IEC TS 61508-3-2

Edition 1.0 2024-08

TECHNICAL SPECIFICATION

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

Functional safety of electrical/electronic/programmable electronic safety-related systems –
Part 3-2: Requirements and guidance in the use of mathematical and logical techniques for establishing exact properties of software and its documentation

IEC TS 61508-3-2:2024-08(en)

THIS PUBLICATION IS COPYRIGHT PROTECTED
Copyright © 2024 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester. If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

IEC Secretariat
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
info@iec.ch
www.iec.ch

About the IEC
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigendum or an amendment might have been published.

IEC publications search - webstore.iec.ch/advsearchform
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee, …). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and once a month by email.

IEC Customer Service Centre - webstore.iec.ch/csc
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: sales@iec.ch.

IEC Products & Services Portal - products.iec.ch
Discover our powerful search engine and read freely all the publications previews, graphical symbols and the glossary. With a subscription you will always have access to up to date content tailored to your needs.

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 500 terminological entries in English and French, with equivalent terms in 25 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

IEC TS 61508-3-2

Edition 1.0 2024-08

TECHNICAL SPECIFICATION

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3-2: Requirements and guidance in the use of mathematical and logical techniques for establishing exact properties of software and its documentation

INTERNATIONAL ELECTROTECHNICAL COMMISSION

ICS 25.040.40

ISBN 978-2-8322-9565-6

Warning! Make sure that you obtained this publication from an authorized distributor.

® Registered trademark of the International Electrotechnical Commission

CONTENTS

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

FUNCTIONAL SAFETY OF ELECTRICAL/ELECTRONIC/PROGRAMMABLE ELECTRONIC SAFETY-RELATED SYSTEMS –

Part 3-2: Requirements and guidance in the use of mathematical and logical techniques for establishing exact properties of software and its documentation

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) IEC draws attention to the possibility that the implementation of this document may involve the use of (a) patent(s). IEC takes no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, IEC had not received notice of (a) patent(s), which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at https://patents.iec.ch. IEC shall not be held responsible for identifying any or all such patent rights.

IEC TS 61508-3-2 has been prepared by subcommittee 65A: System aspects, of IEC technical committee 65: Industrial-process measurement, control and automation. It is a Technical Specification.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

The text of this Technical Specification is based on the following documents:

Draft	Report on voting
65A/1113/DTS	65A/1143/RVDTS

Full information on the voting for its approval can be found in the report on voting indicated in the above table.

The language used for the development of this Technical Specification is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement, available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/publications.

A list of all parts in the IEC 61508 series, published under the general title *Functional safety of electrical/electronic/programmable electronic safety-related systems*, can be found on the IEC website.

The committee has decided that the contents of this document will remain unchanged until the stability date indicated on the IEC website under webstore.iec.ch in the data related to the specific document. At this date, the document will be

- reconfirmed,
- withdrawn, or
- revised.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

INTRODUCTION

IEC 61508-1:2010 through IEC 61508-7:2010 forms the series of basic standards for the functional safety of electric, electronic and programmable electronic systems (E/E/PE systems). It covers the life cycle of these systems. The major part of the functionality of such systems is often implemented in software. IEC 61508-3:2010 sets software requirements.

IEC 61508-3:2010 Annex A (normative) and Annexes B and C (informative) contain tables listing various techniques and measures, and provide some guidance to the selection of such techniques for different safety integrity levels (SIL). It lists general categories and gives different levels of recommendation for these, such as "not recommended", "recommended" or "highly recommended", as well as more specific techniques for various phases of software development.

These techniques and measures are a mix of generic and specific. The phrase "Formal Methods" as used in IEC 61508-3 refers to the use of mathematical and logical techniques for specifying, assessing, designing and verifying software. Today, such methods are available for specifying requirements, for the assessment of the design, for checking source code and object code and for the derivation of test suites, and for monitoring the correct operation of software at runtime. In this document, we refer to these methods by using the description as mathematical and logical techniques (M< sometimes doubled as M&LT techniques). Some of the M&LT techniques in this document are not restricted to software development, being equally applicable to other digital-system-based engineering technologies. None of the M&LT techniques are limited to the domain of safety-related software systems, although in this document only safety-related applications of M&LT techniques are explicitly addressed.

Use of the recommended methods of IEC 61508-3:2010, Annexes A, B and C do not rule out, for example, susceptibility of the software to run-time failure. State of the art in software development enables various types of run-time failures to be ruled out through rigorous development of the software. It is possible using techniques identified here to assure freedom from many types of software run-time failures.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

FUNCTIONAL SAFETY OF ELECTRICAL/ELECTRONIC/PROGRAMMABLE ELECTRONIC SAFETY-RELATED SYSTEMS –

Part 3-2: Requirements and guidance in the use of mathematical and logical techniques for establishing exact properties of software and its documentation

1 Scope

This Technical Specification, part of the IEC 61508 series, covers the general assurance of dependable software used in critical operational-technology (OT) which is running on hardware devices which are specified as part of the OT application. It is particularly aimed at safety-related software which is being developed according to the E/E/PE software functional safety standard IEC 61508-3; in particular, the development of the software follows a Formal Safety Requirements Specification. Successful use of some or all of the assurance points specified in this document enhances the confidence that a particular piece of safety-related software meets the requirements of the SIL of the safety function which it (partially or fully) implements, and thereby increases the systematic capability of the software.

2 Normative references

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

IEC 61508-3:2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements*

IEC 61508-4:2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 4: Definitions and abbreviations*

3 Terms, definitions and abbreviations

3.1 Terms and definitions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

3.1.1
abstract interpretation
<of a computer program> static analysis of a program on abstract program states or abstract machine states that provides sound results for a given property, i.e., that never reports the property to hold if it does not hold

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

3.1.2
assurance point
<in software development using M&LT> triple, consisting of software and/or documentation (S1), another SW and/or documentation (S2), and a property P jointly of S1 and S2 such that P(S1.S2) can be formally mathematically proved

Note 1 to entry: Although a mathematical proof is formally possible at an assurance point, such a proof can be too complex, or require too many resources, to be given in its entirety and reliably checked, say by an assessor.

3.1.3
automated prover
automated theorem prover
computer program which performs inference in a formal logic between sentences of a formal language

3.1.4
automated proving
using an automated prover

3.1.5
characteristic function
<of a set or a relation> function (of variable domain) and codomain {0,1} such that its value is one when its argument belongs to the set or relation, and its value is 0 when its argument does not belong to the set or relation

3.1.6
code generator
automatic code generator
software which effects the transformation of a high-level language program or a specification into a common third- or fourth-generation-language program

3.1.7
coding standard
programming-language subsetting
<in M&LT techniques> restrictions on the constructs with which a program can be written in a high-level programming language

Note 1 to entry: The purpose of a coding standard which restricts programming-language constructs that may be used is to assure an unambiguous semantics to a program written according to the coding standard.

Note 2 to entry: A typical coding standard (n.b., the singular version of this phrase is uncommon) will ensure that known causes of unreliable behaviour in program code are avoided, e.g., pointer variables are proscribed; undefined or compiler-variable language features are avoided. Coding standards for programs written in a language without strong data typing might well ensure that the anticipated or specified range of input or output data is explicitly checked at input or output.

Note 3 to entry: The term coding standards in general use often refers to further properties of code than subsetting.

3.1.8
compilation
translation operation that translates executable source code level (ESCL) into object code (OC)

Note 1 to entry: The definition explicitly mentions ESCL, a concept used in this document, but not generally where compilers are used and compilation is practiced.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

3.1.9
completeness
<of a formal language with a logic with respect to a given semantic property> quality of a formal language with an associated logic and a formal semantics that holds with respect to a given semantic property if every sentence of the language having this property can be shown to have this property through inference in the associated logic

Note 1 to entry: An algorithm is complete with respect to a property if it always proves the property if it holds.

3.1.10
compositional
<of a formal semantics> taking as input just a sentence of a formal language (and not, say, any indication of context, say the reference of indexicals) and which constructs a transcription purely using the parse tree of the sentence

3.1.11
computable
recursive
<computable-function theory> turing-computable

Note 1 to entry: Recursive is used here in the sense in which this term is used in Turing computability and recursive function theory [1][1], [2].

Note 2 to entry: There are in the mathematical literature other notions of "computable" than Turing-computable. Some are known to be equivalent to Turing-computable, but for some the question is open. This definition thus disambiguates use of the term "computable". Similarly, in computer science and system engineering the term "recursive" has variable meanings; this definition disambiguates use of the term.

Note 3 to entry: The term "decidable" is often used to refer to properties; the term "computable" to functions. A property is decidable if and only if its characteristic function is computable.

Note 4 to entry: "Turing computable" is a concept which is usually defined over many tens of pages of textbooks on recursion theory, often using many subsidiary concepts [1] [2]. There lacks a short definition to use here.

Note 5 to entry: There are other notions of computability in logic and computer science which are not, or not known to be, reducible to Turing-computability.

3.1.12
consistency
property of a collection of sentences of a formal language that they are not contradictory

3.1.13
contradictory
property of a collection of sentences of a formal language when their renderings are mutually exclusive, that is, they cannot all hold or be realised at the same time in the same structure

3.1.14
decidable
having a Turing-computable characteristic function

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

[1] Numbers in square brackets refer to the Bibliography.

3.1.15
element
part of a subsystem comprising a single component or any group of components that performs one or more element safety functions

Note 1 to entry: An element may comprise hardware and/or software.

Note 2 to entry: A typical element is a sensor, programmable controller or final element

[SOURCE: IEC 61508-4:2010, 3.4.5]

3.1.16
element safety function
that part of a safety function which is implemented by an element

[SOURCE: IEC 61508-4:2010, 3.5.3]

3.1.17
executable source code level
ESCL
source code at which the exact behaviour of the software is completely described

3.1.18
formal inference
<in a logic> derivation of a sentence from premises using the explicit formal rules of inference of the logic

3.1.19
formal language
<in M&LT> language with a defined syntax which is parsable without exception by means of digital computation, and such that it is computable whether a given sequence of symbols forms a sentence of the language or not

Note 1 to entry: The term “sentence” is used in this document for a member of a formal language, but in many formal languages other terms are more appropriately used. When a formal language consists of strings of symbols simpliciter, as do formal languages in formal language theory, automata theory, and formal logic, then a well-formed member is called a sentence, except in formal logic, where it is called a well-formed formula. In programming languages, a well-formed member is called a valid program, and in specification languages, a well-formed member is called a valid specification. In diagrammatic languages, a well-formed member would be called a valid diagram.

Note 2 to entry: This notion of formal language is that used in logic, linguistics, mathematics, formal language theory, automata theory and theory of compilation. Formal languages such as (engineering and software) specification languages also often come with a preferred formal semantics (e.g., Z, TLA) and the use of the term in such software engineering contexts often implicitly includes the formal semantics.

Note 3 to entry: In mathematics and automata theory, the term formal language denotes just a set of strings of symbols from a defined symbol set, with no constraints upon how these strings are formed. However, all the formal languages used in M&LT satisfy the conditions given in the definition.

Note 4 to entry: In formal logic, the term “language” (without the prefix “formal”) customarily refers to the set of non-logical symbols. The set of logical symbols of a formal language of logic is customarily taken to be determined, although it varies between first-order logic and higher-order logics.

3.1.20
formal logic
propositional logic, predicate logic, higher-order logic, combinatory logic, modal logic, non-classical logic, other mathematical language structure based on a formal language which has a notion of formal inference, consistency and contradiction

3.1.21
formal proof
<in a formal logic, of a sentence from a given set of sentences> written formal inference of the sentence in the formal logic, using as premises the sentences in the given set

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

3.1.22
formal semantics
< of a language> precise mathematical rendering of the (intended) meaning of the sentences of a formal language, such that any two sentences identical in meaning have the same rendering, and any two sentences which differ in meaning have different renderings

Note 1 to entry: A formal semantics has some or all of the structure of a semantics of a formal logic.

Note 2 to entry: In most formal semantics, two sentences with identical meaning can initially receive differing transcriptions, and it can be the case that these transcriptions have to be further manipulated, say by using a formal "reduction agent", such as a theorem prover or checker, to render the judgement that the renderings, namely the transcriptions reduced by the reduction agent, are identical. The rendering in this case consists in transcription, followed by reduction when it is necessary to determine if two sentences have identical meaning or not.

Note 3 to entry: There is no requirement that it be computable whether two sentences have the same meaning. Identity of meaning can be semi-computable, but it must be at least semi-computable.

3.1.23
formal semantics
<of a sentence> rendering of the sentence in the formal (language) semantics

3.1.24
formal specification
functional specification written in a formal language with a formal semantics

3.1.25
formal verification
<in M&LT> mathematically rigorous argument or method to guarantee that required properties are satisfied

Note 1 to entry: Some formal verification methods work to a specified level of confidence less than certainty.

Note 2 to entry: Examples of formal verification methods are theorem proving, model checking, and abstract interpretation.

3.1.26
fulfil
<by a software object, of a (object) specification; or a (subject) specification, of a (object) specification> be such that the rendering of the (object) specification can be formally proven from the rendering of the software object/(subject) specification in a formal logic

Note 1 to entry: The terms "(subject) specification" and "(object) specification" are used here to indicate which of two specifications is the subject, respectively the object, of the verb "fulfil".

3.1.27
intermediate executable specification
IES
source code at a programming-language level chosen to be above that of the executable source code level, when multiple levels of source code exist, according to 6.1

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

3.1.28
level
<of a programming language> sublanguage of a high-level programming language which is related by syntactic transformation to other levels – other sublanguages – of the programming language

EXAMPLE 1 The programming language Java has so-called “source code” and bytecode. Java bytecode is generated by a Java compiler from Java source code. Java source code is a level; Java bytecode is a level. The Java source code level is “above” the Java bytecode level.

EXAMPLE 2 Commonly-used programming languages, for example C and C++, have “standard libraries”, which consist of more or less complex functions which are invoked in source code using a single identifier. C source code which uses the identifiers for library functions is a level, say C.1. Assume further that C.1 is a subset of C which only uses terms and operators with explicit, unambiguous semantics. C.1 code in which the library functions have been replaced by in-line code which does not involve the library-function identifiers, or only involves some but not all of them, also forms a level, say C.2. C1 is “above” C.2 because source code in C.1 is transformed into source code in C.2 with the same meaning through replacing the library functions not in C.2 with in-line code executing those functions.

Note 1 to entry: A programming language level L.1 is said to be “above” another level L.2 if both L.1 and L.2 are rigorously syntactically defined and a program in L.1 is transformable using common programming-language technology into a program in L.2 with the same semantics.

3.1.29
model checker
software which performs model checking on program code

3.1.30
model checking
<of a collection of program states, machine states, or states of a formal model> enumeration of a collection of states and checking for each state in the collection whether a given state property is fulfilled or not

Note 1 to entry: Model checking is often performed using abstract states for reasons of practicality due to the combinatorics involved. The power of model checking lies largely in the method of abstraction, which attempts to cover a lot of actual program states with as few abstract states as possible in order to check the property effectively.

3.1.31
object code
executable code installed directly on, and executable directly on, digital-computational hardware

3.1.32
proof checker
automated proof checker
software which takes machine-readable formal proofs in a formal logical language and returns a value indicating whether the proof is a correct proof or not

3.1.33
refinement
act of transforming an intermediate executable specification into another intermediate executable specification or executable source code level preserving the formal semantics of the original specification but with more detail about execution

EXAMPLE 1 The transformation of Java source code into Java bytecode is refinement.

EXAMPLE 2 The transformation of C.1 into C.2 in EXAMPLE 2 of 3.1.28 is refinement.

EXAMPLE 3 The transformation of a state-machine description into, say, C code implementing that state machine is refinement.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

3.1.34
relative completeness
<of a formal specification> completeness in which the semantics of sentences are partially given by outside constraints

Note 1 to entry: Formal semantics are compositional when they allow deduction of (semantics) transcriptions purely from sentences (see 3.1.17) without using context, purely using the parse tree of the sentence. Some useful formal semantics are non-compositional; context must be used to determine the meaning of a statement. Using context in this way is an example of a semantics being partially given by outside constraints.

3.1.35
rendering
< of a sentence in a formal semantics> form by means of which a sentence is determined to be identical in meaning to or different in meaning from another sentence

3.1.36
rigorous
<of a specification> unambiguous and explicitly understood to cover all possible behaviours and capable of being thoroughly checked for aspects of correctness and incorrectness

3.1.37
rigorous
<of a formal verification> conducted using formal inference and capable of being thoroughly checked for aspects of correctness and incorrectness

3.1.38
rigorous
<of a process> carried out with attention paid to each and every process step to ensure that it has been correctly executed

3.1.39
runtime verification
technique whereby monitoring variables are placed in program code, whose values represent a partial program state which indicates in greater or in less detail whether a computation is correctly running, and raises an alarm or an exception when this is not the case

3.1.40
safety function
function to be implemented by an E/E/PE safety-related system or other risk reduction measures, that is intended to achieve or maintain a safe state for the EUC, in respect of a specific hazardous event

EXAMPLE Examples of safety functions include:

– functions that are required to be carried out as positive actions to avoid hazardous situations (for example switching off a motor); and

– functions that prevent actions being taken (for example preventing a motor starting).

[SOURCE: IEC 61508-4:2010, 3.5.1]

3.1.41
schedulability analysis
analysis determining if all tasks can be scheduled by a given scheduling algorithm to run and finish before their deadlines

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

3.1.42
semi-computable
semi-decidable
recursively enumerable in the sense in which this term is used in Turing computability and recursive function theory

Note 1 to entry: See [1], [2].

3.1.43
sentence
<in formal language theory> string of symbols which is a well-formed member of the language

Note 1 to entry: When a language is taken to be simply a set of strings of symbols, then a sentence of the language is simply a member of this set

Note 2 to entry: When the formal language is a formal specification language, a sentence in this sense is often called an assertion, or a condition, or a module (of those specification languages which require modules). When the formal language is the language of a logic, a sentence in this sense is called a well-formed formula. When the formal language is a programming language, a sentence in this sense is a (valid) program, which is not conventional in programming-language theory.

Note 3 to entry: In computer science, there are diagrammatic languages, such as those for some forms of automata, or for defining hierarchies of classes of data, or for illustrating interprocess communication, that can also be considered a form of formal language, even though they do not consist of strings of symbols, but rather of certain spatial arrangements of symbols, often two-dimensional. It seems inappropriate to call these sentences. The definition of “well formed” equally applies to such arrangement-languages.

3.1.44
sound
exhibits the property of soundness

3.1.45
soundness
<of a logic, with respect to a formal semantics> quality of the logic and formal semantics that holds if, whenever the logic proves a sentence of the language, this sentence is logically valid in the formal semantics

3.1.46
soundness
<of a static analyzer, with respect to a given formal semantics> quality of a static analyzer and a formal semantics that holds if, whenever the analyzer makes an assertion S, S is logically valid with respect to the formal semantics

Note 1 to entry: If assumptions A1, A2, …, An are used by the analyzer to assert S, then S is to be valid under these same assumptions; i.e., the sentence (A1 ∧ A2 ∧…∧ An → S) is logically valid with respect to the formal semantics.

Note 2 to entry: This definition applies to any static formal verification tool, including theorem provers and model checkers.

Note 3 to entry: It can often be the case that the formal semantics being used is undecidable, hence that there is no algorithm for exceptionlessly checking that S is logically valid with respect to the semantics. When practical checking fails, this is most often for reasons of complexity that do not relate to the undecidability of the underlying semantics; undecidability is mostly not a practical hindrance to such checks.

3.1.47
source code
program written in a formal higher-level programming language

3.1.48
static analysis
analysis of the properties of a program or a specification in a formal language through analysis of the text of the program or the specification rather than through execution or partial execution

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

3.1.49
systematic capability
measure (expressed on a scale of SC 1 to SC 4) of the confidence that the systematic safety integrity of an element meets the requirements of the specified SIL, in respect of the specified element safety function, when the element is applied in accordance with the instructions specified in the system manual for compliant items

[SOURCE: IEC 61508-4:2010, 3.5.9]

3.1.50
theorem prover
software whose primary input consists of putative theorems and a set of premises in a formal logic and which attempts to determine if there is a formal proof of the putative theorems in the logic or not

Note 1 to entry: A theorem prover is for a specific (and specified) formal logic.

Note 2 to entry: A theorem prover can succeed on given input (it identifies a proof), or fail (it has not identified a proof). Failure does not necessarily mean that there is not a proof in the logic. A theorem prover for which failure entails that there is no proof in the logic is known as a "decision procedure".

Note 3 to entry: Some logics preclude that any computational engine can always deliver a correct yes/no answer in every case, so theorem provers are often based on algorithms which are necessarily incomplete.

Note 4 to entry: There are many forms of theorem provers, ranging from "proof checkers", which are not interactive and concomitantly require possibly considerable logical work in advance from the human user, to "proof assistants" or "interactive theorem provers" which exhibit various levels of interaction with a user, who "guides" a proof towards the desired goal.

3.1.51
transcription
<of a sentence of a formal language in a formal semantics> written form into which the sentence is translated in a formal semantics, before any reduction is applied to determine the rendering

3.1.52
unambiguous
<of a sentence, of a specification> having a unique rendering in a formal semantics up to logical or behavioural equivalence

Note 1 to entry: This definition does not formally fulfil the substitutability criterion for terms: “X is unambiguous” would be rendered as “X is has a unique.....”. However, readers can resolve the phrase “is has” easily.

3.1.53
undecidability
category of decision problem complexity implying that there is no automatic algorithm that always proves the decision true if it holds (completeness) and never proves the decision true if it does not hold (soundness)

3.1.54
undecidable
not decidable

Note 1 to entry: If a form of formal reasoning is undecidable, it follows that there is a formal logic which accomplishes this reasoning (or is generally understood to do so) and that there is no algorithm for determining whether a given well-formed formula of the logic is provable or not provable. However, while it can be important to know that the problem being addressed is undecidable, the practical issues involved in using theorem provers on concrete problem instances usually vastly outweigh any issues that can arise through the undecidability of the underlying logic. For many undecidable problems sound algorithms provide practical solutions.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

3.1.55
well-formed
<in a formal language> valid member of the formal language built up according to the rules of definition of the language

Note 1 to entry: If the formal language consists of simple strings of symbols, as in a regular language or context-free language, such as the language of a formal logic, then a well-formed member of the language is often called a sentence. If the formal language is a programming language, a well-formed member is called a valid program. If the formal language is a formal specification language, a well-formed member is called a specification.

Note 2 to entry: In computer science, there are diagrammatic languages, such as those for some forms of automata, or for defining hierarchies of classes of data, or for illustrating interprocess communication, that can also be considered a form of formal language, even though they do not consist of strings of symbols. Such arrangements constructed according to the formation rules of such a diagrammatic language are well-formed according to this definition.

3.1.56
worst-case execution time analysis
WCET analysis
analysis resulting in a trustworthy upper bound on the length of time it takes the machine instructions of a specific task to run on specific hardware

Note 1 to entry: WCET analysis assumes non-interrupted execution of a task. Effects of task preemption and blocking are considered during schedulability analysis.

Note 2 to entry: Interference effects that can affect the non-interrupted execution time of a task, e.g., due to conflicting accesses to resources shared between different cores of multi-core systems, are counted to the WCET. The non-interrupted worst-case execution time of a task discounting such interferences is typically called intrinsic WCET.

3.1.57
worst-case response time analysis
WCRT analysis
analysis resulting in a trustworthy upper bound on the maximal time it takes a task invocation to complete all its activity relative to its arrival time

Note 1 to entry: The arrival time is the time at which a task invocation is ready to start running.

Note 2 to entry: Aspects of the worst-case response time of a task include its worst-case execution time, the time it is preempted by higher-priority tasks and the time it is blocked on synchronization primitives

3.2 Abbreviations

ESCL	Executable Source Code Level
FSRS	Formal Safety Requirements Specification
IES	Intermediate Executable Specification
M&LT	Mathematical and logical techniques in systems engineering
OC	Object Code
SWA/DS	Software Architecture/Design Specification
WCET	Worst-Case Execution Time

4 Conformance to this document

To conform to this technical specification, it shall be demonstrated that all the relevant requirements have been satisfied to any required criteria specified and therefore, for each clause, all the objectives have been met. The demonstration shall include justification of the selection of requirements as relevant, based on the claimed application of M&LT techniques across the software lifecycle.

NOTE Conformance to this technical specification does not require satisfaction of every clause, only those relevant to the lifecycle aspects to which mathematical and logical techniques are applied.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

5 Formal safety requirements specification

5.1 An unambiguous, rigorous formal safety requirements specification (FSRS) shall be written for software components.

NOTE 1 The FSRS required in 5.1 is suitable to fulfil the requirement of IEC 61508-3:2010, 7.2, that is, the FSRS is a software safety requirements specification as required by IEC 61508-3:2010, 7.2.

NOTE 2 According to IEC 61508-3:2010 7.4.2.8, when the software implements some safety function, then the entire software is safety-related and the associated software requirements specification includes a safety requirements specification.

NOTE 3 There are three main ways in which a safety requirements specification could ultimately be inadequate. First, it could fail to govern some safety-related behaviour which it should indeed subsume (that is, fail to exclude some evidently dangerous system behaviour). Second, it could be ambiguous, and through the ambiguity allow some dangerous behaviour which could be excluded in an unambiguous specification. Third, it could use a formal specification language which is inadequate to capture some distinctions between non-dangerous and dangerous behaviours, and thereby exclude some non-dangerous behaviours which are in fact benign, unnecessarily restricting the system developed. The adequacy of a software safety requirements specification is governed by IEC 61508-1:2010, 7.10 and IEC 61508-3:2010, 7.2 and is not completely determined by Clause 4 of this document.

NOTE 4 A language for the FSRS is not specified here; it is for the developer to choose and adopt. However, requiring the FSRS to be unambiguous constrains the possible ways in which an FSRS can be written, as does 5.2. Formal specification languages and their formal semantics are suitable for assuring these properties and accomplishing these tasks, as are controlled natural languages (which are a form of formal specification language.

NOTE 5 Formal safety requirements specification means that a specification language suitable for formal analysis using automated or semi-automated methods is used. Automated analysis itself is not required by 5.1; manual analysis can be used.

5.2 The FSRS shall be checked using mathematical and/or logical techniques for

a) consistency,

b) relative completeness.

NOTE 1 The check for relative completeness is to assure that all scenarios which can lead to hazards have been accounted for in the FSRS.

NOTE 2 Checking for consistency and relative completeness can be performed whether the formal reasoning required is in a decidable logic, or whether it is undecidable. For example, predicate logic is undecidable; Boolean (propositional) logic is decidable, but it is often necessary to try to check statements in predicate logic for consistency, and this is often accomplished in a theorem prover through a technique known as Skolemising, which yields a formula which can be handled by propositional-logic provers, often called SAT solvers.

5.3 The methods and results used for checking consistency and relative completeness shall be documented.

6 Formal software architecture / Design specification

6.1 The software architecture / design specification (SWA/DS) shall be rigorous.

6.2 That the SWA/DS fulfils the FSRS is an assurance point. There shall be a formal, rigorous and correct verification that the SWA/DS fulfils the FSRS.

NOTE 1 The SWA/DS is both a software architecture design and a software system design specification as required by IEC 61508-3:2010 7.4.

NOTE 2 Automated formal verification is not required by 6.2. Manual formal verification can be used.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

7 Higher-level programming languages: Selection of ESCL

7.1 If a higher-level programming language is used, there is one level or there are many levels at which the exact behaviour of the software is completely described. One of these levels shall be chosen to be called Executable Source-Code Level (ESCL).

NOTE 1 For example, if Java is chosen, then either of Java source-code or byte-code can be chosen as ESCL. If a state-machine specification environment is used, and source-code in the programming language C is automatically generated from state-machine specifications, then either the state-machine specification or the resulting C source-code can be chosen as ESCL.

NOTE 2 For example, an imperative programming language with precise semantics is a language in which the exact behaviour of the software is completely described, in the sense of this 7.1.

NOTE 3 When there are many possible choices of ESCL, then the formal verification process can be split up into a chain of individual formal verifications between the levels. The levels before the last (lowest) are called Intermediate Executable Specifications (IES); the last (lowest) is the ESCL.

7.2 That the code in the ESCL fulfils the SWA/DS is an assurance point. There shall be a formal, rigorous and correct formal verification that the code in the ESCL fulfils the SWA/DS.

NOTE Automated formal verification is not required by 7.2. Manual formal verification can be used.

8 Compilation to object code

That the object code fulfils the ESCL is an assurance point. There shall be a rigorous and correct formal verification that the object code fulfils the ESCL.

NOTE 1 Manual formal verification that OC fulfils ESCL is usually impracticable.

NOTE 2 The required rigorous and correct formal verification can fall short of being a fully formal mathematical proof. Typically, there will be many steps in the verification and some of these steps will have formal status. One such step with formal status is compilation of the ESCL using a certified compiler: the certification of the compiler typically renders formal status on a claim that the compiled code fulfils the ESCL. However, compiled code is not necessarily the final object code. There can be calls to library functions which are outside the compiler, and the code generated by the compiler will in any case be linked, along with library functionality. Arguments are typically provided in the documentation to this assurance point that calls to library functions fulfil (part of) the ESCL, as well as that the linker preserves the semantics of the ESCL.

9 Run-time errors and exceptions

9.1 A list of types of run-time errors which are avoided shall be formulated.

9.2 There shall be a rigorous, correct formal verification that run-time errors of the types in the list in 9.1 do not occur.

9.3 There shall be a rigorous, correct formal verification that run-time exception handlers achieve the specified system states when exceptions are raised.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

10 Applicable techniques

10.1 M&LT techniques chosen to substantiate claims for software and its documentation required at assurance points shall be correctly defined, justified, sound and appropriate for the required task.

NOTE 1 Applicable methods and techniques to achieve certain properties of software when applying them during the development are given in Annex A (see Table A.1).

NOTE 2 Annex B (see Table B.1) shows specific methods, which can be used to implement each of the more general methods of Table A.1.

NOTE 3 Annex C (see Table C.1) shows which properties of the software can be assured when using each specific methods of Table B.1.

10.2 The attributes required of the chosen M&LT techniques in 10.1 shall be documented.

10.3 M&LT techniques as given in Annex A (see Table A.1) shall be used regardless of the required SC of the software.

NOTE Use of appropriate techniques during each step of development will result in complete traceability of functional and safety requirements specifications down to source-code level.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

Annex A
(normative)

Applicable Mathematical and Logical Techniques

Table A.1 – M&LT Techniques

	Method / Technique
1	Formal safety requirements specification (FSRS)
2	Formal FSRS analysis (relative completeness, consistency, appropriateness)
3	Automated proving/proof checking of the properties of FSRS[a]
4	Formal modelling, model checking, and model exploration of FSRS
5	Formal software architecture / design specification (SWA/DS)
6	Formal analysis of SWA/DS
7	Automated proving/proof checking of fulfilment of the FSRS by SWA/DS[a]
8	Formal modelling, model checking, and model exploration of SWA/DS
9	Abstract interpretation (sound static analysis)
10	Co-development of SWA/DS with ESCL[b]
11	Automated source-code generation from SWA/DS or intermediate executable specification (IES)[a]
12	Automated proving/proof checking of fulfilment of SWA/DS by IES[a]
13	Automated verification-condition generation from/with ESCL[a]
14	Rigorous formal semantics of ESCL
15	Automated ESCL-level proving /proof checking of properties (such as freedom from susceptibility to certain kinds of run-time error)[a]
16	Automated proving/proof checking of fulfilment of SWA/DS by ESCL[a]
17	Formal test case generation from FSRS[c]
18	Formal test case generation from SWA/DS[c]
19	Formal test case generation from IES[c]
20	Formal test case generation from ESCL[c]
21	Formal coding-standards analysis
22	Worst-Case Execution Time (WCET) analysis/Worst-Case Response Time (WCRT) analysis/Schedulability analysis
23	Monitor synthesis/runtime verification
24	Formally verified compilation
25	Automated proving/proof checking of fulfilment of ESCL by object code

[a] "Automated" means that a tool is used, in part or in whole. Qualification and use of tools is covered in IEC 61508-3:2010 7.4.4.

[b] "Co-development" refers to software development with the use of such annotated programming languages as ANNA (Stanford ANNotated Ada), SPARK[2] and Eiffel™[3].

[c] Testing is not a method which can provide guarantees at an assurance point – it can, in a well-known quotation from 1969, "demonstrate the presence of bugs but not their absence." However, M&LT is used in test case generation and so is listed here. Test case generation involves requirements on the output of a test to be formulated, and checked when the test is run.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

---

2 SPARK is the trade name of a product supplied by AdaCore. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

3 This trademark is provided for reasons of public interest or public safety. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC.

NOTE Table A.1 is intended to classify industrially-mature M&LT. It is the intent of the table that every industrially-mature M&LT technique falls under one of the categories in the second column of the table. However, methods develop and evolve: for example, WCET analysis was included in initial versions of this table in 2010 and is complemented by WCRT analysis and schedulability analysis. It is to be expected that techniques not yet included evolve and become industrially-mature; the table is to be read as a best attempt at classification at time of publication.

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

Annex B
(informative)

Specific Mathematical and Logical Techniques

Table B.1 – Specific M&LT Techniques/Tools

	Name of the Technique or Method	Specific M&LT Techniques and Tools	Existing in / new
1	Formal safety requirements specification (FSRS)	• Formal Description Language • Formal Notation • Controlled Natural Language • Formal Logic • Ontological Hazard Analysis	In IEC 61508-3:2010 Table A.1
2	Formal FSRS analysis	• Automated Consistency Checking • Theorem Prover / Proof Assistant / Proof Checker	New
3	Automated proving/proof checking of properties (consistency, completeness of certain types) of FSRS	• Automated Consistency Checking • Theorem Prover / Proof Assistant / Proof Checker	New
4	Formal modelling, model checking, and model exploration of FSRS	• Modelling Language • Formal modelling of functions • Hierarchical Modelling • Model Checker	In IEC 61508-3:2010 Table A.2
5	Formal design specification (SWA/DS)	• Graphical Design Language • Formal Description Language • Formal Logic • Formal Refinement	In IEC 61508-3:2010 Table A.2
6	Formal analysis of SWA/DS	• Automated Consistency Checking • Theorem Prover / Proof Assistant / Proof Checker	In IEC 61508-7:2010, B.2.4
7	Automated proving/proof checking of fulfilment of the FSRS by SWA/DS	• Formal Refinement • Theorem Prover • Proof Assistance • Proof Checker • Model Checking	New
8	Formal modelling, model checking, and model exploration of SWA/DS	• Modelling of Functions • Hierarchical State Machines • Modelling using petri-nets • Model Checking • Model State Exploration	In IEC 61508-3:2010 Tables A.2, A.4, A.7, B.5, B.7

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

	Name of the Technique or Method	Specific M&LT Techniques and Tools	Existing in / new
9	Abstract interpretation (sound static analysis)	• Data flow analysis • Information flow analysis • Code guideline checking • Runtime error analysis • Worst-case execution time analysis • Worst-case stack usage analysis	New
10	Co-development of SWA/DS with Executable Source-Code Level (ESCL)	• Co-development Tools for software source-code • Languages allowing simultaneous development of source code and formal design specifications	New
11	Automated source-code generation from SWA/DS or intermediate executable specification (IES)	• Formal semantics-preserving Code Generators from textual formal design specifications • Formal semantics-preserving Code Generators from graphical design specifications	In IEC 61508-7:2010, C.4.6
12	Automated or assisted proving/proof checking of fulfilment of SWA/DS by IES	• Proof Checker • Theorem Prover	New
13	Automated or assisted formal-verification-condition generation from/with ESCL	• Proof Checker • Theorem Prover	New
14	Rigorous formal semantics of ESCL	• Programming language with rigorous formal semantics • Programming language safe subset with rigorous formal semantics	New
15	Automated ESCL-level proving /proof checking of properties (such as freedom from susceptibility to certain kinds of run-time error)	• Static Code analysis tools	New
16	Automated proving/proof checking of fulfilment of SWA/DS by ESCL	• Automated Proof Checker	New
17	Formal test case generation from FSRS	• Automatic test case generators	New
18	Formal test case generation from SWA/DS	• Automatic test case generators	New
19	Formal test case generation from IES	• Automatic test case generators	New
20	Formal test case generation from ESCL	• Automatic test case generators	New
21	Formal coding-standards analysis (SPARK, MISRA C, etc)	• Coding Standards • Coding Standards Analysis	In IEC 61508-3:2010, Tables A.3, A.4, B.1
22	Worst-Case Execution Time (WCET) analysis/Worst-Case Response Time (WCRT) analysis/Schedulability analysis	• WCET-analysis tools • WCRT-analysis tools • Schedulability-analysis tools	New

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024

	Name of the Technique or Method	Specific M&LT Techniques and Tools	Existing in / new
23	Monitor synthesis/runtime verification	• Monitor Synthesis / Runtime Verification	New
24	Formally verified compilation	• Formally verified compiler	New
25	Automated proving/proof checking of fulfilment of ESCL by object code	• Automated proof checker • Automated theorem prover	New

IECNORM.COM : Click to view the full PDF of IEC TS 61508-3-2:2024