator | Auditor |
|---|---|---|---|---|
| 1.0 | 2020-05-27 | First version of architecture specification | SaS | DS |
| 1.1 | 2021-04-06 | Formal modifications | SaS | DS |
| 1.2 | 2023-05-07 | SW Architecture Specification for Function for Temperature Monitoring System | Kaushik | |
| | | | | |

# Content

1.0 Software architecture specification ..... 3
1.1 Initial situation ..... 3
1.2 Specification of the software architecture ..... 3
1.2.1 Fault detection [SW-AS1] ..... 3
1.2.2 Fault detection codes [SW-AS2] ..... 3
1.2.3 Assertion programming [SW-AS3] ..... 4
1.2.4 Modular approach [SW-AS4] ..... 4
1.2.5 Use of existing software elements [SW-AS5] ..... 4
1.2.6 Resources ..... 5
1.3 Specification of the tools and the programming language ..... 6
1.3.1 Programming language [SW-AS9] ..... 6
1.3.2 Certificated tools and compiler [SW-AS10] ..... 6
2.0 Verification of the software architecture specification ..... 8
2.1 Spezifikation der Softwarearchitektur ..... 8
3.0 Appendix ..... 9
4.0 Note ..... 10

# 1.0 Software architecture specification

## 1.1 Initial situation

A safety-relevant software function for safe temperature monitoring is to be developed. In addition, some auxiliary functions for the conversion of the units and the output of the temperature have to be programmed.

## 1.2 Specification of the software architecture

### 1.2.1 Fault detection [SW-AS1]

The fault detection procedure is recommended for SIL 2 and especially for SIL 3 or higher. If the procedure is not implemented for SIL 3 or higher, a sufficient justification must be described.

- Do physical quantities influence the program? If yes, these must be listed including their limits.
  - Yes
    - The temperature values read from the two temperature sensors
    - Limits are defined by the customer
      - E.g. -10°C to +45° (14°F to 113°F)

- Are functional values/variables applied in the program, that are subject to any limits or restrictions? If yes, these must be listed including their limit/restrictions.
  - Yes
    - The return values of **monitorTemp** function
      - 7 – temperature range OK
      - 5 – function error
      - 3 – temperature range limits exceeded, alarm

- Do external variables/values (e.g. currency) influence the program? If yes, these must be listed including their limit values.
  - Yes
    - TempS1 - Limits are defined by the customer
    - TempS2 - Limits are defined by the customer

- Reason for not implementing:

### 1.2.2 Fault detection codes [SW-AS2]

The procedure is recommended for SIL 1 or higher, especially for SIL 4. If the procedure is not implemented for SIL 4, a plausible justification must be given.

- False data shall not be processed and discarded in safety-related systems (an error message shall be displayed, if possible).

- List the input data including limits:

- List the output data including limit values:

- Reason for not implementing:

    - The integrity of the sensor values (i.e., TempS1 and TempS2) are assumed to be guaranteed since they were stored in the flash memory after safe parameterization and a checksum (CRC) was calculated.

### 1.2.3 Assertion programming [SW-AS3]

The procedure is recommended for SIL 1 or higher, especially for SIL 4. If the procedure is not implemented for SIL 4, a plausible justification must be given.

- A function or module must be checked for valid precondition and postcondition.
- Precondition of the safety function:

- Postcondition of the safety function:

### 1.2.4 Modular approach [SW-AS4]

The procedure is recommended for SIL 1 or higher. If the procedure is not implemented for SIL 1 or higher levels, a plausible justification must be given.

- Subdivision of the program description (based on the SRS) into individual functions.
    - monitorTemp
    - checkTemp
    - displayTemp
    - calcC2F
    - calcF2C

- For different operating modes: assign functions to operating modes

- Show temporal relationships of functions for each operating mode (e.g. by use of Petri net)

### 1.2.5 Use of existing software elements [SW-AS5]

The procedure is recommended for SIL 1 or higher, especially for all levels starting at SIL 2. If the procedure is not implemented for SIL 2 or higher, a plausible justification must be given.

- If existing software is used, evidence must be provided and documented for each software element.

- The design of the software element is known and documented (attach documentation file)

- The design was verified and validated at each stage of development (attach documentation file)

- Unnecessary functions of the element do not affect the requirements of the safety function (verify and attach documentation)

- All realistic fault mechanisms of the software element have been discovered for the new system and have been mitigated by appropriate measures (verify and attach documentation)

### 1.2.6 Resources

- Does the program have to be updateable?

[SW-AS6] Yes. The program has to be extendable for implementing future requirements

- Do there exist any requirements regarding the response time?

[SW-AS7] Yes. The maximum execution time of **monitorTemp** function shall be 5ms

- What is the maximum hardware resource which are required for the safety function?

[SW-AS8] Yes. The maximum execution time of **monitorTemp** function shall be 5ms

## 1.3 Specification of the tools and the programming language

### 1.3.1 Programming language [SW-AS9]

The programming language can be chosen according to IEC 61508-7 Table C.1. (see appendix)

Programming language: C

### 1.3.2 Certificated tools and compiler [SW-AS10]

- Which tools / compiler are used? Include short justification.
    - MinGW GCC compiler is used
    - Considering:
        - Support of the software properties
        - Clarity of useability and functionality
        - Correctness and repeatability of results

- Which classification do the applied tools have according to the IEC 61508-3 standard?
  T3

**T1 tools**

Do not generate output that can directly or indirectly contribute to the executable code (including data) of the safety-related system;

- Example: a text editor, a requirements or design description tool without automatic code generation capabilities and configuration management tools

**T2 tools**

Support testing or verification of the design or executable code, where deviations in the tool can lead to failure of fault detection, but cannot generate deviations directly in the executable software;

- Example: A test facility with test generator, a tool for measuring test coverage/code coverage and a tool for static analysis.
- Specification of the tool
- Product documentation
    - Tool name and its version
    - Describes behavior, user manual and restriction of use
- When using a tool, the following must be determined in advance:
    - Tool name and its version
    - Designation of the basic configuration of the applied tool version
    - Type of use and the accompanying configuration parameters
- Perform assessment of the tools
    - Define a „degree of trust”
    - Potential fault mechanisms
    - Taking fault limitation measures
- The correct collaboration of several tools must be ensured.
- Evaluation of problems/errors which are detected afterwards e.g. in the community/field application resp. in any FS application

**T3 tools**

Generate output that can directly or indirectly contribute to the executable code (including data) of the safety-related system.

- Example: an optimizing compiler where the relationship between the source code and the generated object code is not obvious; a compiler which includes executable runtime packages in the executable code.
- Must fulfill all mentioned requirements for T2 and additional:
- Proof of compliance of the tool and the specification
    - May be based on field testing (long-term experience) in similar environment and application
    - Or by validation of a tool (documentation of results)
        - Chronological record of the validation activity
        - Version of the tool manual
        - Validated tool part, tool function(s) and tool configuration
        - Documentation of the results of all validation activity
        - Documentation of the test cases and the results for further analysis
        - Reproducibility and repeatability of test cases and results
    - If proof of compliance is not available:
        - Taking effective measures to control fault modes which may originate in the application of the tool

# 2.0 Verification of the software architecture specification

## 2.1 Spezifikation der Softwarearchitektur

- Who shall perform the verification?

  Company: CKN GmbH
  Name, first name: Halpert Jim
  expertise: Functional Safety Professional
  phone / e- mail: jhalpert@de.ckn.com

- Does the software architecture specification meet all safety requirements?
  Yes

- Is the specified integration test plan appropriate for verification of the software architecture specification?
  Yet to be checked

- Are there discrepancies between:
  - the software architecture specification and the SRS?
    No
  - the software architecture specification and its integration test plan?
    Yet to be checked
  - the integration test plan and the validation plan?
    Yet to be checked

# 3.0 Appendix

**Table C.1 – Recommendations for specific programming languages**

| | Programming language | SIL1 | SIL2 | SIL3 | SIL4 |
|---|---|---|---|---|---|
| 1 | ADA | HR | HR | R | R |
| 2 | ADA with subset | HR | HR | HR | HR |
| 3 | Java | NR | NR | NR | NR |
| 4 | Java with subset (including either no garbage collection or garbage collection which will not cause the application code to stop for a significant period of time). See Annex G for guidance on use of object oriented facilities. | R | R | NR | NR |
| 5 | PASCAL (see Note 1) | HR | HR | R | R |
| 6 | PASCAL with subset | HR | HR | HR | HR |
| 7 | FORTRAN 77 | R | R | R | R |
| 8 | FORTRAN 77 with subset | HR | HR | HR | HR |
| 9 | C | R | – | NR | NR |
| 10 | C with subset and coding standard, and use of static analysis tools | HR | HR | HR | HR |
| 11 | C++ (see Annex G for guidance on use of object oriented facilities) | R | – | NR | NR |
| 12 | C++ with subset and coding standard, and use of static analysis tools (see Annex G for guidance on use of object oriented facilities) | HR | HR | HR | HR |
| 13 | Assembler | R | R | – | – |
| 14 | Assembler with subset and coding standard | R | R | R | R |
| 15 | Ladder diagrams | R | R | R | R |
| 16 | Ladder diagram with defined subset of language | HR | HR | HR | HR |

Extract IEC 61508-7 table C.1

| | Programming language | SIL1 | SIL2 | SIL3 | SIL4 |
|---|---|---|---|---|---|
| 17 | Functional block diagram | R | R | R | R |
| 18 | Function block diagram with defined subset of language | HR | HR | HR | HR |
| 19 | Structured text | R | R | R | R |
| 20 | Structured text with defined subset of language | HR | HR | HR | HR |
| 21 | Sequential function chart | R | R | R | R |
| 22 | Sequential function chart with defined subset of language | HR | HR | HR | HR |
| 23 | Instruction list | R | – | NR | NR |
| 24 | Instruction list with defined subset of language | HR | R | R | R |

Extract IEC 61508-7 table C.1

# 4.0 Note

This template is **not considered complete, but applicable**. The template contains the requirements of IEC 61508 for the development of software functions. The template is to be regarded as alive, i.e. during the development process, possibly issues may be discovered, which are not contained in the template. These should be added to the template at the correct place, in order to be able to consider them during next use. The more diverse software is created by means of this template, the more completely and exactly can the development process for software be realized in the future.