



BSI Standards Publication

Functional safety of electrical/electronic/ programmable electronic safety-related systems

Part 3-3: Object-oriented software in safety-related systems

National foreword

This Published Document is the UK implementation of IEC TR 61508-3-3:2025.

The UK participation in its preparation was entrusted to Technical Committee GEL/65, Measurement and control.

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.

This publication is not to be regarded as a British Standard.

© The British Standards Institution 2025
Published by BSI Standards Limited 2025

ISBN 978 0 580 99577 4

ICS 13.110; 25.040.40

Compliance with a Published Document cannot confer immunity from legal obligations.

This Published Document was published under the authority of the Standards Policy and Strategy Committee on 31 July 2025.

Amendments/corrigenda issued since publication

Date	Text affected

TECHNICAL REPORT

**Functional safety of electrical/electronic/programmable electronic safety-related systems –
Part 3-3: Object-oriented software in safety-related systems**



THIS PUBLICATION IS COPYRIGHT PROTECTED
Copyright © 2025 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 Varembe
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.

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



Documents

Document Sharing Hub

CONTENTS

FOREWORD 2

INTRODUCTION..... 4

1 Scope 5

2 Normative references 5

3 Terms, definitions and abbreviated terms 5

4 Structure of this document..... 9

5 Initial consideration at the project start 10

6 Contract-oriented programming 12

7 Encapsulation..... 15

8 Inheritance 21

9 Polymorphism..... 27

10 Dynamic objects 31

11 Cross issue considerations during the project..... 39

12 Maintenance..... 43

13 Exception handling 46

Bibliography..... 52

Figure 1 – Class hierarchy – inheritance tree 27

Table 1 – Tailoring in proposal tables 10

Table 2 – Initial consideration at the project start..... 12

Table 3 – Contract-oriented programming 14

Table 4 – Encapsulation 18

Table 5 – Inheritance 23

Table 6 – Polymorphism 29

Table 7 – Dynamic objects 33

Table 8 – Cross issue considerations during the project..... 40

Table 9 – Maintenance 44

Table 10 – Exception handling 49

INTERNATIONAL ELECTROTECHNICAL COMMISSION

**Functional safety of electrical/electronic/programmable
electronic safety-related systems -
Part 3-3: Object-oriented software in safety-related systems**

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

This TR is a supplement to the IEC 61508 standard series. It has to be read in conjunction with IEC 61508-3 and proposes a way how the use of object-oriented software in safety relevant applications can be justified.

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

Draft	Report on voting
65A/1176/DTR	65A/1181/RVDTR

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 Report 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.

INTRODUCTION

This document addresses specific concepts associated with object-oriented (OO) software. It deals only with OO software in general without referencing any specific language. Each of the concepts is discussed under separate clauses, one addressing fundamentals – i.e. benefits, disadvantages and counter-measures to the disadvantages, the others detailing guidance on the attributes to be satisfied in safety-related systems, according to the systematic capability to be achieved.

It is useful to consider addressing the language-specific best practice contained in guidelines, coding rules, handbooks etc. for each OO language. If an object-oriented module is modified, it is proposed that the entire module conform to the guidance within this document. Further, it is useful to consider assessing the interfaces, interactions and side effects on unchanged modules to determine that there is no impact on other unchanged modules and their integration. See also IEC 61508-3:2010, Annex D.

This document is intended as a supplement to the existing requirements in the IEC 61508 series which continue to apply.

1 Scope

This part of IEC 61508, which is a Technical Report, makes a proposal as to which topics to consider and which methods and techniques to use when designing object-oriented software to ensure suitable quality for use in functional safety applications.

Object-oriented languages are perceived as "state-of-the-art" nowadays. Such languages seem to be excluded from use by several statements in IEC 61508-3. However there are additions in some tables such as in IEC 61508-3:2010, Table B.1, where notes are added under which their use might be justified. Such exceptions that would allow, for example, dynamic objects, name the main concerns such as memory allocation and predictable timing issues and guide the user to safe use of object-oriented languages. These considerations are taken up in this document to specify methods and techniques that allow the reduction of systematic faults to the levels required by the respective systematic capabilities.

This document is not intended to replace any part of IEC 61508-3. Rules that exist in IEC 61508-3 are valid here as well and are not repeated, including rules that concern:

- the software life cycle,
- involvement of the assessor,
- modularization,
- principle of information hiding,
- proving and conventional testing,
- basic aspects of documentation,
- low coupling and high cohesion,
- responsibilities and training of people,
- operational experience as described in IEC 61508-4 and IEC 61508-7.

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 61508-1:2010 *Functional Safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements*

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

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 61508-4 and the following apply.

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

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

3.1.1**abstract class**

class that cannot be directly instantiated

3.1.2**abstract data type**

data type for which the user can create instances and operate on those instances, but the range of valid operations available to the user does not depend in any way on the internal representation of the instances or the way in which the operations are realized. The data is "abstract" in the sense that values in the extent, i.e., the concrete values that represent the instances, are any set of values that support the operations and are irrelevant to the user. An abstract data type defines the operations on the data as part of the definition of the data and separates what can be done (interface) from how it is done (realization)

[SOURCE: ISO/IEC/IEEE 31320-2:2012, 3.1.2]

3.1.3**abstraction layer**

interface to the system that allows some or all the capabilities of the system to be accessed in a different and generally more abstract manner

Note 1 to entry: An abstraction layer for a module is the same in the case where the system is the module.

[SOURCE: ISO 22166-1:2021, 3.1.1]

3.1.4**assertion**

sentence or proposition in logic which is asserted (or assumed) to be true

[SOURCE: ISO/TS 21526:2019, 3.7]

3.1.5**invariant**

<object-oriented software> assertion that is always true for a specified segment or at a specified point of a computer program

3.1.6**pre-condition**

condition that is required to be true before making an execution of a property request

[SOURCE: ISO/IEC/IEEE 31320-2:2012, 3.1.148]

3.1.7**post-condition**

condition that is guaranteed to be true after a successful execution of a property request

[SOURCE: ISO/IEC/IEEE 31320-2:2012, 3.1.147]

3.1.8**base class**

class which defines attributes and operations that are inherited by a subclass via a generalization relationship

[SOURCE: ISO 13374-2:2007, B.2.2.7]

3.1.9

cast

treat an object of one type as an object of another type

[SOURCE: ISO/IEC/IEEE 31320-2:2012, 3.1.17]

3.1.10

down-cast

cast of an object from a base class sub-object, to a more derived class sub-object

Note 1 to entry: Depending on the complexity of the object's type, this may make RTTI (runtime type information) and the use of the dynamic cast operator necessary.

[SOURCE: ISO/IEC TR 18015:2006, 3.15]

3.1.11

delegation

action that assigns something, such as authorization, responsibility or provision of a service to another object

Note 1 to entry: A delegation, once made, may later be withdrawn.

[SOURCE: ISO/IEC 15414:2015, 6.6.6]

3.1.12

derived class

class created by inheritance from another class

Note 1 to entry: A derived class is also named extended class or child class.

[SOURCE: IEC 61131-3:2013, 3.27]

3.1.13

flattening

virtual disassembling an inheritance hierarchy into classes of the lowest level; mainly for deriving the necessary or desired test cases

3.1.14

framework

reusable design (models and/or code) that can be refined (specialized) and extended to provide some portion of the overall functionality of many applications

[SOURCE: ISO/IEC/IEEE 31320-2:2012, 3.1.64]

3.1.15

friend class

other class that has the same access rights to methods and attributes as the class itself

3.1.16

garbage collection

<computer resource management> storage allocation technique in which the contents of all allocated storage areas are moved to the beginning of the storage space and the remaining storage blocks are combined into a single block

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.2412, modified – the term “memory compaction” has been removed, the domain has been added, and definition 2 has been removed.]

3.1.17

information hiding

principle of denying access to or knowledge of a language construct or specific details thereof, except for those details considered essential for the user to know

3.1.18

inheritance hierarchy

hierarchical arrangement of managed object classes where the hierarchy is organized on the basis of the class specialization

[SOURCE: ISO/IEC 10165-1:1993, 3.8.17]

3.1.19

interface (of an object)

names of all the methods defined for the object, including inherited methods; for each of the methods the ordered list of its formal parameters and the description and passing technique associated with each, any returned value and its description, and exceptions that can be raised

[SOURCE: ISO/IEC 1989:2014, 4.109]

3.1.20

Liskov Substitution Principle

LSP

principle for hierarchies in which none of the used methods is supposed to have a stronger pre-condition than the related methods of the classes it is derived from and no weaker post-condition than the related methods of these classes

3.1.21

memory fragmentation

available memory split into pieces such that it is difficult to use

3.1.22

reflective construct

construct that is viewed as the cause of measures in the relationship between a construct and its measures

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.3361]

3.1.23

exception

- 1) event that causes suspension of normal program execution
- 2) indication that an operation request was not performed successfully

Note 1 to entry: Types include addressing exception, data exception, operation exception, overflow exception, protection exception, and underflow exception.

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.1496]

3.1.24

class

description of a set of objects that share the same attributes, operations, relationships, and semantics; classes may have attributes and may support operations

[SOURCE: ISO/IEC 14776-261:2012, 3.1.26]

3.1.25**root class**

class, from which all other classes in a hierarchy are derived, but which is not derived itself from any other class, top of an inheritance hierarchy, special base class

Note 1 to entry: In a wider context a root class forms a description of a set of objects that share the same attributes, operations, relationships, and semantics; classes may have attributes and may support operations.

3.2 Abbreviated terms

EUC	Equipment under control
OO	Object-oriented or object-orientation
SIL	Safety integrity level

4 Structure of this document**4.1 General**

This document describes the special characteristics of OO software and methods for mitigation of the safety related problems considering these topics:

- 1) initial considerations at the project start;
- 2) contract-oriented programming;
- 3) encapsulation;
- 4) inheritance;
- 5) polymorphism;
- 6) dynamic objects;
- 7) cross issue considerations during the project;
- 8) maintenance;
- 9) exception handling.

Each of the concepts is discussed under two separate aspects:

- 1) fundamentals: benefits, disadvantages and counter-measures to the disadvantages;
- 2) mitigating methods: to satisfy in safety-related systems, according to the systematic capability to be achieved.

This document does not refer to a specific lifecycle. The main focus is on the development phase (Box 10) of the life cycle of IEC 61508-1, and the details that conform to IEC 61508-3.

4.2 Tailoring of methods or techniques

Clause 5 to Clause 13 propose techniques and measures based on topics related to object-oriented software. The appropriate techniques and measures are selected according to the systematic capability. The use of other measures and techniques where necessary can be considered, provided that the same effectiveness is met. Details regarding the techniques and measures are described in Table 1.

The ranking of the techniques and measures is linked to the concept of effectiveness used in IEC 61508-2.

Table 1 – Tailoring in proposal tables

++	The technique or measure is strongly proposed for this systematic capability. If this technique or measure is not used then the rationale behind not using it is detailed during the safety planning and agreed with the assessor.
+	The technique or measure is proposed for this safety integrity level as a lower proposal to a ++ proposal.
-	The technique or measure has no proposal for or against being used.
--	The technique or measure is positively not proposed for this safety integrity level. If this technique or measure is used then the rationale behind using it is detailed during the safety planning and agreed with the assessor.

All other factors being equal, techniques which are ranked ++ will be more effective in either preventing the introduction of systematic faults during software development, or (for the case of the software architecture) more effective in controlling residual faults in the software revealed during execution than techniques ranked as +.

Given the large number of factors that affect systematic capability it is not possible to give an algorithm for combining the techniques and measures that will be correct for any given application. For a particular application, it is proposed to state the selected combination of techniques or measures during safety planning found appropriate.

5 Initial consideration at the project start

5.1 Fundamentals

5.1.1 General

Clause 5 deals with issues relevant to the decision whether OO design and programming could be used. It is considered useful to take the indicated considerations and measures before design or programming starts, or both.

5.1.2 Flexible design capabilities

Benefits:

- use of tailored design options to the problem at hand.

Problems:

- fault options might be missed in verification planning.

5.1.3 Facilitates formal graphical design concepts

Benefits:

- graphical design enables the designer to grasp the main design features at a glance;
- most graphical design techniques are based on formal concepts;
- modern graphical design formalisms cover a broad range of relevant design aspects, from data models to state machines to constraints;
- graphical designs are easier to understand for many people.

Problems:

- complex graphical designs might lead to restricted safety due to inefficient verification techniques;
- complex graphical designs might lead to restricted safety due to inefficient implementation and verification efforts;
- effort for manual test planning might explode;
- verification strategies planned without regard to the graphical design might become inefficient.

5.1.4 Abstraction layers

Benefits:

- concept of interfaces of class hierarchies enables an easier overview over the software system.

Problems:

- additional editing effort.

5.1.5 Facilitates reuse

Benefits:

- improved encapsulation, inheritance and polymorphism each provide additional facilities that could be exploited for reuse;
- by helping to reduce code redundancy, OO designs cater for reuse from the very beginning of a design.

Problems

- reuse might lead to unwanted functionality;
- reused software might contain elements which are not validated according to safety prerequisites;
- reuse might increase the risk of an unstructured heap of code entities.

5.1.6 More functionality in the programming environment

Benefits:

- OO languages provide implicit mechanisms to cope with the functionality added by OO's concepts (through compilation or run-time systems or IDE), like stack handling for recursion.

Problems:

- maximum response time exceeded due to invisible activities.

5.2 Mitigating measures

A typical set of mitigating measures would be as follows:

- 1) requirements of IEC 61508-3 and their SIL classification states are observed;
- 2) requirements of IEC 61508-3 are adjusted as per the proposals in this document;
- 3) proposals of 5.3 are observed from the beginning of the development cycle;
- 4) adequate patterns and tools are used;
- 5) considered libraries are known before programming starts;

NOTE OO software generally relies heavily on libraries.

- 6) libraries are evaluated to meet the proposals of this document;

- 7) ease of modification and adaptation are considered from the beginning on during the whole development project;
- 8) well-accepted language-specific design and coding rules are followed and documented throughout the project. See also IEC 61508-3.

It is beneficial to know the safety-related language manual of the manufacturer as well as the compiler, programming environment, run-time system and operating system to be used and when the developer is competent in using them. See also IEC 61508-3.

5.3 Technical details

5.3.1 Proposals

Table 2 contains proposals for consideration at project start.

Table 2 – Initial consideration at the project start

No.	Proposal	References	SC1	SC2	SC3	SC4
5.3.1.1	Use OO patterns or frameworks, or both.		+	+	+	++
5.3.1.2	Use automated OO-related methods and tools.	IEC 61508-3:2010, 7.4.4; 7.3.2.2	+	+	+	+

5.3.2 OO patterns and/or frameworks

It is better to use existing OO design patterns and frameworks instead of completely new design. The design and implementation effort will be reduced and already existing safety considerations for these patterns and frameworks will help avoid new errors. Moreover, since people are already acquainted with them, understanding the new designs will be much easier. Patterns and frameworks as they are common in OO design are helpful in taking advantage of design experiences from earlier projects. If a framework is available that has already been employed successfully for other projects or similar problems, it is beneficial to use it, or use a part of it, or at least take it as a template. The requirements for pre-existing software from IEC 61508-1, IEC 61508-2 and IEC 61508-7 apply accordingly.

5.3.3 Automated OO-related methods and tools

In OO, methods and tools exist to support specification, design, code generation and verification including test generation on the base of (semi-)formal specifications and designs.

Semi-formal modelling systems such as UML provide a sound basis for automated design verification, code and test generation where, whenever the corresponding UML software is appropriately validated (according to tool levels and SILs), reliable results are reached.

6 Contract-oriented programming

6.1 Fundamentals

6.1.1 General

Contracts are an implementation of the general concept of behavioural subtyping which extends the inheritance principle: Its basis – the Liskov substitution principle – states that replacing any object of a given class by an object of a derived class results in identically correct statements without altering any of its genuine properties.

NOTE The Liskov substitution principle (LSP) is observed especially during creation of an object hierarchy.

Though not necessarily related to object-oriented programming – contract-oriented programming could as well be applied in more basic programming approaches, yet with the aid of a lot of infrastructure programming. It is best integrated into the object-oriented approach where many programming languages already support it, for example Ada, Eiffel, Scala, or are planned to support it in the future for example C++¹.

Contracts provide means for formal description of interfaces, designs and code, and with the aid of run-time checking the constraints allow for a better way of introducing exceptions. If applied properly, however, explicitly raising exceptions becomes unnecessary.

Contract-oriented programming provides means for static formal correctness checks of designs and programs. Even if contract-oriented programming is not used consider applying the proposed measures.

6.1.2 Definition of interfaces

Benefits:

- well-defined interfaces by formal definitions;
- well-defined specification of methods;
- responsibility-driven design;
- support of run-time checks.

Problems:

- necessity of learning another type of thinking;
- if not used properly, faulty situations might not be adequately checked and formal verification becomes more difficult;
- overhead for writing the contracts.

6.1.3 Controlled checkpointing

Benefits:

- better testing capabilities;
- well defined location of checkpoints.

Problems:

- redundancy due to checkpoints;
- maximum allowed reaction time exceeded due to invisible activities.

6.1.4 Verification, validation

Benefits:

- tests could be supported by the contract; in particular test data selection could be supported by preconditions, while test evaluation could be supported by post conditions;
- tests could be generated such as to attempt to breach the contracts;
- improved efficiency and reliability;
- already existing designs could be verified.

¹ Ada, Eiffel, Scala and C++ are examples of suitable products available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of these products.

Problems:

- overhead for ensuring that classes are derived such that preconditions and invariants within the hierarchy are properly set.

6.2 Mitigating measures

A typical set of mitigating measures would be as follows:

- 1) contracts are used during design and coding;
- 2) verification is done against the contracts; contracts are used for test planning in general;
- 3) all contracts are covered during testing.

6.3 Technical details

6.3.1 Proposals

Table 3 contains proposals for consideration for contract-oriented programming.

Table 3 – Contract-oriented programming

No.	Proposal	References	SC1	SC2	SC3	SC4
6.3.1.1	Class invariants are stated and kept.	IEC 61508-3:2010, 7.4.2.2	+	+	+	++
6.3.1.2	Invariants of a base class are preserved in a derived class.	IEC 61508-3:2010, 7.4.2.2	+	+	++	++
6.3.1.3	All assertion checks are executed during testing.	IEC 61508-3:2010, Table A.2 row 3a; Table C.2 row 3a	+	+	++	++
6.3.1.4	Pre- and post-conditions and invariants of methods are explicitly stated.	IEC 61508-3:2010, 7.4.2.2	++	++	++	++
6.3.1.5	Pre-conditions of a base class are not to be strengthened in a derived class.	IEC 61508-3:2010, 7.4.2.2	++	++	++	++
6.3.1.6	Post-conditions of a base class will not be weakened in a derived class.	IEC 61508-3:2010, 7.4.2.2; Table C.2	++	++	++	++
6.3.1.7	All component interactions are tested such as to cover all possible combinations of internal pre- and post-states.	IEC 61508-3:2010, Table A.6	+	+	++	++
6.3.1.8	No derived class introduces new public methods for modifying the state of the base class (history constraint).		++	++	++	++

6.3.2 Keeping class invariants

In OO, invariants are especially important, as they are a major means of asserting integrity. Class invariants help specifying interfaces of classes, documenting and understanding the semantics of classes and class hierarchies.

6.3.3 Preserved invariants of a base class in a derived class

Facilitates understanding the related hierarchy by understanding the base class.