



BSI Standards Publication

Railways Applications — Rolling stock applications — Software on Board Rolling Stock

National foreword

This British Standard is the UK implementation of EN 50657:2017+A1:2023. It supersedes BS EN 50657:2017, which is withdrawn.

The start and finish of text introduced or altered by amendment is indicated in the text by tags. Tags indicating changes to CENELEC text carry the number of the CENELEC amendment. For example, text altered by CENELEC amendment A1 is indicated by A1 A1.

The UK participation in its preparation was entrusted to Technical Committee GEL/9/2, Railway Electrotechnical Applications - Rolling stock.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2023
Published by BSI Standards Limited 2023

ISBN 978 0 539 25541 6

ICS 35.080; 35.240.60

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 September 2017.

Amendments/corrigenda issued since publication

Date	Text affected
30 November 2023	Implementation of CENELEC amendment A1:2023

English Version

Railways Applications - Rolling stock applications - Software on Board Rolling Stock

Applications ferroviaires - Applications du matériel roulant -
Logiciels embarqués

Bahnanwendungen - Anwendungen für Schienenfahrzeuge
- Software auf Schienenfahrzeugen

This European Standard was approved by CENELEC on 2017-05-08. 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.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CENELEC member.

This European Standard exists 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, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.



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

CEN-CENELEC Management Centre: Avenue Marnix 17, B-1000 Brussels

Contents

Page

European foreword.....	8
Introduction.....	9
1 Scope	12
2 Normative references	13
3 Terms, definitions and abbreviations	13
3.1 Terms and definitions	13
3.2 Abbreviations	19
4 Objectives, conformance and software integrity levels.....	20
5 Software management and organization.....	21
5.1 Organization, roles and responsibilities	21
5.1.1 Objective	21
5.1.2 Requirements.....	21
5.2 Personnel competence	25
5.2.1 Objectives	25
5.2.2 Requirements.....	25
5.3 Lifecycle issues and documentation	25
5.3.1 Objectives	25
5.3.2 Requirements.....	25
6 Software assurance	28
6.1 Software testing	28
6.1.1 Objective	28
6.1.2 Input documents	28
6.1.3 Output documents.....	28
6.1.4 Requirements.....	29
6.2 Software verification.....	29
6.2.1 Objective	29
6.2.2 Input documents	30
6.2.3 Output documents.....	30
6.2.4 Requirements.....	30
6.3 Software validation.....	31
6.3.1 Objective	31
6.3.2 Input documents	31
6.3.3 Output documents.....	31
6.3.4 Requirements.....	32
6.4 Software assessment	33
6.4.1 Objective	33
6.4.2 Input documents	33
6.4.3 Output documents.....	33
6.4.4 Requirements.....	33
6.5 Software quality assurance	35
6.5.1 Objectives	35
6.5.2 Input documents	35
6.5.3 Output documents.....	35
6.5.4 Requirements.....	35
6.6 Modification and change control	38

6.6.1	Objectives	38
6.6.2	Input documents	38
6.6.3	Output documents.....	38
6.6.4	Requirements.....	38
6.7	Support tools and languages	39
6.7.1	Objectives	39
6.7.2	Input documents	39
6.7.3	Output documents.....	39
6.7.4	Requirements.....	39
7	Software development.....	42
7.1	Lifecycle and documentation for software	42
7.1.1	Objectives	42
7.1.2	Requirements.....	42
7.2	Software requirements.....	42
7.2.1	Objectives	42
7.2.2	Input documents	42
7.2.3	Output documents.....	43
7.2.4	Requirements.....	43
7.3	Architecture and Design.....	45
7.3.1	Objectives	45
7.3.2	Input documents	45
7.3.3	Output documents.....	45
7.3.4	Requirements.....	46
7.4	Component design.....	52
7.4.1	Objectives	52
7.4.2	Input documents	52
7.4.3	Output documents.....	52
7.4.4	Requirements.....	52
7.5	Component implementation and testing	54
7.5.1	Objectives	54
7.5.2	Input documents	54
7.5.3	Output documents.....	54
7.5.4	Requirements.....	54
7.6	Integration	55
7.6.1	Objectives	55
7.6.2	Input documents	55
7.6.3	Output documents.....	55
7.6.4	Requirements.....	56
7.7	Overall Software Testing / Final Validation	57
7.7.1	Objectives	57
7.7.2	Input documents	57
7.7.3	Output documents.....	57
7.7.4	Requirements.....	58
7.8	Development of Software configured by application data	59
7.8.1	Objective	59
7.8.2	Requirements.....	59
8	Systems configured by application data: development of application data.....	60
8.1	Objectives	60
8.2	Input documents	60
8.3	Output documents	61
8.4	Requirements	61
8.4.1	Application Development Process	61
8.4.2	Application Requirements Specification.....	62

8.4.3	Architecture and Design.....	62
8.4.4	Application Data Production.....	63
8.4.5	Application Integration and Testing.....	63
8.4.6	Application Validation and Assessment	64
8.4.7	Application preparation procedures and tools.....	64
9	Software deployment and maintenance	64
9.1	Software deployment	64
9.1.1	Objective	64
9.1.2	Input documents	64
9.1.3	Output documents.....	64
9.1.4	Requirements.....	65
9.2	Software maintenance	66
9.2.1	Objective	66
9.2.2	Input documents	66
9.2.3	Output documents.....	66
9.2.4	Requirements.....	67
Annex A	(normative) Criteria for the Selection of Techniques and Measures.....	69
A.1	General	69
A.2	Clauses tables	70
A.3	Detailed tables	77
Annex B	(normative) Key software roles and responsibilities	82
Annex C	(informative) Documents Control Summary	95
Annex D	(informative) Bibliography of techniques.....	97
D.1	Artificial Intelligence Fault Correction.....	97
D.2	Analysable Programs	97
D.3	Avalanche/Stress Testing	98
D.4	Boundary Value Analysis	98
D.5	Backward Recovery	99
D.6	Cause Consequence Diagrams	99
D.7	Checklists	99
D.8	Control Flow Analysis.....	100
D.9	Common Cause Failure Analysis	100
D.10	Data Flow Analysis.....	100
D.11	Data Flow Diagrams	101
D.12	Data Recording and Analysis.....	101
D.13	Decision Tables and Truth Tables	102
D.14	Defensive Programming	102
D.15	Coding Standards and Style Guide.....	103
D.16	Diverse Programming	104
D.17	Dynamic Reconfiguration.....	105
D.18	Equivalence Classes and Input Partition Testing.....	105
D.19	Error Detecting and Correcting Codes.....	106
D.20	Error Guessing.....	106
D.21	Error Seeding.....	106
D.22	Event Tree Analysis	107
D.23	Fagan Inspections.....	107

D.24	Failure Assertion Programming	107
D.25	SEEA – Software Error Effect Analysis.....	108
D.26	Fault Detection and Diagnosis	108
D.27	Finite State Machines/State Transition Diagrams.....	109
D.28	Formal Methods	110
D.28.1	General	110
D.28.2	CSP – Communicating Sequential Processes.....	110
D.28.3	CCS – Calculus of Communicating Systems.....	111
D.28.4	HOL – Higher Order Logic	111
D.28.5	LOTOS.....	111
D.28.6	OBJ	111
D.28.7	Temporal logic	112
D.28.8	VDM – Vienna Development Method.....	112
D.28.9	Z method.....	113
D.28.10	B method.....	113
D.28.11	Model Checking	114
D.29	Formal Proof	114
D.30	Forward Recovery.....	114
D.31	Graceful Degradation.....	115
D.32	Impact Analysis.....	115
D.33	Information Hiding / Encapsulation	115
D.34	Interface Testing	116
D.35	Language Subset.....	116
D.36	Memorizing Executed Cases	116
D.37	Metrics	117
D.38	Modular Approach.....	117
D.39	Performance Modelling	118
D.40	Performance Requirements.....	118
D.41	Probabilistic Testing	119
D.42	Process Simulation	119
D.43	Prototyping / Animation	120
D.44	Recovery Block	120
D.45	Response Timing and Memory Constraints.....	120
D.46	Re-Try Fault Recovery Mechanisms.....	120
D.47	Safety Bag	121
D.48	Software Configuration Management	121
D.49	Strongly Typed Programming Languages	121
D.50	Structure Based Testing	122
D.51	Structure Diagrams.....	122
D.52	Structured Methodology.....	123
D.53	Structured Programming.....	123
D.54	Suitable Programming languages.....	124
D.55	Time Petri Nets	125
D.56	Walkthroughs / Design Reviews.....	125
D.57	Object Oriented Programming	125
D.58	Traceability.....	126

D.59 Metaprogramming.....	126
D.60 Procedural programming	127
D.61 Clause intentionally left empty	127
D.62 Clause intentionally left empty	127
D.63 Clause intentionally left empty	127
D.64 Clause intentionally left empty	127
D.65 Data modelling	127
D.66 Control Flow Diagram/Control Flow Graph	128
D.67 Sequence diagram	129
D.68 Tabular Specification Methods	129
D.69 Application specific language.....	130
D.70 UML (Unified Modelling Language)	130
D.71 Domain specific languages.....	131
D.72 Segregation.....	131
Annex E (informative) Changes in this European Standard compared to EN 50128:2011	133
Annex ZZ (informative) Relationship between this European Standard and the Essential Requirements of EU Directive 2008/57/EC	139
Bibliography	140

Figures

Figure 1 — Illustrative Software Route Map	11
Figure 2 — Illustration of the preferred organizational structure	22
Figure 3 — Illustrative Development Lifecycle 1	27
Figure 4 — Illustrative Development Lifecycle 2	28

Tables

Table 1 — Relation between tool class and applicable numbered entries	42
Table A.1 — Lifecycle Issues and Documentation (5.3).....	70
Table A.2 — Software Requirements Specification (7.2)	72
Table A.3 — Software Architecture (7.3).....	73
Table A.4 — Software Design and Implementation (7.3 and 7.4)	74
Table A.5 — Verification and Testing (6.2, 7.3 and 7.4).....	75
Table A.6 — Integration (7.6)	75
Table A.7 — Overall Software Testing (6.2 and 7.7)	75
Table A.8 — Software Analysis Techniques (6.3).....	76
Table A.9 — Software Quality Assurance (6.5)	76
Table A.10 — Software Maintenance (9.2).....	76
Table A.11 — Data Preparation Techniques (8.4).....	77
Table A.12 — Coding Standards	77
Table A.13 — Dynamic Analysis and Testing	78
Table A.14 — Functional/Black Box Test	78
Table A.15 — Intentionally left empty	78

Table A.16 — Intentionally left empty	78
Table A.17 — Modelling	79
Table A.18 — Performance Testing	79
Table A.19 — Static Analysis	79
Table A.20 — Components	80
Table A.21 — Test Coverage for Code.....	80
Table A.22 — Object Oriented Software Architecture	81
Table A.23 — Object Oriented Detailed Design	81
Table B.1 — Requirements Manager Role Specification	83
Table B.2 — Designer Role Specification.....	84
Table B.3 — Implementer Role Specification	85
Table B.4 — Tester Role Specification	86
Table B.5 — Verifier Role Specification.....	87
Table B.6 — Integrator Role Specification	88
Table B.7 — Validator Role Specification.....	89
Table B.8 — Assessor Role Specification	91
Table B.9 — Project Manager Role Specification.....	93
Table B.10 — Configuration Manager Role Specification	94
Table C.1 — Documents Control Summary	95
Table E.1 — Correspondence between this European Standard and EN 50128:2011	133
Table ZZ.1 — Correspondence between this European Standard, the TSI “Locomotives and Passenger Rolling Stock” (REGULATION (EU) No 1302/2014 of 18 November 2014) and Directive 2008/57/EC	139

European foreword

This document (EN 50657:2017) has been prepared by CLC/SC 9XB, “Electrical, electronic and electromechanical material on board rolling stock, including associated software”.

The following dates are fixed:

- latest date by which this document has (dop) 2018-05-08
to be implemented at national level by
publication of an identical national
standard or by endorsement
- latest date by which the national (dow) 2020-05-08
standards conflicting with this document
have to be withdrawn

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC shall not be held responsible for identifying any or all such patent rights.

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

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

This document adapts EN 50128:2011 (prepared by CLC/SC 9XA “Communication, signalling and processing systems”) for the application in the Rolling Stock domain. It uses the same structure and section numbering as EN 50128:2011. Where requirements of EN 50128:2011 do not apply to rolling stock, the respective text is replaced by the term “intentionally left empty”.

The main changes with respect to EN 50128:2011 are listed in Annex E.

European foreword to Amendment 1

This document [EN 50657:2017/A1:2023] has been prepared by CLC/SC 9XB “Electrical, electronic and electromechanical material on board rolling stock, including associated software”.

The following dates are fixed:

- latest date by which this document has (dop) 2024-08-03
to be implemented at national level by
publication of an identical national
standard or by endorsement
- latest date by which the national (dow) 2026-08-03
standards conflicting with this document
have to be withdrawn

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC shall not be held responsible for identifying any or all such patent rights.

This document has been prepared under a standardization request addressed to CENELEC by the European Commission. The Standing Committee of the EFTA States subsequently approves these requests for its Member States.

For the relationship with EU Legislation, see informative Annex ZZ, which is an integral part of this document.

Any feedback and questions on this document should be directed to the users’ national committee. A complete listing of these bodies can be found on the CENELEC website.

Introduction

This European Standard is related to, and should be read in conjunction with the EN 50126 series, *Railway applications — The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)*.

This European Standard 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 European Standard provides a set of requirements for the development, deployment and maintenance of any software intended for railway rolling stock 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 European Standard.

The key concept of this European Standard is that of levels of software integrity. This European Standard 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 European Standard was first introduced in the EN 50126 series.

This European Standard 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. In this version, 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 European Standard does not give guidance on which level of software safety 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 the EN 50126 series to define the process of specifying the safety-related functions allocated to software.

This European Standard specifies those measures necessary to achieve these requirements.

The EN 50126 series 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 Specification 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 European Standard are shown in Figure 1 and are as follows:

- f) 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);
- g) 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);
- h) integrate the software on the target hardware and verify functionality (7.6);
- i) accept and deploy the software (7.7 and 9.1);
- j) if software maintenance is required during operational life then re-activate this European Standard as appropriate (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 (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 European Standard does not mandate the use of a particular software development lifecycle. However, illustrative lifecycle and documentation sets are given in 5.3, Figure 3 and Figure 4 and in 7.1.

Tables have been formulated ranking various techniques/measures against the safety 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 European Standard 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 that address IT security in depth are the ISO/IEC 27000 standards, ISO/IEC/TR 19791 and the IEC 62443 series.

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.

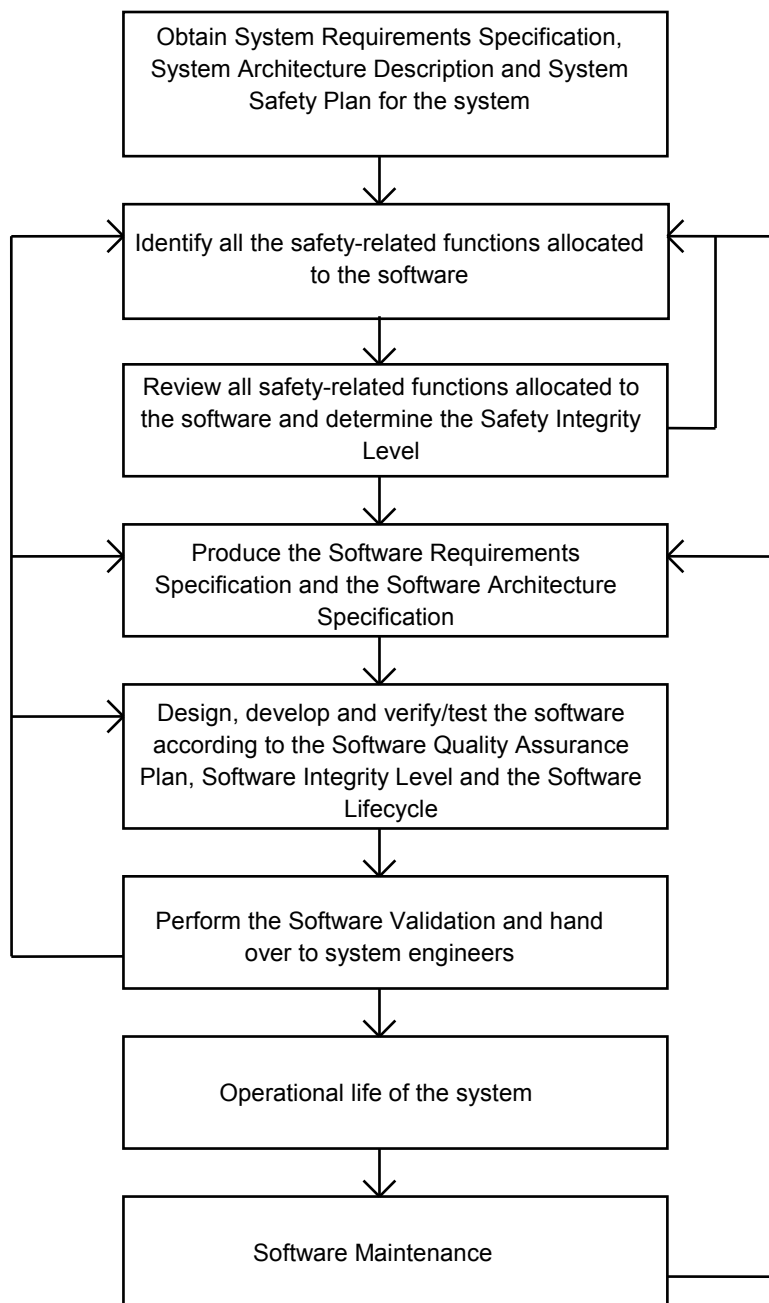


Figure 1 — Illustrative Software Route Map

1 Scope

1.1 This European Standard specifies the process and technical requirements for the development of software for programmable electronic systems for use in rolling stock applications.

Outside the scope of this standard is software that:

- is part of signalling equipment (CENELEC sub-committee SC9XA applications) installed on board trains, or
- does not contribute to, and is segregated from Rolling Stock operational functions.

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

1.3 Entry intentionally left empty

1.4 This European Standard applies to safety-related as well as non-safety-related software, including for example:

- 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 European Standard also addresses the use of pre-existing software and tools. Such software may 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 a valid version of EN 50128 is considered as compliant to this standard. Software previously developed in accordance with any version of EN 50128 is also considered as compliant and not subject to the requirements on pre-existing software. For SIL1-SIL4 software under the scope of this standard, requirements included in this European Standard are equivalent to the SIL1-SIL4 software requirements of EN 50128:2011.

A1 NOTE This document was derived from the signalling standard EN 50128 which in many cases was also applied in Rolling Stock applications. Subclause 1.6 ensures continuity in the application of the standards, i.e., software that was developed in accordance with EN 50128 can still be re-used for new projects. **A1**

1.7 This European Standard 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. This European Standard applies to such software. In addition, specific requirements for application data will be given.

1.8 Entry intentionally left empty.

1.9 This European Standard 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 European Standard during upgrades and maintenance of existing software is recommended.

1.10 The relevant sections of this software standard are also applicable to programmable components (e.g. FPGA and CPLD), in addition to the applicable hardware standard (e.g. EN 50129, EN 50155, EN 61508-2). However, requirements of this software standard that are already covered by the applicable hardware standard do not need to be re-addressed.

When it is possible to exhaustively test the programmable logic for all possible inputs and internal logic states, this European Standard does not apply.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

ISO/IEC 90003:2014, *Software engineering — Guidelines for the application of ISO 9001:2008 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 to form a judgement on whether a product, system or process meet the specified requirements, based on evidence

Note 1 to entry: With regards to software, assessment may include development processes, documentation, system, subsystem hardware and/or software components. Assessment is focused on but not limited to the safety properties of the software being assessed.

3.1.2

Assessor

appointed independent entity that carries out an assessment

Note 1 to entry: This definition is based on EN 50126–1:2017 but in this European Standard, independence of the Assessor is always required, see 5.1.2.

Note 2 to entry: The specific meaning for the term “Assessor” in this standard is defined in 5.1.2.4, 5.1.2.5, 5.1.2.6 and Annex B, Table B.8 in combination herein. This is a software specific role and should not be confused with different types of Assessor 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

[SOURCE: EN 50126-1:2017, 3.10, modified]

3.1.4

component

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

[SOURCE: EN 50128:2011, 3.1.4]

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

[SOURCE: EN 50128:2011, 3.1.5]

3.1.6

entry intentionally left empty

3.1.7

Designer

entity that analyses and transforms specified requirements into acceptable design solutions which have the required integrity level

[SOURCE: EN 50128:2011, 3.1.7, modified]

3.1.8

entity

person, group or organization who fulfils a role as defined in this Standard

[SOURCE: EN 50126-1:2017, 3.19]

3.1.9

error, fault

defect, mistake or inaccuracy which could result in failure or in a deviation from the intended performance or behaviour

[SOURCE: EN 50128:2011, 3.1.9]

3.1.10

failure

unacceptable difference between required and observed performance

[SOURCE: EN 50128:2011, 3.1.10]

3.1.11

Entry intentionally left empty

3.1.12

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

[SOURCE: EN 50128:2011, 3.1.12]

3.1.13

Entry intentionally left empty

3.1.14

Implementer

entity that transforms specified designs into their realization

3.1.15

integration

process of assembling software and/or hardware items, according to the architectural and design specification, and testing the integrated unit

[SOURCE: EN 50128:2011, 3.1.15]

3.1.16

Integrator

entity that carries out software integration

[SOURCE: EN 50128:2011, 3.1.16]

3.1.17

pre-existing software

all software developed prior to the application currently in question, including commercial off-the shelf (COTS) software, open source software and software previously developed but not in accordance with this European Standard

Note 1 to entry: Software developed in accordance with EN 50128 is not subject to the requirements for pre-existing software, see 1.6.

3.1.18

open source software

source code available to the general public with relaxed or non-existent copyright restrictions

[SOURCE: EN 50128:2011, 3.1.18]

3.1.19

programmable logic controller

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

[SOURCE: EN 50128:2011, 3.1.19]

3.1.20

project management

administrative and/or technical conduct of a project, including safety aspects

[SOURCE: EN 50126-1:2017, 3.49]

3.1.21

Project Manager

entity that carries out project management

[SOURCE: EN 50128:2011, 3.1.21]

3.1.22

Entry intentionally left empty

3.1.23

robustness

ability of an item to detect and handle abnormal situations

[SOURCE: EN 50128:2011, 3.1.23]

3.1.24

Requirements Manager

entity that carries out requirements management

[SOURCE: EN 50128:2011, 3.1.24]

3.1.25

requirements management

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

Note 1 to entry: Requirements management is a continuous process throughout a project.

[SOURCE: EN 50128:2011, 3.1.25, modified — The second part of the definition was turned into the present Note 1 to entry.]

3.1.26

risk

combination of expected frequency of loss and the expected degree of severity of that loss

[SOURCE: EN 50126-1:2017, 3.59]

3.1.27

safety

freedom from unacceptable risk

[SOURCE: ISO/IEC Guide 51:1999, 3.1]

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

3.1.28

safety authority

body entrusted with the regulatory tasks regarding railway safety in accordance with the respective legal framework

[SOURCE: EN 50126-1:2017, 3.67]

3.1.29

safety integrity level

one of a number of defined discrete levels for specifying the safety integrity requirements of safety-related functions

[SOURCE: EN 50126-1:2017, 3.72, modified]

3.1.30

safety-related

carries responsibility for safety

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

3.1.31

safety-related software

software which performs safety-related functions

3.1.32**segregation**

technique to separate software components in order to isolate one or more attributes of the software to prevent hazardous interactions and interference

3.1.33**software**

intellectual creation comprising the programs, procedures, rules, data and any associated documentation pertaining to the operation of a system

[SOURCE: EN 50128:2011, 3.1.31]

3.1.34**software baseline**

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

[SOURCE: EN 50128:2011, 3.1.32]

Note 1 to entry: Information about compilers, operating systems, pre-existing software and dependent tools is stored as part of the 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.

3.1.35**software deployment**

transferring, installing and activating a deliverable software baseline

[SOURCE: EN 50126-1:2017, 3.78, modified]

3.1.36**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

[SOURCE: EN 50128:2011, 3.1.34]

Note 1 to entry: The software lifecycle typically includes a requirements phase, design phase, test phase, integration phase, deployment phase and a maintenance phase.

3.1.37**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

[SOURCE: EN 50128:2011, 3.1.35]

3.1.38**software maintenance**

action, or set of actions, carried out on software after deployment with the aim of enhancing or correcting its functionality

[SOURCE: EN 50128:2011, 3.1.36]

3.1.39**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.

3.1.40

Tester

entity that carries out testing

[SOURCE: EN 50128:2011, 3.1.40]

3.1.41

testing

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

[SOURCE: EN 50128:2011, 3.1.41]

3.1.42

tool class T1

tools which generate no outputs which can directly or indirectly contribute to the executable code (including data) of the software

[SOURCE: EN 50128:2011, 3.1.42, modified]

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

3.1.43

tool class T2

tools which 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

[SOURCE: EN 50128:2011, 3.1.43, modified]

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

3.1.44

tool class T3

tools which 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.

[SOURCE: EN 50128:2011, 3.1.44, modified]

3.1.45

Entry intentionally left empty

3.1.46

validation

process of analysis followed by a judgment based on evidence to determine whether an item (e.g. process, documentation, software or application) fits the user needs, in particular with respect to safety and quality and with emphasis on the suitability of its operation in accordance to its purpose in its intended environment

[SOURCE: EN 50128:2011, 3.1.46]

3.1.47**Validator**

entity that is responsible for the validation

[SOURCE: EN 50128:2011, 3.1.47]

3.1.48**verification**

process of examination followed by a judgment based on evidence that output items (process, documentation, software or application) of a specific development phase fulfil the requirements of that phase with respect to completeness, correctness and consistency

[SOURCE: EN 50128:2011, 3.1.48]

Note 1 to entry: Verification is mostly based on document reviews (design, implementation, test documents, etc.).

3.1.49**Verifier**

entity that is responsible for one or more verification activities

[SOURCE: EN 50128:2011, 3.1.49]

3.2 Abbreviations

For the purposes of this document, the following abbreviations apply.

ASR	Assessor
COTS	Commercial off-the-shelf
CPLD	Complex Programmable Logic Device
DES	Designer
FPGA	Field Programmable Gate Array
HR	Highly Recommended
IMP	Implementer
INT	Integrator
IT	Information Technology
JSD	Jackson System Development Method
M	Mandatory
MASCOT	Modular Approach to Software Construction, Operation and Test
NR	Not Recommended
PM	Project Manager
R	Recommended
RAMS	Reliability, Availability, Maintainability and Safety
RQM	Requirements Manager
SDL	Specification and Description Language
SIL	Safety Integrity Level
SOM	Service Oriented Modelling
SSADM	Structured Systems Analysis and Design Methodology
TST	Tester

V&V	Verification and Validation
VAL	Validator
VER	Verifier

4 Objectives, conformance and software integrity levels

4.1 The allocation of system functions to software, as well as software interfaces, shall be identified in the system documentation. The system in which the software is embedded shall be fully defined with respect to the following:

- functions and interfaces;
- application conditions;
- configuration or architecture of the system;
- hazards to be controlled;
- safety integrity requirements;
- apportionment of requirements and allocation of SIL or basic integrity to software and hardware;
- timing constraints.

NOTE 1 The allocation of safety integrity requirements may lead to different integrity levels for well-separated software and hardware parts of a subsystem. This allocation depends on the contribution of the software and hardware parts of the subsystem to the safety-related functions and on the mechanisms for the failure mitigation including the separation of functions with different integrity levels (see EN 50126–2:2017).

NOTE 2 This European Standard uses the term System Requirements Specification to address the complete set of system requirements including the safety requirements.

4.2 The software integrity shall be specified as one of five levels, from basic integrity (the lowest) to SIL 4 (the highest).

4.3 The required software integrity level shall be decided and assessed at system level, on the basis of the system safety integrity level and the level of risk associated with the use of the software in the system.

4.4 The basic integrity requirements of this European Standard shall be fulfilled for the software part of functions that are not safety-related or that have a safety impact below SIL 1.

4.5 To conform to this European Standard it shall be shown that each of the requirements has been satisfied to the software integrity level defined and therefore the objective of the subclause in question has been met.

4.6 Where a requirement is qualified by the words “to the extent required by the software integrity level”, this indicates that a range of techniques and measures shall be used to satisfy that requirement.

4.7 Where 4.6 is applied, tables from normative Annex A shall be used to assist in the selection of techniques and measures appropriate to the software integrity level. The selection shall be documented in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan. Guidance to these techniques is given in the informative Annex D.

4.8 If a technique or measure which is ranked as highly recommended (HR) in the tables is not used, then the rationale for using alternative techniques shall be detailed and recorded either in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan. This is not necessary if an approved combination of techniques given in the corresponding table is used. The selected techniques shall be demonstrated to have been applied correctly.

4.9 If a technique or measure is proposed to be used that is not contained in the tables then its effectiveness and suitability in meeting the particular requirement and overall objective of the subclause shall be justified and recorded in either the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan.

4.10 Compliance with the requirements of a particular subclause and their respective techniques and measures detailed in the tables shall be verified by the inspection of documents required by this European Standard. Where appropriate, other objective evidence, auditing and the witnessing of tests shall also be taken into account.

5 Software management and organization

5.1 Organization, roles and responsibilities

5.1.1 Objective

To ensure that all personnel who have responsibilities for the software are organized, empowered and capable of fulfilling their responsibilities.

5.1.2 Requirements

5.1.2.1 As a minimum, the supplier shall implement requirements dealing with the organization and management of the personnel and responsibilities.

NOTE These requirements might be met with the parts of EN ISO 9001.

5.1.2.2 Responsibilities shall be compliant with the requirements defined in Annex B.

5.1.2.3 The personnel assigned to the roles involved in the development or maintenance of the software shall be named and recorded.

5.1.2.4  The Assessor shall be appointed for SIL 1 – SIL 4 and may be appointed by the supplier or by the customer. 

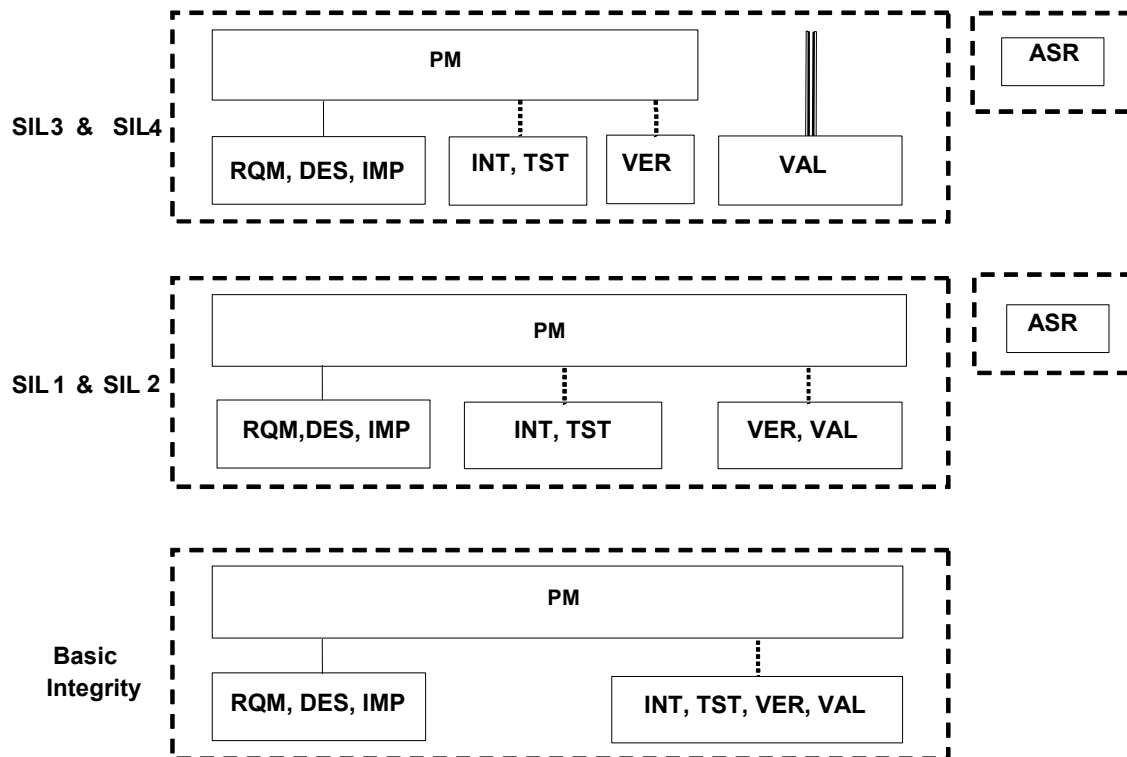
5.1.2.5  The Assessor may be part of any stakeholder organization (e.g. supplier, customer or third-party organisation). 

5.1.2.6 The Assessor shall be independent from the project.

5.1.2.7 The Assessor shall be given authority to perform the assessment of the software.

5.1.2.8 The Validator shall give agreement/disagreement for the software release.

5.1.2.9 Throughout the Software Lifecycle, the assignment of roles to persons shall be in accordance with 5.1.2.10 to 5.1.2.14, to the extent required by the software integrity level.



Key



can be the same person

can be the same organization

shall report to the Project Manager

can report to the Project Manager

shall not report to the Project Manager

PM Project Manager**RQM** Requirements Manager**DES** Designer**IMP** Implementer**ASR** Assessor**INT** Integrator**TST** Tester**VER** Verifier**VAL** Validator

NOTE For the role of the Configuration Manager, see Table B.10, there are no independence requirements.

Figure 2 — Illustration of the preferred organizational structure

NOTE Figure 2 is only illustrative for the preferred organizational structure.

5.1.2.10 The preferred organizational structure for SIL 3 and SIL 4 is as follows:

- Requirements Manager, Designer and Implementer for a software component can be the same person.
- Requirements Manager, Designer and Implementer for a software component shall report to the Project Manager.
- Integrator and Tester for a software component can be the same person.
- Integrator and Tester for a software component can report to the Project Manager or to the Validator.

- e) Verifier can report to the Project Manager or to the Validator.
- f) Validator shall not report to the Project Manager i.e. the Project Manager shall have no influence on the Validator's decisions but the Validator informs the Project Manager about his decisions.
- g) A person who is Requirements Manager, Designer or Implementer for a software component shall neither be Tester nor Integrator for the same software component.
- h) A person who is Integrator or Tester for a software component shall neither be Requirements Manager, Designer nor Implementer for the same software component.
- i) A person who is Verifier shall neither be Requirements Manager, Designer, Implementer, Integrator, Tester nor Validator.
- j) A person who is Validator shall neither be Requirements Manager, Designer, Implementer, Integrator, Tester nor Verifier.
- k) A person who is Project Manager can additionally perform the roles of Requirements Manager, Designer, Implementer, Integrator, Tester or Verifier, providing that the requirements for the independence between these additional roles are respected.
- l) Project Manager, Requirements Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator can belong to the same organization.
- m) The Assessor shall be independent and organizationally independent from the roles of Project Manager, Requirements Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator.

However, one of the following options may apply:

- n) A person who is Validator may also perform the role of Verifier, but still maintaining independence from the Project Manager. In this case the Verifier's output documents shall be reviewed by another competent person with the same level of independence as the Validator.
- o) A person who is Verifier may also perform the role of Integrator and Tester, in which case the role of Validator shall check the adequacy of the documented evidence from integration and testing with the specified verification objectives, hence maintaining two levels of checking within the project organization.

5.1.2.11 The preferred organizational structure for SIL 1 and SIL 2 is as follows:

- a) Requirements Manager, Designer and Implementer for a software component can be the same person and shall report to the Project Manager.
- b) Integrator and Tester for a software component can be the same person.
- c) Integrator and Tester for a software component can report to the Project Manager or to the Validator.
- d) Verifier and Validator can be the same person.
- e) Verifier and Validator can report to the Project Manager.
- f) A person who is Requirements Manager, Designer or Implementer for a software component shall be neither Tester nor Integrator for the same software component.
- g) A person who is Integrator or Tester for a software component shall neither be Requirements Manager, Designer nor Implementer for the same software component.
- h) A person who is Verifier or Validator shall neither be Requirements Manager, Designer, Implementer, Integrator nor Tester.

- i) A person who is a Project Manager can additionally perform the roles of Requirements Manager, Designer, Implementer, Integrator, Tester, Verifier or Validator, provided that the requirements for the independence between these additional roles are respected.
- j) Project Manager, Requirements Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator can belong to the same organization.
- k) The Assessor shall be independent and organizationally independent from the roles of Project Manager, Requirements Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator.

However, the following options can apply:

- l) A person who is Verifier may also perform the role of Integrator and Tester, in which case the role of Validator shall include reviewing the Verifier's output documents hence maintaining two levels of checking within the project organization.
- m) A person who is Validator may also perform the role of Verifier, Integrator and Tester. In this case the Verifier's output documents shall be reviewed by another person who is neither the Requirements Manager, the Designer nor the Implementer.

5.1.2.12 The preferred organizational structure for basic integrity is:

- a) Requirements Manager, Designer and Implementer for a software component can be the same person and shall be managed by the Project Manager.
- b) Integrator, Tester, Verifier and Validator for a software component can be the same person.
- c) Integrator, Tester, Verifier and Validator can be managed by the Project Manager.
- d) A person who is Requirements Manager, Designer or Implementer for a software component shall be neither Tester nor Integrator for the same software component.
- e) A person who is Verifier or Validator shall neither be Requirements Manager, Designer, nor Implementer.
- f) A person who is Project Manager can additionally perform the roles of Requirements Manager, Designer, Implementer, Integrator, Tester, Verifier or Validator providing that the requirements for the independence between these additional roles are respected.
- g) Project Manager, Requirements Manager, Designer, Implementer, Integrator, Tester, Verifier and Validator can belong to the same organization.

However, the following option can apply:

- h) Requirements Manager, Designer, Implementer, Integrator and Tester can be the same person;

also the Validator and Verifier can be the same person;

however, a person who is Verifier or Validator shall neither be Requirements Manager, Designer, nor Implementer.

Where the Tester is the Implementer, the Verifier and/or the Validator shall determine whether the software integration and/or the overall software tests address all relevant software behaviours including response to unexpected or unintended inputs.

5.1.2.13 The roles Requirements Manager, Designer and Implementer for one component can perform the roles Tester and Integrator for a different component.

5.1.2.14 The roles of the Verifier and the Validator shall be defined at the project level and shall remain unchanged throughout the development project.

NOTE The intention of this requirement is to ensure continuous and consistent verification and validation activities throughout the project. Obviously, a change in personnel cannot be fully excluded.

5.2 Personnel competence

5.2.1 Objectives

To ensure that all personnel who have responsibilities for the software are competent to discharge those responsibilities by demonstrating the ability to perform relevant tasks correctly, efficiently and consistently to a high quality and under varying conditions.

5.2.2 Requirements

5.2.2.1 The key competencies required for each role in the software development are defined in Annex B. If additional experience, capabilities or qualifications are required for a role in the software lifecycle, these shall be defined in the Software Quality Assurance Plan.

5.2.2.2 Documented evidence of personnel competence, including technical knowledge, qualifications, relevant experience and appropriate training, shall be maintained by the supplier's organization in order to demonstrate appropriate organization.

5.2.2.3 The organization shall maintain procedures to manage the competence of personnel to suit appropriate roles in accordance to existing quality standards.

5.2.2.4 Once it has been proved to the satisfaction of an Assessor or by a certification that competence has been demonstrated for all personnel appointed in various roles, each individual will need to show continuous maintenance and, if needed, development of competence. This could be demonstrated by keeping a logbook showing the activity is being regularly carried out correctly, and that additional training is being undertaken in accordance with EN ISO 9001 and ISO/IEC 90003:2014, 6.2.2 "Competence, awareness and training".

5.3 Lifecycle issues and documentation

5.3.1 Objectives

5.3.1.1 To structure the development of the software into defined phases and activities.

5.3.1.2 To record all information pertinent to the software throughout the lifecycle of the software.

5.3.2 Requirements

5.3.2.1 A lifecycle model for the development of software shall be selected. It shall be detailed in the Software Quality Assurance Plan in accordance with 6.5.

Two examples of lifecycle models are shown in Figure 3 and Figure 4.

5.3.2.2 The lifecycle model shall take into account the possibility of iterations within and between phases.

5.3.2.3 Quality Assurance procedures shall run in parallel with lifecycle activities and use the same terminology.

5.3.2.4 The Software Quality Assurance Plan, Software Verification Plan, Software Validation Plan and Software Configuration Management Plan shall be drawn up at the start of the project and maintained throughout the software development lifecycle.

5.3.2.5 All activities to be performed during a phase shall be defined and planned prior to the commencement of the phase.

5.3.2.6 All documents shall be structured to allow continued expansion in parallel with the development process.

5.3.2.7 For each document, traceability shall be provided in terms of a unique reference number and a defined and documented relationship with other documents.

5.3.2.8 Each term, acronym or abbreviation shall have the same meaning in every document. If, for historical reasons, this is not possible, the different meanings shall be listed and the references given.

5.3.2.9 Except for documents relating to pre-existing software (see 7.3.4.7), each document shall be written according to the following rules:

- it shall contain or implement all applicable conditions and requirements of the preceding document with which it has a hierarchical relationship;
- it shall not contradict the preceding document.

5.3.2.10 Each item or concept shall be referred to by the same name or description in every document.

5.3.2.11 The contents of all documents shall be recorded in a form appropriate for manipulation, processing and storage.

5.3.2.12 When documents which are produced by independent roles are combined into a single document, the relation to the parts produced by any independent role shall be traced within the document.

5.3.2.13 Documents may be combined or divided in accordance with 5.3.2.12. Some development steps may be combined, divided or, when justified, eliminated, at the discretion of the Project Manager and with the agreement of the Validator. It is permitted to automatically generate documents from databases or modelling tools. By producing documents from databases or modelling tools it is mandatory that the traceability between document version and database version is preserved.

NOTE If the database is not longer available the generated documents can be used as the basis for software maintenance.

5.3.2.14 Where any alternative lifecycle or documentation structure is adopted it shall be established that it meets all the objectives and requirements of this European Standard.

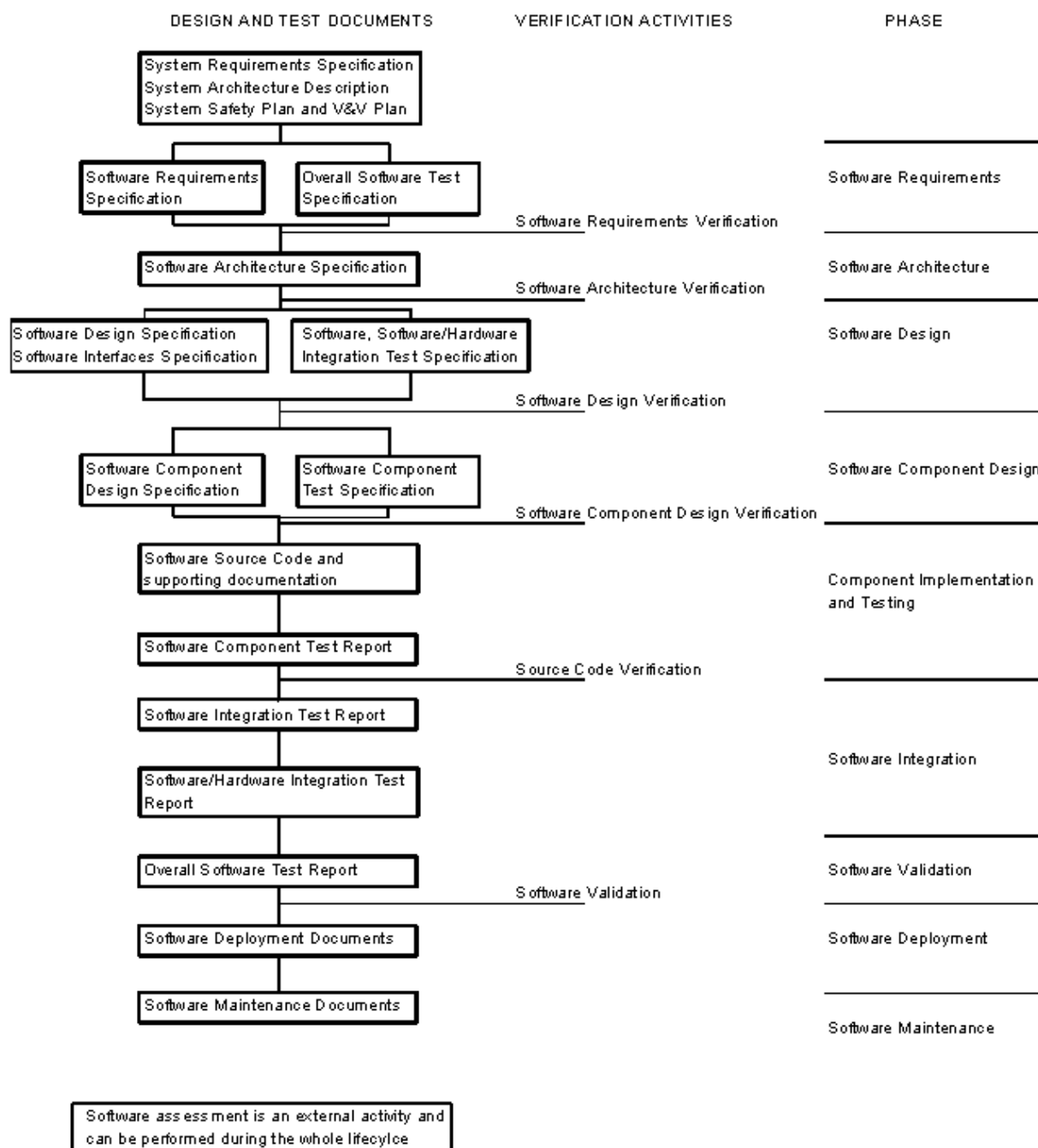


Figure 3 — Illustrative Development Lifecycle 1

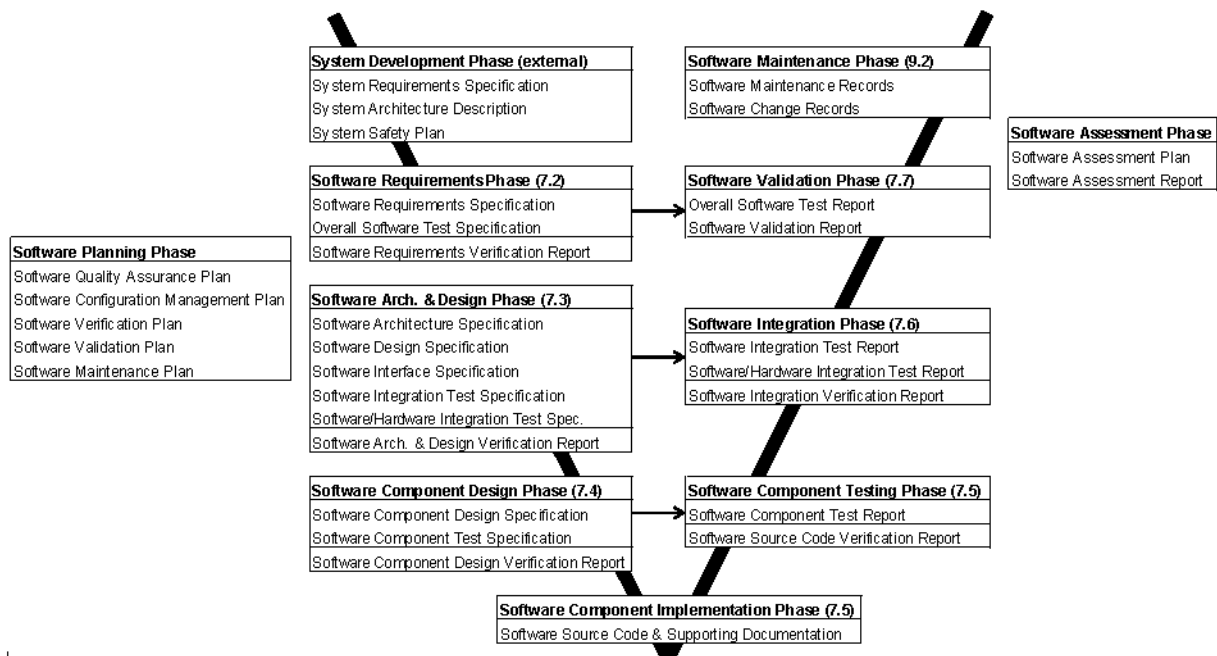


Figure 4 — Illustrative Development Lifecycle 2

6 Software assurance

6.1 Software testing

6.1.1 Objective

The objective of software testing, as performed by the Tester and/or Integrator, is to ascertain the behaviour or performance of software against the corresponding test specification to the extent achievable by the selected test coverage.

6.1.2 Input documents

- 1) All necessary System, Hardware and Software Documentation as specified in the Software Verification Plan.

6.1.3 Output documents

- 1) Overall Software Test Specification
- 2) Overall Software Test Report
- 3) Software Integration Test Specification
- 4) Software Integration Test Report
- 5) Software/Hardware Integration Test Specification
- 6) Software/Hardware Integration Test Report
- 7) Software Component Test Specification

8) Software Component Test Report

6.1.4 Requirements

6.1.4.1 Tests performed by other parties such as the Requirements Manager, Designer or Implementer, if fully documented and complying with the following requirements, may be accepted by the Verifier.

6.1.4.2 Measurement equipment used for testing shall be calibrated appropriately. Any tools, hardware or software, used for testing shall be shown to be suitable for the purpose.

6.1.4.3 Software testing shall be documented by a Test Specification and a Test Report, as defined in the following.

6.1.4.4 Each Test Specification shall document the following:

- a) test objectives;
- b) test cases, test data and expected results;
- c) types of tests to be performed;
- d) test environment, tools, configuration and programs;
- e) test criteria on which the completion of the test will be judged;
- f) the criteria and degree of test coverage to be achieved;
- g) the roles and responsibilities of the personnel involved in the test process;
- h) the requirements which are covered by the test specification;
- i) the selection and utilization of the software test equipment.

6.1.4.5 A Test Report shall be produced as follows:

- a) the Test Report shall mention the Tester names, state the test results and whether the test objectives and test criteria of the Test Specification have been met. Failures shall be documented and summarized;
- b) test cases and their results shall be recorded, preferably in a machine-readable form for subsequent analysis;
- c) tests shall be repeatable and, if practicable, be performed by automatic means;
- d) test scripts for automatic test execution shall be verified;
- e) the identity and configuration of all items involved (hardware used, software used, equipment used, equipment calibration, as well as version information of the test specification) shall be documented;
- f) an evaluation of the test coverage and test completion shall be provided and any deviations noted.

6.2 Software verification**6.2.1 Objective**

The objective of software verification is to examine and arrive at a judgment based on evidence that output items (process, documentation, software or application) of a specific development phase fulfil the requirements and plans with respect to completeness, correctness and consistency. These activities are managed by the Verifier.

6.2.2 Input documents

- 1) All necessary System, Hardware and Software Documentation.

6.2.3 Output documents

- 1) Software Verification Plan
- 2) Software Verification Report(s)

6.2.4 Requirements

6.2.4.1 Verification shall be documented by at least a Software Verification Plan and one or more (process-related) Verification Reports.

6.2.4.2 A Software Verification Plan shall be written, under the responsibility of the Verifier, on the basis of the necessary documentation.

Requirements from 6.2.4.3 to 6.2.4.9 refer to the Software Verification Plan.

6.2.4.3 The Software Verification Plan shall describe the activities to be performed to ensure proper verification and that particular design or other verification needs are suitably provided for.

6.2.4.4 During development (and depending upon the size of the system) the plan may be sub-divided into a number of child documents, as the detailed needs of verification become clearer.

6.2.4.5 The Software Verification Plan shall document all the criteria, techniques and tools to be used in the verification process. The Software Verification Plan shall include techniques and measures chosen from Table A.5, Table A.6, Table A.7 and Table A.8. The selected combination shall be justified as a set satisfying 4.8, 4.9 and 4.10.

6.2.4.6 The Software Verification Plan shall describe the activities to be performed to ensure correctness and consistency with respect to the input to that phase. These include reviewing, testing and integration.

6.2.4.7 In each development phase it shall be shown that the functional, performance and safety requirements are met.

6.2.4.8 The results of each verification shall be retained in a format defined or referenced in the Software Verification Plan.

6.2.4.9 The Software Verification Plan shall address the following:

- a) the selection of verification strategies and techniques (to avoid undue complexity in the assessment of the verification and testing, preference shall be given to the selection of techniques which are in themselves readily analysable);
- b) selection of techniques from Table A.5, Table A.6, Table A.7 and Table A.8;
- c) the selection and documentation of verification activities;
- d) the evaluation of verification results gained;
- e) the evaluation of the safety and robustness requirements;
- f) the roles and responsibilities of the personnel involved in the verification process;
- g) the degree of the functional based test coverage required to be achieved;

- h) the structure and content of each verification step, especially for the Software Requirement Verification (7.2.4.22), Software Architecture and Design Verification (7.3.4.41, 7.3.4.42), Software Components Verification (7.4.4.13), Software Source Code Verification (7.5.4.10) and Integration Verification (7.6.4.13) in a way that facilitates review against the Software Verification Plan.

6.2.4.10 Entry intentionally left empty

6.2.4.11 Once the Software Verification Plan has been established, verification shall address:

- a) that the Software Verification Plan meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 6.2.4.3 to 6.2.4.9,
- b) the internal consistency of the Software Verification Plan.

The results shall be recorded.

6.2.4.12 Any Software Verification Reports shall be written, under the responsibility of the Verifier, on the basis of the input documents. These reports can be partitioned for clarity and convenience, and shall follow the Software Verification Plan. The requirement in 6.2.4.13 refers to the Software Verification Reports.

6.2.4.13 Each Software Verification Report shall document the following:

- a) the identity and configuration of the items verified, as well as the Verifier names;
- b) items which do not conform to the specifications;
- c) components, data, structures and algorithms poorly adapted to the problem;
- d) detected errors or deficiencies;
- e) the fulfilment of, or deviation from, the Software Verification Plan (in the event of deviation the Verification Report shall explain whether the deviation is critical or not);
- f) assumptions if any;
- g) a summary of the verification results.

6.3 Software validation

6.3.1 Objective

6.3.1.1 The objective of software validation is to demonstrate that the processes and their outputs are such that the software is of the defined software integrity level, fulfils the software requirements and is fit for its intended application. This activity is performed by the Validator.

6.3.1.2 The main validation activities are to demonstrate by analysis and/or testing that all the software requirements are specified, implemented, tested and fulfilled as required by the applicable software integrity level, and to evaluate the safety criticality of all anomalies and non-conformities based on the results of reviews, analyses and tests.

6.3.2 Input documents

All system, hardware and software documentation as specified in this European Standard.

6.3.3 Output documents

- 1) Software Validation Plan
- 2) Software Validation Report

6.3.4 Requirements

6.3.4.1 The Software Validation activities shall be developed and performed, with their results evaluated, by a Validator with an appropriate level of independence as defined in 5.1.

6.3.4.2 Validation shall be documented with, at least, a Software Validation Plan and a Software Validation Report, as defined in the following.

6.3.4.3 A Software Validation Plan shall be written, under the responsibility of the Validator, on the basis of the input documents.

Requirements from 6.3.4.4 to 6.3.4.6 refer to the Software Validation Plan.

6.3.4.4 The Software Validation Plan shall include a summary justifying the validation strategy chosen. The justification shall include consideration, according to the required software integrity level, of:

- a) manual or automated techniques or both,
- b) static or dynamic techniques or both,
- c) analytical or statistical techniques or both,
- d) testing in a real or simulated environment or both.

6.3.4.5 The Software Validation Plan shall identify the steps necessary to demonstrate the adequacy of any Software Specification in fulfilling the requirements set out in the System Requirements Specification.

6.3.4.6 The Software Validation Plan shall identify the steps necessary to demonstrate the adequacy of the Overall Software Test Specification as a test against the Software Requirements Specification.

6.3.4.7 A Software Validation Report shall be written, under the responsibility of the Validator, on the basis of the input documents.

Requirements from 6.3.4.8 to 6.3.4.11 refer to the Software Validation Report.

6.3.4.8 The results of the validation shall be documented in the Software Validation Report.

6.3.4.9 The Validator shall check that the verification process is complete.

6.3.4.10 The Software Validation Report shall fully state the software baseline that has been validated.

6.3.4.11 The Validation Report shall clearly identify any known deficiencies in the software and the impact these may have on the use of the software.

6.3.4.12 Entry intentionally left empty

6.3.4.13 Once the Software Validation Plan has been established, verification shall address:

- a) that the Software Validation Plan meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 6.3.4.4 to 6.3.4.6,
- b) the internal consistency of the Software Validation Plan.

The results shall be recorded.

6.3.4.14 Once the Software Validation Report has been established, verification shall address:

- a) that the Software Validation Report meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 6.3.4.8 to 6.3.4.11 and 7.7.4.7 to 7.7.4.11,

b) the internal consistency of the Software Validation Report.

The results shall be recorded.

6.3.4.15 The Validator shall be empowered to require or perform additional reviews, analyses and tests.

6.3.4.16 The software shall only be released for operation after authorization by the Validator.

6.3.4.17 Simulation and modelling may be used to supplement the validation process.

6.4 Software assessment

6.4.1 Objective

6.4.1.1 To evaluate that the lifecycle processes and their outputs are such that the software is of the defined safety integrity levels 1-4 and is fit for its intended application.

6.4.1.2 For basic integrity, requirements of this standard shall be fulfilled but no assessment will be required.

NOTE For basic integrity, the requirements of Subclauses 6.4.2 to 6.4.4 do not need to be fulfilled.

6.4.2 Input documents

- 1) System Safety Requirements (see 4.1, NOTE 2)
- 2) Software Requirements Specification
- 3) All other documents necessary to carry out the assessment process.

6.4.3 Output documents

- 1) Software Assessment Plan
- 2) Software Assessment Report

6.4.4 Requirements

6.4.4.1 The assessment of the software shall be carried out by an Assessor who is independent as described in 5.1.2.6 and 5.1.2.7.

6.4.4.2 Software with a Software Assessment Report from another Assessor does not have to be an object of a new assessment. The Assessor shall check that the software is fit for its intended use within the intended environment, and that the former assessment stated the software has achieved a safety integrity level at least equal to the required level.

6.4.4.3 The Assessor shall have access to all project-related documentation throughout the development process.

6.4.4.4 A Software Assessment Plan shall be written, under the responsibility of the Assessor, on the basis of the input documents from 6.4.2. Where appropriate, an existing documented generic Software Assessment Plan or procedure may be used. The requirement in 6.4.4.5 refers to the Software Assessment Plan.

6.4.4.5 The Software Assessment Plan shall include the following scope:

- a) aspects with which the assessment deals;

- b) activities throughout the assessment process and their sequential link to engineering activities;
- c) documents to be taken into consideration;
- d) statements on pass/fail criteria and the way to deal with non-conformance cases;
- e) requirements with regard to content and form of the Software Assessment Report.

6.4.4.6 The Software Assessment Plan shall be verified on the basis of the input documents from 6.4.2.

The requirement in 6.4.4.7 refers to this verification.

6.4.4.7 Once the Software Assessment Plan has been established, verification shall address:

- a) that the Software Assessment Plan meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 6.4.4.5,
- b) the internal consistency of the Software Assessment Plan,
- c) that the assessment plan fits the needs of the project (e.g. in terms of schedule and scope of the assessment).

The results shall be recorded.

6.4.4.8 The Assessor shall assess that the software of the system is fit for its intended purpose and responds correctly to safety issues derived from the System Safety Requirements.

6.4.4.9 The Assessor shall assess if an appropriate set of techniques from Annex A, suitable for the intended development, has been selected and applied in accordance to the required safety integrity level.

Moreover, the Assessor shall consider the extent to which each technique from Annex A is applied, i.e. whether it is applied to all or to only part of the software, and also look for evidence that it is properly applied.

6.4.4.10 The Assessor shall assess the configuration and change management system and the evidence on its use and application.

6.4.4.11 The Assessor shall review the evidence of the competency of the project staff according to Annex B and shall assess the organization for the software development according to 5.1.

6.4.4.12 For any software containing safety-related application conditions, the Assessor shall check for noted deviations, non-compliances to requirements and recorded non-conformities if these have an impact on safety, and make a judgment whether the justification from the project is acceptable. The result shall be stated in the Software Assessment Report.

6.4.4.13 The Assessor shall assess the verification and validation activities and the supporting evidence.

6.4.4.14 The Assessor shall agree the scope and contents of the Software Validation Plan. This agreement shall also make a statement concerning the presence of the Assessor during testing.

6.4.4.15 The Assessor may carry out audits and inspections (e.g. witnessing tests) throughout the entire development process. The Assessor may ask for additional verification and validation work.

NOTE It is of advantage to involve the Assessor early in the project.

6.4.4.16 A Software Assessment Report shall be written under the responsibility of the Assessor. Requirements from 6.4.4.17 to 6.4.4.19 refer to the Software Assessment Report.

6.4.4.17 The Software Assessment Report shall meet the requirements of the Software Assessment Plan and provide a conclusion and recommendations.

6.4.4.18 The Assessor shall record his/her activities as a consistent base for the Software Assessment Report. These shall be summarized in the Software Assessment Report.

6.4.4.19 The Assessor shall identify and evaluate any non-conformity with the requirements of this European Standard and judge the impact on the final result. These non-conformities and their judgments shall be listed in the Software Assessment Report.

6.5 Software quality assurance

6.5.1 Objectives

6.5.1.1 To identify, monitor and control all those activities, both technical and managerial, which are necessary to ensure that the software achieves the quality required. This is necessary to provide the required qualitative defence against systematic faults and to ensure that an audit trail can be established to allow verification and validation activities to be undertaken effectively.

6.5.1.2 To provide evidence that the above activities have been carried out.

6.5.2 Input documents

All the documents available at each stage of the lifecycle.

6.5.3 Output documents

- 1) Software Quality Assurance Plan
- 2) Software Configuration Management Plan, if not available at system level

6.5.4 Requirements

6.5.4.1 All the plans shall be issued at the beginning of the project and updated during the lifecycle.

6.5.4.2 The organizations taking part in the software development shall implement and use a Quality Assurance System compliant with EN ISO 9000:2015, to support the requirements of this European Standard (see Table A.9). EN ISO 9001 certification is highly recommended.

6.5.4.3 A Software Quality Assurance Plan shall be written, under the responsibility of the Verifier, on the basis of the input documents from 6.5.2.

The requirements from 6.5.4.4 to 6.5.4.6 refer to the Software Quality Assurance Plan.

6.5.4.4 A Software Quality Assurance Plan shall be written and shall be specific to the project. It shall implement the requirements of 6.5.4.5.

6.5.4.5 As a minimum, the following items shall be specified or referenced in the Software Quality Assurance Plan.

- a) Definition of the lifecycle model:
 - 1) activities and elementary tasks consistent with the plans, e.g. Safety Plan, that have been established at the System level;
 - 2) entry and exit criteria of each activity;
 - 3) inputs and outputs of each activity;
 - 4) major quality activities;

- 5) the entity responsible for each activity.
- b) Documentation structure.
- c) Documentation control:
 - 1) roles involved for writing, checking and approval;
 - 2) scope of distribution;
 - 3) archiving.
- d) Tracking and tracing of deviations.
- e) Methods, measures and tools for quality assurance according to the allocated software integrity levels (refer to Annex A).
- f) Justifications, as defined in 4.7 to 4.9, that each combination of techniques or measures selected according to Annex A is appropriate to the defined software integrity level.

Some of the Software Quality Assurance Plan required information may be contained in other documents, such as a separate Software Configuration Management Plan, a Maintenance Plan, a Software Verification Plan and a Software Validation Plan. The subclauses of the Software Quality Assurance Plan shall reference the documents in which the information is contained. In any case the content of each subclause of the Software Quality Assurance Plan shall be specified either directly or by reference to another document.

The referenced documents shall be reviewed in order to ensure they provide all the required information and that they fully address the requirements of this European Standard.

6.5.4.6 Quality assurance activities, actions, documents, etc. required by all normative subclauses of this European Standard shall be specified or referenced in the Software Quality Assurance Plan and tailored to the specific project.

6.5.4.7 Entry intentionally left empty

6.5.4.8 Once the Software Quality Assurance Plan has been established, verification shall address:

- a) that the Software Quality Assurance Plan meets the general requirements for readability and traceability in 5.3.2.7 to 5.3.2.10 and in 6.5.4.14 to 6.5.4.17 as well as the specific requirements in 6.5.4.4 to 6.5.4.6,
- b) the internal consistency of the Software Quality Assurance Plan.

The results shall be recorded.

6.5.4.9 Each planning document shall have a paragraph specifying details about its own updating throughout the project: frequency, responsibility, method.

6.5.4.10 Each software document and deliverable shall be placed under configuration control from the time of its first release.

6.5.4.11 Changes to all items under Configuration Management Control shall be authorized and recorded.

6.5.4.12 In addition to software development, the Configuration Management System shall also cover the software development environment used during the full lifecycle.

This extension, necessary for the reproducibility of the development and for the maintenance activities, shall include all the tools, translators, data and test files, parameterization files, and supporting hardware platforms.

6.5.4.13 The supplier shall establish, document and maintain procedures for control of the external suppliers, including:

- methods and relevant records to ensure that software provided by external suppliers adheres to established requirements. Previously developed software shall be ensured to be compliant with the required software integrity level and dependability. New software shall be developed and maintained in conformity with the Software Quality Assurance Plan of the Supplier or with a specific Software Quality Assurance Plan prepared by the external supplier in accordance with the Software Quality Assurance Plan of the Supplier,
- methods and relevant records to ensure that the requirements provided to the External Supplier are adequate and complete.

6.5.4.14 Traceability to requirements shall be an important consideration in the validation of a safety-related system and means shall be provided to allow this to be demonstrated throughout all phases of the lifecycle.

6.5.4.15 Within the context of this European Standard, and to a degree appropriate to the specified software integrity level, traceability shall particularly address:

- a) traceability of requirements to the design or other objects which fulfil them,
- b) traceability of design objects to the implementation objects which instantiate them,
- c) traceability of requirements and design objects to the tests (component, integration, overall test) and analyses that verify them.

Traceability shall be the subject of configuration management.

6.5.4.16 In special cases, e.g. pre-existing software or prototyped software, traceability may be established after the implementation and/or documentation of the code, but prior to verification/validation. In these cases, it shall be shown that verification/validation is as effective as it would have been with traceability over all phases.

6.5.4.17 Objects of requirements, design or implementation that cannot be adequately traced shall be demonstrated to have no bearing upon the safety or integrity of the system.

6.5.4.18 Where different parts of a specific software development are being undertaken by separate and distinct organizations there should be a clear definition of the tasks being undertaken by each organization. The documentation produced by each organization should provide sufficient references to the documentation of the other participating organizations for an understanding of the overall software development to be clearly and readily obtained by the organizations themselves or a reviewing entity. The sub-contracting organization remains responsible for compliance with this European Standard.

The complexity of the development should not obscure the tasks and responsibilities of the contributing organizations. Where more than one distinct organization contributes to the development of the software, a description of the overall software development including a definition of the organizational interfaces between the involved parties should be contained in the Software Quality Assurance Plan.

EXAMPLE: A rolling stock manufacturer subcontracts the development of some of the software components to an external supplier. The external supplier produces the software components along with design and design verification documentation. The external supplier in this example does not have the facilities to extensively test the software in its target environment. The integration, overall testing and validation in this example is carried out by the rolling stock manufacturer. In this case the various test reports and validation documentation are provided by the coordinating organization, the rolling stock manufacturer. However, the external supplier in this example is requested to contribute to the specification of the tests. Whatever the circumstances, it is clearly understood by all parties what their responsibilities and tasks are. In this example the contribution of each organization is specified in the Software Quality Assurance Plan of the rolling stock manufacturer. The external supplier in this example references the relevant documentation, including the Software Quality Assurance Plan, of the rolling stock manufacturer, in their own Software Quality Assurance Plan.

6.6 Modification and change control

6.6.1 Objectives

6.6.1.1 To ensure that the software performs as required, preserving the software integrity and dependability when modifying the software.

6.6.1.2 These objectives are managed by the Configuration Manager.

6.6.2 Input documents

- 1) Software Quality Assurance Plan
- 2) Software Configuration Management Plan
- 3) All relevant design, development and analysis documentation
- 4) Change Requests
- 5) Change impact analysis and authorization

6.6.3 Output documents

- 1) All changed input documents
- 2) Software Change records (see 9.2.4.10)
- 3) New Configuration records

6.6.4 Requirements

6.6.4.1 The Change Management Process shall define at least the following aspects:

- a) the documentation needed for problem reporting and/or corrective actions, with the aim of giving feedback to the responsible management;
- b) analysis of the information collected in the problem reports to identify its causes;
- c) the practices to be followed for reporting, tracking and resolving problems identified both during the development phase and during software maintenance;
- d) the specific organizational responsibilities with regard to development and software maintenance;
- e) how to apply controls to ensure that corrective actions are taken and that they are effective;
- f) impact analysis of the effect of the changes on the software component under development or already delivered;
- g) impact analysis shall state the re-verification, re-validation and re-assessment necessary for the change;
- h) where multiple changes are applied, the impact analysis shall consider the cumulative impact;

NOTE Several changes may cumulatively require a complete re-test.

- i) authorization before implementation.

6.6.4.2 All changes shall initiate a return to an appropriate phase of the lifecycle. All subsequent phases shall then be carried out in accordance with the procedures specified for the specific phases in accordance with the requirements in this European Standard.

6.7 Support tools and languages

6.7.1 Objectives

The objective is to provide evidence that potential failures of tools do not adversely affect the integrated toolset output in a safety-related manner that is undetected by technical and/or organizational measures outside the tool. To this end, software tools are categorized into three classes namely, T1, T2 and T3 respectively (see definitions in 3.1).

When tools are being used as a replacement for manual operations, the evidence of the integrity of tools output can be adduced by the same process steps as if the output was done in manual operation. These process steps might be replaced by alternative methods if an argumentation on the integrity of tools output is given and the software integrity level is not decreased by the replacement.

6.7.2 Input documents

Tools specification or manual.

6.7.3 Output documents

Tools validation report (when needed see 6.7.4.4 or 6.7.4.6).

6.7.4 Requirements

6.7.4.1 Software tools shall be selected as a coherent part of the software development activities and compatible with the needs of the application. The availability of suitable tools to supply the services that are necessary over the whole lifetime of the software shall be considered.

Appropriate tools to support the development of software should be used in order to increase the integrity of the software by reducing the likelihood of introducing or not detecting faults during the development. Examples of tools relevant to the phases of the software development lifecycle include:

- a) transformation or translation tools that convert a software or design representation (e.g. text or a diagram) from one abstraction level to another: design refinement tools, compilers, assemblers, linkers, binders, loaders and code generation tools,
- b) verification and validation tools such as static code analysers, test coverage monitors, theorem proving assistants, simulators and model checkers,
- c) diagnostic tools used to maintain and monitor the software under operating conditions,
- d) infrastructure tools such as development support systems,
- e) configuration control tools such as version control tools,
- f) application data tools that produce or maintain data which are required to define parameters and to instantiate system functions e.g. function parameters, instrument ranges, alarm and trip levels, output states to be adopted at failure.

The selected tools should be able to cooperate. In this context, tools cooperate if the outputs from one tool have suitable content and format for automatic input to a subsequent tool, thus minimizing the possibility of introducing human error in the reworking of intermediate results.

6.7.4.2 The selection of the tools in classes T2 used for SIL1 – SIL4, and T3 used for SIL1 – SIL4 shall be justified (see 7.3.4.12). The justification shall include the identification of potential failures which can be injected into the tools output and the measures to avoid or handle such failures.

6.7.4.3 All tools in classes T2 used for SIL1 – SIL4, and T3 shall have a specification or manual which clearly defines the behaviour of the tool and any instructions or constraints on its use.

6.7.4.4 For each tool in class T3 used for SIL1 – SIL4, evidence shall be available that the output of the tool conforms to the specification of the output or failures in the output are detected. Evidence may be based on the same steps necessary for a manual process as a replacement for the tool and an argument presented if these steps are replaced by alternatives (e.g. validation of the tool). Evidence may also be based on:

- a) a suitable combination of history of successful use in similar environments and for similar applications (within the organization or other organizations),
- b) tool validation as specified in 6.7.4.5,
- c) diverse redundant code which allows the detection and control of failures resulting in faults introduced by a tool,
- d) compliance with the safety integrity levels derived from the risk analysis of the process and procedures including the tools,
- e) other appropriate methods for avoiding or handling failures introduced by tools.

NOTE 1 A version history may provide assurance of maturity of the tool, and a record of the errors / ambiguities associated with its use in the environment.

NOTE 2 The evidence listed for T3 can also be used for T2 tools in judging the correctness of their results.

6.7.4.5 For each tool in class T3 used for SIL1 – SIL4, the results of tool validation shall be documented covering the following results:

- a) a record of the validation activities;
- b) the version of the tool manual being used;
- c) the tool functions being validated;
- d) tools and equipment used to perform the validation;
- e) the results of the validation activity; the documented results of validation shall state either that the software has passed the validation or the reasons for its failure;
- f) test cases and their results for subsequent analysis;
- g) discrepancies between expected and actual results.

NOTE Entry 6.7.4.5 only applies if tool validation is selected as evidence to fulfil 6.7.4.4.

6.7.4.6 For each tool in class T3 used for SIL1 – SIL4, where the conformance evidence of 6.7.4.4 is unavailable, there shall be effective measures to control failures of the executable safety-related software that result from faults that are attributable to the tool.

NOTE 1 An example is the generation of diverse redundant code which allows the detection and control of failures resulting in faults introduced by a translator.

NOTE 2 As an example, the fitness for purpose of a non-trusted compiler can be justified as follows.

The object code produced by the compiler has been subjected to a combination of tests, checks and analyses which are capable of ensuring the correctness of the code to the extent that it is consistent with the target safety integrity level. In particular, the following applies to all tests, checks and analyses.

- Testing has been shown to have a sufficiently high coverage of the implemented code. If there is any code unreachable by testing, it has been shown by checks or analyses that the function concerned is executed correctly when the code is reached on the target.
- Checks and analyses have been applied to the object code and shown to be capable of detecting the types of errors which might result from a defect in the compiler.
- No more translation with the compiler has taken place after testing, checking and analysis.
- If further compilation or translation is carried out, all tests, checks and analyses will be repeated.

6.7.4.7 For each tool in class T3 used for SIL1 – SIL4, the software or design representation (including a programming language) selected shall:

- a) have a translator which has been evaluated for fitness for purpose including, where appropriate, evaluated against the international or national standards,
- b) match the characteristics of the application,
- c) contain features that facilitate the detection of design or programming errors,
- d) support features that match the design method.

A programming language is one of a class of representations of software or design. A Translator converts a software or design representation (e.g. text or a diagram) from one abstraction level to another level. Examples of Translators include: design refinement tools, compilers, assemblers, linkers, binders, loaders and code generation tools.

The evaluation of a Translator may be performed for a specific application project, or for a class of applications. In the latter case all necessary information on the tool regarding the intended and appropriate use of the tool shall be available to the user of the tool. The evaluation of the tool for a specific project may then be reduced to checking general suitability of the tool for the project and compliance to the “specification or manual” (i.e. proper use of the tool). Proper use might include additional verification activities within the specific project.

A validation suite may be used to evaluate the fitness for purpose of a Translator according to defined criteria, which shall include functional and non-functional requirements. For the functional Translator requirements, dynamic testing may be a main validation technique. If possible an automatic testing suite shall be used.

6.7.4.8 For each tool in class T3 used for SIL1 – SIL4, where 6.7.4.7 cannot be fully satisfied, the fitness for purpose of the language, and any additional measures which address any identified shortcomings of the language shall be justified and evaluated.

NOTE See NOTE 2 from 6.7.4.6.

6.7.4.9 For each tool in class T3 used for SIL1 – SIL4, where automatic code generation or similar automatic translation takes place, the suitability of the automatic Translator for the application shall be evaluated at the point in the development lifecycle where development support tools are selected.

6.7.4.10 Configuration management shall ensure that for tools in classes T2 used for SIL1 – SIL4, and T3, only justified versions are used.

6.7.4.11 For tools in classes T2 used for SIL1 – SIL4, and T3, each new version of a tool that is used shall be justified (see Table 1). This justification may rely on evidence provided for an earlier version if sufficient evidence is provided that:

- a) the functional differences (if any) will not affect tool compatibility with the rest of the toolset,
- b) the new version is unlikely to contain significant new, unknown faults.

NOTE Evidence that the new version is unlikely to contain significant new unknown faults may be based on a credible identification of the changes made, and on an analysis of the verification and validation actions performed.

6.7.4.12 The relation between the tool classes and the applicable numbered entries is defined within Table 1.

Table 1 — Relation between tool class and applicable numbered entries

Tool class	Applicable numbered entries for SIL1 – SIL4	Applicable numbered entries for basic integrity
T1	6.7.4.1	6.7.4.1
T2	6.7.4.1, 6.7.4.2, 6.7.4.3, 6.7.4.10, 6.7.4.11	6.7.4.1
T3	6.7.4.1, 6.7.4.2, 6.7.4.3, 6.7.4.4, 6.7.4.5, 6.7.4.6, 6.7.4.7, 6.7.4.8, 6.7.4.9, 6.7.4.10, 6.7.4.11	6.7.4.1, 6.7.4.3, 6.7.4.10, 6.7.4.11

7 Software development

7.1 Lifecycle and documentation for software

7.1.1 Objectives

To provide a description of the software itself, from the higher levels of abstraction down to the detailed refinements, in order to create a frame for the demonstration of the achieved safety as well as for future maintenance actions.

7.1.2 Requirements

7.1.2.1 To the extent required by the software integrity level, the requirements related to documents listed in Table A.1 shall be fulfilled.

NOTE The options given by Entries 5.3.2.12, 5.3.2.13 and 5.3.2.14 can be applied.

7.1.2.2 The sequence of deliverable documents as they are described in Table A.1 reflects an ideal linear waterfall model. This model is however not intended to be a reference in the sense of schedule and linkage of activities, as it would usually be difficult to achieve a strict compliance in practice. Phases can overlap but verification and validation activities shall demonstrate the consistency of inputs and outputs (documents and software) within and between the phases.

7.2 Software requirements

7.2.1 Objectives

7.2.1.1 To describe a complete set of requirements for the software meeting all System and Safety Requirements and provide a comprehensive set of documents for each subsequent phase.

7.2.1.2 To describe the Overall Software Test Specification.

7.2.2 Input documents

- 1) System Requirements Specification
- 2) intentionally left empty