SLOVENSKI STANDARD
oSIST prEN 50716:2022

01-april-2022

Standard za navzkrižno delujočo programsko opremo za železnice

Cross-functional Software Standard for Railways

Ta slovenski standard je istoveten z: **prEN 50716**

ICS:

35.080	Programska oprema	Software
35.240.60	Uporabniške rešitve IT v prometu	IT applications in transport
45.020	Železniška tehnika na splošno	Railway engineering in general

oSIST prEN 50716:2022 **en**

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

DRAFT
prEN 50716

January 2022

ICS 35.240.60

English Version

Cross-functional Software Standard for Railways

To be completed

To be completed

This draft European Standard is submitted to CENELEC members for enquiry.
Deadline for CENELEC: 2022-04-22.

It has been drawn up by CLC/TC 9X.

If this draft becomes a European Standard, CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

This draft European Standard was established by CENELEC in three official versions (English, French, German).
A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Warning : This document is not a European Standard. It is distributed for review and comments. It is subject to change without notice and shall not be referred to as a European Standard.

CENELEC

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

Contents

European foreword

This document (prEN 50716:2022) has been prepared by CLC/TC 9X "Electrical and electronic applications for railways".

This document is currently submitted to the Enquiry.

The following dates are proposed:

• latest date by which the existence of this document has to be announced at national level	(doa)	dor + 6 months
• latest date by which this document has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	dor + 12 months
• latest date by which the national standards conflicting with this document have to be withdrawn	(dow)	dor + 36 months (to be confirmed or modified when voting)

This document will supersede EN 50128:2011+A2:2020 and EN 50657:2017.

prEN 50716:2022 includes the following significant technical changes with respect to EN 50128:2011+A2:2020 and EN 50657:2017:

— [Will be completed within FprEN - see also guidance on changes provided with prEN]

— ….

This document is read in conjunction with EN 50126-1 "*Railway applications – The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS): Basic requirements and generic process*" [1] and EN 50126-2 "*Railway applications – The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS): Systems Approach to Safety*" [2].

This document has been prepared under a Standardization Request given to CENELEC by the European Commission and the European Free Trade Association, and supports essential requirements of EU Directive(s) / Regulation(s).

For relationship with EU Directive(s) / Regulation(s), see informative Annex ZZ, which is an integral part of this document.

Introduction

This document concentrates on the methods which need to be used in order to provide software which meets the demands for software integrity which are placed upon it by these wider considerations.

This document provides a set of requirements for the development, deployment and maintenance of any software intended for railway applications. It defines requirements concerning organizational structure, the relationship between organizations and division of responsibility involved in the development, deployment and maintenance activities. Criteria for the qualification and expertise of personnel are also provided in this document.

The key concept of this document is that of levels of software integrity. This document addresses five software integrity levels where basic integrity is the lowest and 4 the highest one. The higher the risk resulting from software failure, the higher the software integrity level will be.

NOTE 1 The concept of basic integrity used in this document was first introduced in the EN 50126 series ([1] [2]).

This document has identified techniques and measures for the five levels of software integrity. The required techniques and measures for basic integrity and for the safety integrity levels 1-4 are shown in the normative tables of Annex A. The required techniques for level 1 are the same as for level 2, and the required techniques for level 3 are the same as for level 4. This document does not give guidance on which level of software integrity is appropriate for a given risk. This decision will depend upon many factors including the nature of the application, the extent to which other systems carry out safety-related functions and social and economic factors.

It is within the scope of EN 50126-1 and EN 50126-2 to define the process of specifying the safety-related functions allocated to software.

This document specifies those measures necessary to achieve these requirements.

The EN 50126 series ([1] [2]) requires that a systematic approach is taken to:

a) identify hazards, assessing risks and arriving at decisions based on risk criteria,

b) identify the necessary risk reduction to meet the risk acceptance criteria,

c) define the overall system safety requirements for the safeguards necessary to achieve the required risk reduction,

d) select a suitable system architecture,

e) plan, monitor and control the technical and managerial activities necessary to translate the system safety requirements into a safety-related system of a validated safety integrity level.

As decomposition of the specification into a design comprising safety-related systems and components takes place, further allocation of safety integrity levels is performed. Ultimately this leads to the required software integrity levels.

The current state-of-the-art is such that neither the application of quality assurance methods (so-called fault avoiding measures and fault detecting measures) nor the application of software fault tolerant approaches can guarantee the absolute safety of the software. There is no known way to prove the absence of faults in reasonably complex safety-related software, especially the absence of specification and design faults.

The principles applied in developing high integrity software include, but are not restricted to

— top-down design methods,

— modularity,

— verification of each phase of the development lifecycle,

— verified components and component libraries,

— clear documentation and traceability,

— auditable documents,

— validation,

— assessment,

— configuration management and change control and

— appropriate consideration of organization and personnel competency issues.

At the system level, the allocation of system requirements to software functions takes place. This includes the definition of the required software integrity level for the functions. The successive functional steps in the application of this document are shown in Figure 1 and are as follows:

a) define the Software Requirements Specification and in parallel consider the software architecture. The software architecture is where the safety strategy is developed for the software and the software integrity level (7.2 and 7.3);

b) design, develop and test the software according to the Software Quality Assurance Plan, software integrity level and the software lifecycle (7.4 and 7.5);

c) integrate the software on the target hardware and verify functionality (7.6);

d) accept and deploy the software (7.7 and 9.1);

e) software maintenance, if required during operational life (9.2).

A number of activities run across the software development. These include testing (6.1), verification (6.2), validation (6.3), assessment (6.4), quality assurance (6.5) and modification and change control (6.6).

Requirements are given for support tools (6.7) and for systems which are configured by application data or algorithms (Clause 8).

Requirements are also given for the independence of roles and the competence of staff involved in software development (5.1, 5.2 and Annex B).

This document does not mandate the use of a particular software development lifecycle. However, illustrative lifecycle and documentation sets are given in 5.3, 7.1, and in C.1

Tables have been formulated ranking various techniques/measures against the software integrity levels 1-4 and for basic integrity. The tables are in Annex A. Cross-referenced to the tables is a bibliography giving a brief description of each technique/measure with references to further sources of information. The bibliography of techniques is in Annex D.

This document does not specify the requirements for the development, implementation, maintenance and/or operation of security policies or security services needed to meet security requirements that may be needed by the safety-related system. IT security can affect not only the operation but also the functional safety of a system. For IT security, appropriate IT security standards should be applied.

NOTE 2 IEC/ISO standards (or series of standards) that address IT security in depth are [3], [4] and [5].

It may be necessary to balance between measures against systematic errors and measures against security threats. An example is the need for fast security updates of software arising from security threats, whereas if such software is safety related, it should be thoroughly developed, tested, validated and approved before any update.

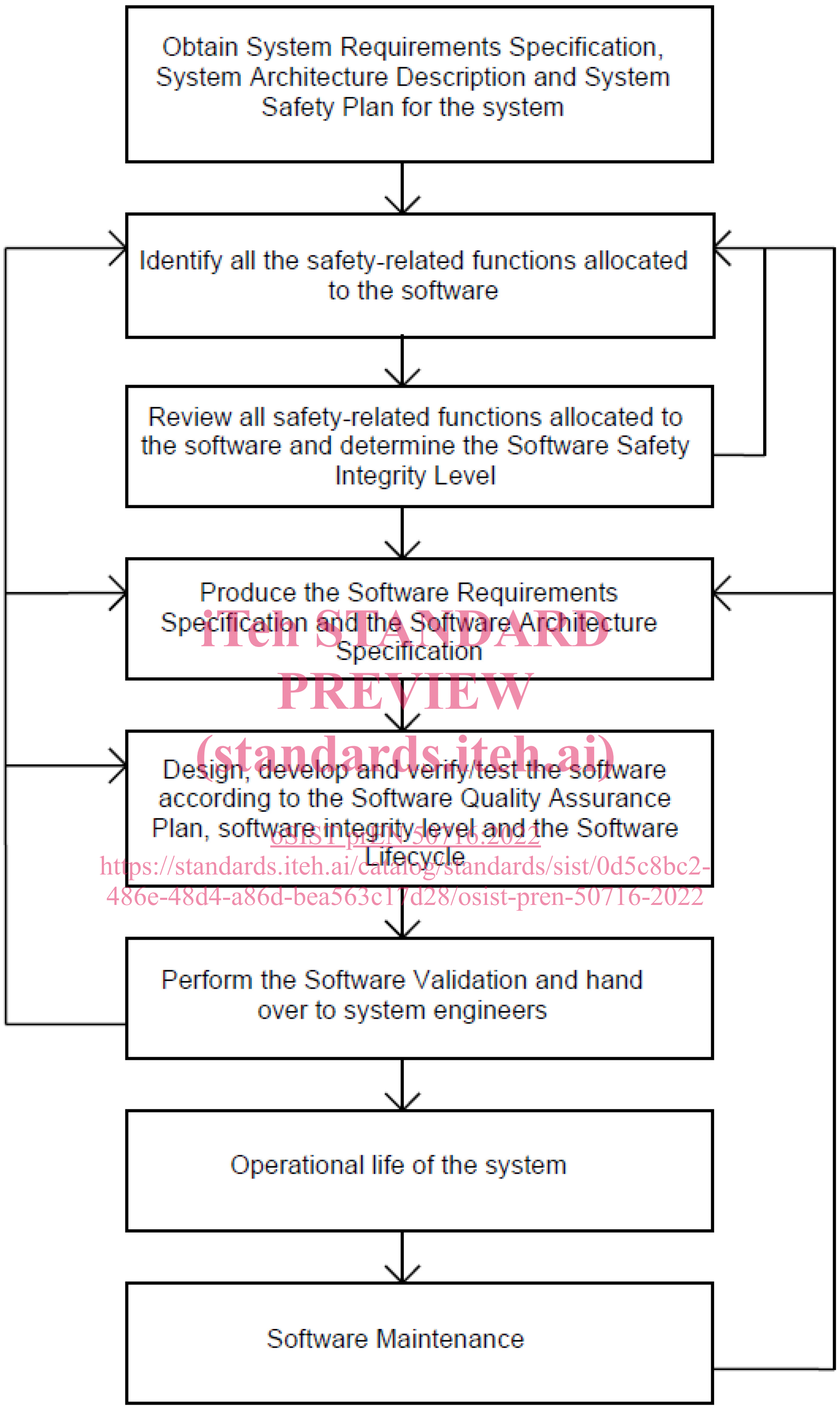
Obtain System Requirements Specification, System Architecture Description and System Safety Plan for the system
Identify all the safety-related functions allocated to the software
Review all safety-related functions allocated to the software and determine the Software Safety Integrity Level
Produce the Software Requirements Specification and the Software Architecture Specification
Design, develop and verify/test the software according to the Software Quality Assurance Plan, software integrity level and the Software Lifecycle
Perform the Software Validation and hand over to system engineers
Operational life of the system
Software Maintenance

Figure 1 — Illustrative Software Route Map

1 Scope

1.1 This document specifies the process and technical requirements for the development of software for programmable electronic systems for use in control, command for signalling applications and any on-board of rolling stock.

This document is not intended to be applied in the area of electric traction power supply (fixed installation) or for power supply and control of conventional applications, e.g. station power supply for offices, shops etc. These applications are typically covered by standards for energy distribution and/or non-railway sectors and/or local legal frameworks.

For railway related fixed installations (electric traction power control and supply) EN 50562 is applicable.

1.2 This document is applicable exclusively to software and the interaction between software and the system of which it is part.

1.3 Intentionally left blank

1.4 This document applies to software as per subclause 1.1 of this document used in railway systems, including

— application programming,

— operating systems,

— support tools,

— firmware.

Application programming comprises high level programming, low level programming and special purpose programming (for example: Programmable logic controller ladder logic).

1.5 This document also addresses the use of pre-existing software (as defined in 3.1.16) and tools. Such software can be used if the specific requirements in 7.3.4.7 and 6.5.4.16 on pre-existing software and for tools in 6.7 are fulfilled.

1.6 Software developed according to any version of EN 50128 or EN 50657 will be considered as compliant and not subject to the requirements on pre-existing software.

NOTE This document is the successor of EN 50128 and EN 50657. Subclause 1.6 ensures continuity in the application of these standards, i.e. software that was developed in accordance with the previous SW standards can still be re-used for new projects.

1.7 This document considers that modern application design often makes use of software that is suitable as a basis for various applications. Such software is then configured by application data for producing the executable software for the application.

1.8 Entry intentionally left empty.

1.9 This document is not intended to be retrospective. It therefore applies primarily to new developments and only applies in its entirety to existing systems if these are subjected to major modifications. For minor changes, only 9.2 applies. However, application of this document during upgrades and maintenance of existing software is highly recommended.

1.10 For the development of User Programmable Integrated Circuits (e.g. field programmable gate arrays (FPGA) and complex programmable logic devices (CPLD)) guidance is provided in EN 50129:2018 Annex F for safety related functions and in EN 50155:2017 for non-safety related functions. Software running on soft-cores of User Programmable Integrated Circuits is within the scope of this document.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

EN ISO 9001:2015, *Quality management systems — Requirements (ISO 9001:2015)*

EN ISO 9000, *Quality management systems — Fundamentals and vocabulary (ISO 9000:2015)*

ISO/IEC/IEEE 90003:2018, *Software engineering — Guidelines for the application of ISO 9001:2015 to computer software*

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1.1
assessment
process of analysis to determine whether software, which may include process, documentation, system, subsystem hardware and/or software components, meets the specified requirements and to form a judgement as to whether the software is fit for its intended purpose

Note 1 to entry: Safety assessment is focused on but not limited to the safety properties of a system.

3.1.2
assessor
entity that carries out an assessment and, by extension, the role carried out by this entity

Note 1 to entry: This definition is based on EN 50126-1:2017 [1] but in this document, independence of the Assessor is always required, see 5.1.2.

Note 2 to entry: This is a software specific role and should not be confused with different types of assessors specified in other standards.

3.1.3
commercial off-the-shelf (COTS) software
software defined by market-driven need, commercially available and whose fitness for purpose has been deemed acceptable by a broad spectrum of commercial users

3.1.4
software component
constituent part of software which has well-defined interfaces and behaviour with respect to the software architecture and design

Note 1 to entry: A software component fulfils the following criteria:

— it is designed according to “Components” (see Table A.20);

— it covers a specific subset of software requirements;

— it is clearly identified and has an independent version inside the configuration management system or is a part of a collection of components (e.g. subsystems) which have an independent version.

3.1.5
configuration manager
entity that is responsible for implementing and carrying out the processes for the configuration management of documents, software and related tools including change management

3.1.6
designer
entity that analyses and transforms specified requirements into acceptable design solutions and, by extension, the role carried out by this entity

3.1.7
entity
person, group or organization who fulfils a role

3.1.8
error
<in software> defect, mistake or inaccuracy in the development process which could result in a deviation from the intended performance or behaviour of the software

Note 1 to entry: Definition is derived from EN 50126-1 3.20 [1] and adapted for software.

3.1.9
failure
<of an item> loss of ability to perform as required

Note 1 to entry: “Failure” is an event, as distinguished from “fault”, which is a state.

[SOURCE: IEC 60050-821:2017, 821-11-19, modified – The notes 1 and 2 have been omitted. A new note 1 to entry has been added]

3.1.10
fault
abnormal condition that could lead to an error in a system

Note 1 to entry: A fault for software is systematic

[SOURCE: IEC 60050-821:2017, 821-11-20, modified – The note 1 to entry has been modified.]

3.1.11
firmware
software stored in read-only memory or in semi-permanent storage such as flash memory, in a way that is functionally independent of applicative software

3.1.12
generic software
software which can be used for a variety of installations purely by the provision of application-specific data

3.1.13
implementer
<of software> entity that transforms specified designs into their realization and, by extension, the role carried out by this entity

3.1.14
integration
<of software> process of assembling software items or software with hardware items, according to the architectural and design specification

3.1.15
maintainability
<of software> capability of the software to be modified; to correct faults, improve performance or other attributes, or adapt it to a different environment

3.1.16
pre-existing software
software developed prior to the application currently in question

Note 1 to entry: This includes commercial off-the-shelf software, open-source software and software previously developed but not in accordance with this document.

[SOURCE: EN 50126-1:2017 [1], 3.43, modified – The end of the definition has been moved to the note 1 to entry.]

3.1.17
programmable logic controller
solid-state control system which has a user programmable memory for storage of instructions to implement specific functions

3.1.18
project management
administrative and/or technical conduct of a project, including RAMS aspects

[SOURCE: EN 50126-1:2017, 3.46]

3.1.19
project manager
entity that carries out project management

3.1.20
robustness
ability of an item to detect and handle abnormal situations

3.1.21
requirements manager
entity that carries out requirements management and, by extension, the role carried out by this entity

3.1.22
requirements management
process of eliciting, documenting, analysing, prioritizing and agreeing on requirements and then controlling change and communicating to relevant stakeholders

3.1.23
risk
<for railway RAMS> combination of expected frequency of loss and the expected degree of severity of that loss

[SOURCE: EN 50126-1:2017, 3.57 [1]]

3.1.24
safety
freedom from unacceptable risk

[SOURCE: IEC 60050-903:2013, 903-01-19 [6]]

3.1.25
safety authority
body responsible for delivering the authorization for the operation of the safety-related system

[SOURCE: IEC 60050-821:2017, 821-12-52]

3.1.26
safety integrity level
one of a number of defined discrete levels for specifying the safety integrity requirements for safety-elated functions to be allocated to the safety-related systems

Note 1 to entry: Safety Integrity Level with the highest figure has the highest level of safety integrity.

Note 2 to entry: It is not possible to allocate a Safety Integrity Level to safety-related processes or other measures.

[SOURCE: EN 50126-1:2017, 3.70 [1]]

3.1.27
safety-related
carries responsibility for safety

[SOURCE: IEC 60050-821:2017, 821-01-73]

3.1.28
safety-related software
software which performs safety-related functions

Note 1 to entry: Software is called safety-related if at least one of its properties is used in the safety argument for the system in which it is applied. These properties can be of functional or non-functional nature.

[SOURCE: IEC 60050-821:2017, 821-12-60, modified – “safety functions” has been replaced with “safety-related functions”. The note 1 to entry has been added.]

3.1.29
software
programs, procedures, rules, documentation and data of an information processing system

EXAMPLE Requirements and design specifications; source code listings, check lists and comments; “Help” text and messages for display at the computer/human interface; instructions for installation and operation; and support guides for hardware and software maintenance.

Note 1 to entry: Software is an intellectual creation that is independent of the medium upon which it is recorded.

Note 2 to entry: Software requires hardware devices to execute programs, and to store and transmit data.

Note 3 to entry: Types of software include firmware, system software, and application software.

[SOURCE: IEC 60050-192:2015, 192-01-07]

3.1.30
software baseline
complete and consistent set of source code, executable files, configuration files, installation scripts and documentation that are needed for a software release

Note 1 to entry: Information about compilers, operating systems, pre-existing software and dependent tools is stored as part of the software baseline. This will enable the organization to reproduce defined versions and be the input for future releases at enhancements or at upgrade in the maintenance phase.

3.1.31
software deployment
transferring, installing and activating a deliverable software baseline

3.1.32
software integrity level
classification which determines the techniques and measures that have to be applied to software

Note 1 to entry: This includes basic integrity and the safety integrity levels 1 to 4.

3.1.33
software lifecycle
those activities occurring during a period of time that starts when software is conceived and ends when the software is no longer available for use

3.1.34
software maintenance
modification for the purposes of software fault removal, adaptation to a new environment, or improvement of performance

[SOURCE: IEC 60050-192:2015, 192-06-15, modified: notes avoided]

3.1.35
tester
entity that carries out testing and, by extension, the role carried out by this entity

3.1.36
testing
<of software> process of executing software under controlled conditions as to ascertain its behaviour and performance compared to the corresponding requirements specification

3.1.37
tool class T1
tool which is not used to generate outputs which can directly or indirectly contribute to the executable code (including data) of the software

Note 1 to entry: T1 examples include: a text editor or a requirement or design support tool with no automatic code generation capabilities.

3.1.38
tool class T2
tool which is used to support the test or verification of the design or executable code, where errors in the tool can fail to reveal defects but cannot directly create errors in the executable software

Note 1 to entry: T2 examples include: a test harness generator; a test coverage measurement tool; a static analysis tool

3.1.39
tool class T3
tool which is used to generate outputs which can directly or indirectly contribute to the executable code (including data) of the system

Note 1 to entry: T3 examples include: a source code compiler, a data/algorithms compiler, a tool to change set-points during system operation; an optimizing compiler where the relationship between the source code program and the generated object code is not obvious; a compiler that incorporates an executable run-time package into the executable code