IEC 61508-7

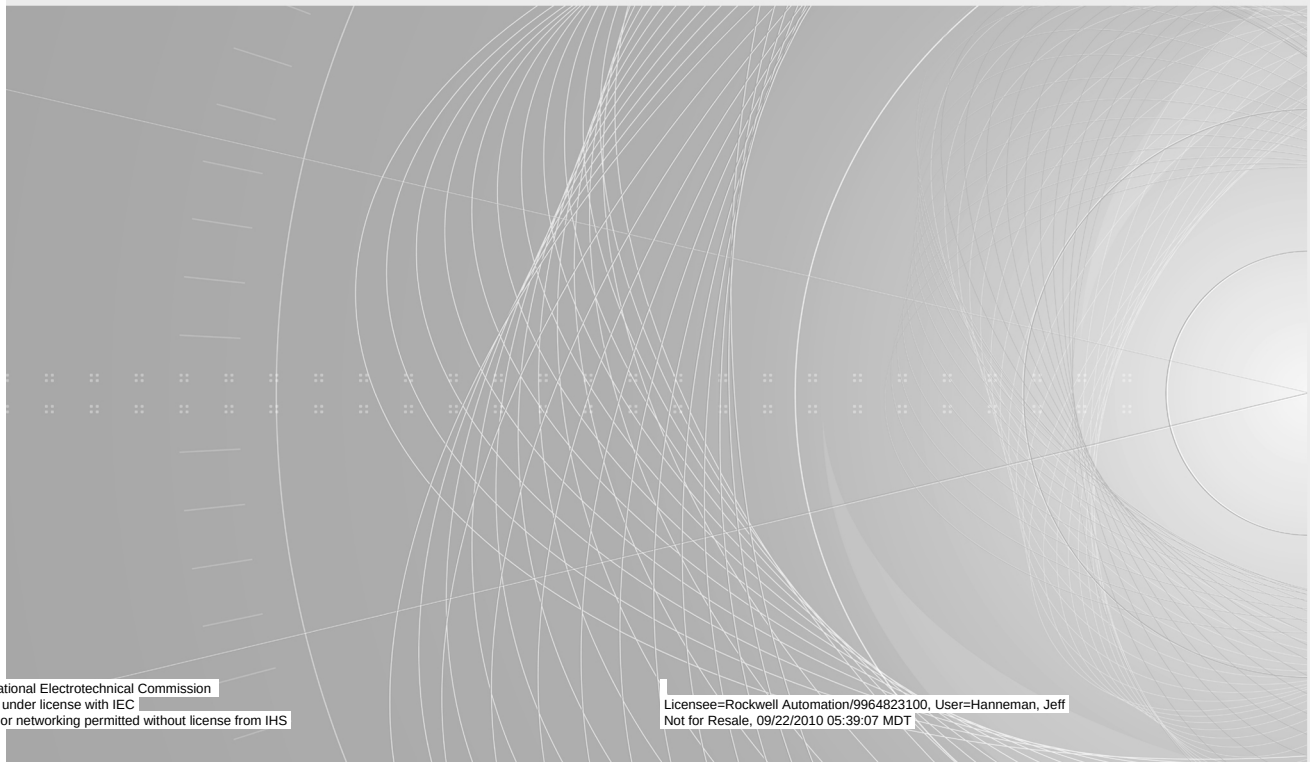
Edition 2.0 2010-04

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Functional safety of electrical/electronic/programmable electronic safety-related systems –
Part 7: Overview of techniques and measures

Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité –
Partie 7: Présentation de techniques et mesures



IEC 61508-7:2010

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: 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 corrigenda or an amendment might have been published.

- Catalogue of IEC publications: www.iec.ch/searchpub
The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.
- IEC Just Published: www.iec.ch/online_news/justpub
Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.
- Electropedia: www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.
- Customer Service Centre: www.iec.ch/webstore/custserv
If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:
Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

- Catalogue des publications de la CEI: www.iec.ch/searchpub/cur_fut-f.htm
Le Catalogue en-ligne de la CEI vous permet d'effectuer des recherches en utilisant différents critères (numéro de référence, texte, comité d'études,…). Il donne aussi des informations sur les projets et les publications retirées ou remplacées.
- Just Published CEI: www.iec.ch/online_news/justpub
Restez informé sur les nouvelles publications de la CEI. Just Published détaille deux fois par mois les nouvelles publications parues. Disponible en-ligne et aussi par email.
- Electropedia: www.electropedia.org
Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 20 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International en ligne.
- Service Clients: www.iec.ch/webstore/custserv/custserv_entry-f.htm
Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions, visitez le FAQ du Service clients ou contactez-nous:
Email: csc@iec.ch
Tél.: +41 22 919 02 11
Fax: +41 22 919 03 00

IEC 61508-7

Edition 2.0 2010-04

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 7: Overview of techniques and measures

Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité – Partie 7: Présentation de techniques et mesures

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE XG
CODE PRIX

ICS 25.040.40; 35.240.50

ISBN 978-2-88910-530-4

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

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

Part 7: Overview of techniques and measures

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

International Standard IEC 61508-7 has been prepared by subcommittee 65A: System aspects, of IEC technical committee 65: Industrial-process measurement, control and automation.

This second edition cancels and replaces the first edition published in 2000. This edition constitutes a technical revision.

This edition has been subject to a thorough review and incorporates many comments received at the various revision stages.

The text of this standard is based on the following documents:

FDIS	Report on voting
65A/554/FDIS	65A/578/RVD

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

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts of 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 publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

INTRODUCTION

Systems comprised of electrical and/or electronic elements have been used for many years to perform safety functions in most application sectors. Computer-based systems (generically referred to as programmable electronic systems) are being used in all application sectors to perform non-safety functions and, increasingly, to perform safety functions. If computer system technology is to be effectively and safely exploited, it is essential that those responsible for making decisions have sufficient guidance on the safety aspects on which to make these decisions.

This International Standard sets out a generic approach for all safety lifecycle activities for systems comprised of electrical and/or electronic and/or programmable electronic (E/E/PE) elements that are used to perform safety functions. This unified approach has been adopted in order that a rational and consistent technical policy be developed for all electrically-based safety-related systems. A major objective is to facilitate the development of product and application sector international standards based on the IEC 61508 series.

NOTE 1 Examples of product and application sector international standards based on the IEC 61508 series are given in the bibliography (see references [21], [22] and [37]).

In most situations, safety is achieved by a number of systems which rely on many technologies (for example mechanical, hydraulic, pneumatic, electrical, electronic, programmable electronic). Any safety strategy must therefore consider not only all the elements within an individual system (for example sensors, controlling devices and actuators) but also all the safety-related systems making up the total combination of safety-related systems. Therefore, while this International Standard is concerned with E/E/PE safety-related systems, it may also provide a framework within which safety-related systems based on other technologies may be considered.

It is recognized that there is a great variety of applications using E/E/PE safety-related systems in a variety of application sectors and covering a wide range of complexity, hazard and risk potentials. In any particular application, the required safety measures will be dependent on many factors specific to the application. This International Standard, by being generic, will enable such measures to be formulated in future product and application sector international standards and in revisions of those that already exist.

This International Standard

– considers all relevant overall, E/E/PE system and software safety lifecycle phases (for example, from initial concept, through design, implementation, operation and maintenance to decommissioning) when E/E/PE systems are used to perform safety functions;
– has been conceived with a rapidly developing technology in mind; the framework is sufficiently robust and comprehensive to cater for future developments;
– enables product and application sector international standards, dealing with E/E/PE safety-related systems, to be developed; the development of product and application sector international standards, within the framework of this standard, should lead to a high level of consistency (for example, of underlying principles, terminology etc.) both within application sectors and across application sectors; this will have both safety and economic benefits;
– provides a method for the development of the safety requirements specification necessary to achieve the required functional safety for E/E/PE safety-related systems;
– adopts a risk-based approach by which the safety integrity requirements can be determined;
– introduces safety integrity levels for specifying the target level of safety integrity for the safety functions to be implemented by the E/E/PE safety-related systems;

NOTE 2 The standard does not specify the safety integrity level requirements for any safety function, nor does it mandate how the safety integrity level is determined. Instead it provides a risk-based conceptual framework and example techniques.

– sets target failure measures for safety functions carried out by E/E/PE safety-related systems, which are linked to the safety integrity levels;

– sets a lower limit on the target failure measures for a safety function carried out by a single E/E/PE safety-related system. For E/E/PE safety-related systems operating in

  – a low demand mode of operation, the lower limit is set at an average probability of a dangerous failure on demand of 10^{-5};

  – a high demand or a continuous mode of operation, the lower limit is set at an average frequency of a dangerous failure of 10^{-9} [h^{-1}];

NOTE 3 A single E/E/PE safety-related system does not necessarily mean a single-channel architecture.

NOTE 4 It may be possible to achieve designs of safety-related systems with lower values for the target safety integrity for non-complex systems, but these limits are considered to represent what can be achieved for relatively complex systems (for example programmable electronic safety-related systems) at the present time.

– sets requirements for the avoidance and control of systematic faults, which are based on experience and judgement from practical experience gained in industry. Even though the probability of occurrence of systematic failures cannot in general be quantified the standard does, however, allow a claim to be made, for a specified safety function, that the target failure measure associated with the safety function can be considered to be achieved if all the requirements in the standard have been met;

– introduces systematic capability which applies to an element with respect to its confidence that the systematic safety integrity meets the requirements of the specified safety integrity level;

– adopts a broad range of principles, techniques and measures to achieve functional safety for E/E/PE safety-related systems, but does not explicitly use the concept of fail safe. However, the concepts of “fail safe” and “inherently safe” principles may be applicable and adoption of such concepts is acceptable providing the requirements of the relevant clauses in the standard are met.

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

Part 7: Overview of techniques and measures

1 Scope

1.1 This part of IEC 61508 contains an overview of various safety techniques and measures relevant to IEC 61508-2 and IEC 61508-3.

The references should be considered as basic references to methods and tools or as examples, and may not represent the state of the art.

1.2 IEC 61508-1, IEC 61598-2, IEC 61508-3 and IEC 61508-4 are basic safety publications, although this status does not apply in the context of low complexity E/E/PE safety-related systems (see 3.4.3 of IEC 61508-4). As basic safety publications, they are intended for use by technical committees in the preparation of standards in accordance with the principles contained in IEC Guide 104 and ISO/IEC Guide 51. IEC 61508-1, IEC 61508-2, IEC 61508-3 and IEC 61508-4 are also intended for use as stand-alone publications. The horizontal safety function of this international standard does not apply to medical equipment in compliance with the IEC 60601 series.

1.3 One of the responsibilities of a technical committee is, wherever applicable, to make use of basic safety publications in the preparation of its publications. In this context, the requirements, test methods or test conditions of this basic safety publication will not apply unless specifically referred to or included in the publications prepared by those technical committees.

1.4 Figure 1 shows the overall framework for parts 1 to 7 of IEC 61508 and indicates the role that IEC 61508-7 plays in the achievement of functional safety for E/E/PE safety-related systems.

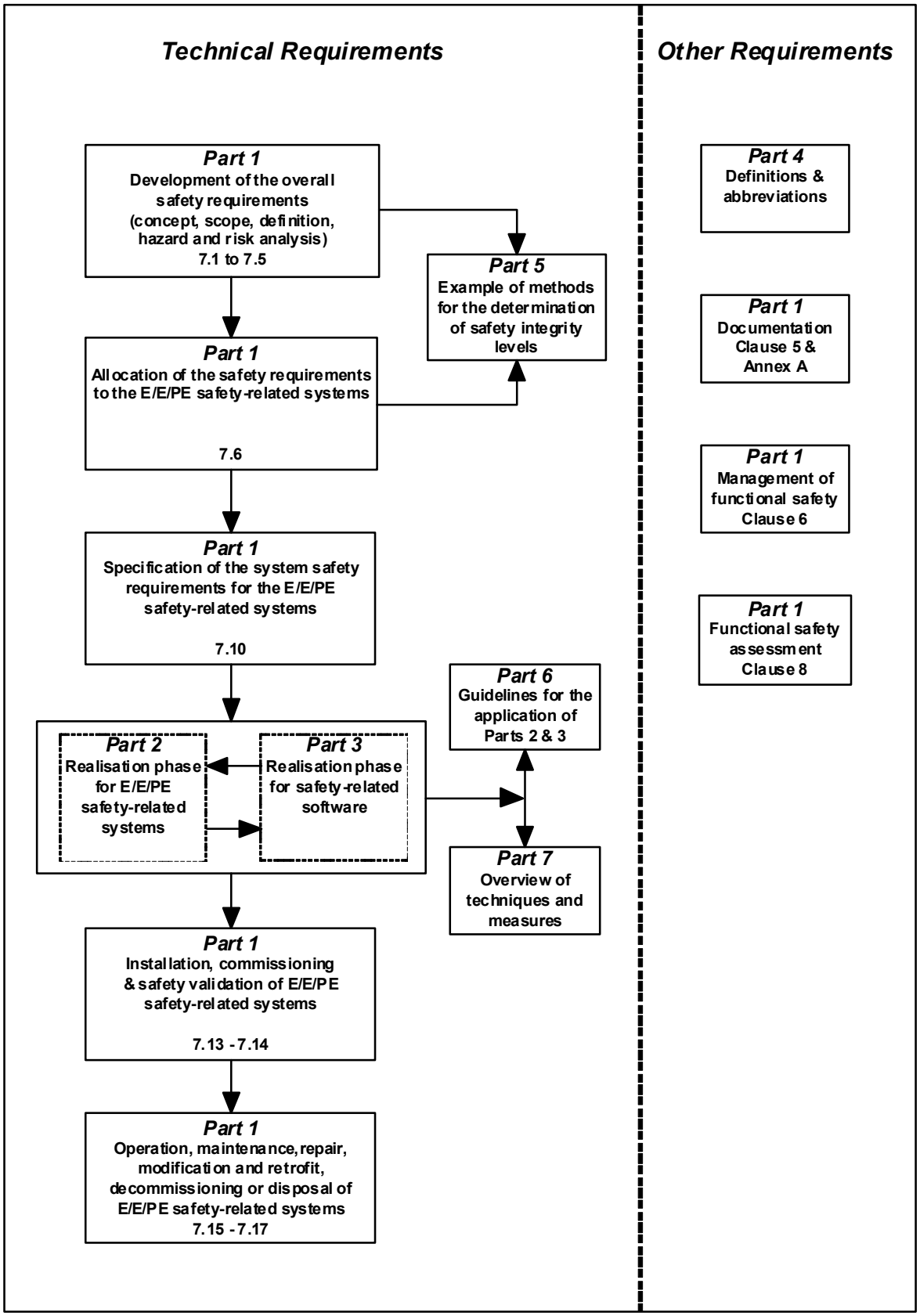


Figure 1 – Overall framework of IEC 61508

2 Normative references

The following referenced documents are indispensable for the application 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 61508-4:2010 *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 4: Definitions and abbreviations*

3 Definitions and abbreviations

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

Annex A
(informative)

Overview of techniques and measures for E/E/PE safety-related systems: control of random hardware failures (see IEC 61508-2)

A.1 Electric

Global objective: To control failures in electromechanical components.

A.1.1 Failure detection by on-line monitoring

NOTE This technique/measure is referenced in Tables A.2, A.3, A.7 and A.13 to A.18 of IEC 61508-2.

Aim: To detect failures by monitoring the behaviour of the E/E/PE safety-related system in response to the normal (on-line) operation of the equipment under control (EUC).

Description: Under certain conditions, failures can be detected using information about (for example) the time behaviour of the EUC. For example, if a switch, which is part of the E/E/PE safety-related system, is normally actuated by the EUC, then if the switch does not change state at the expected time, a failure will have been detected. It is not usually possible to localise the failure.

A.1.2 Monitoring of relay contacts

NOTE This technique/measure is referenced in Tables A.2 and A.14 of IEC 61508-2.

Aim: To detect failures (for example welding) of relay contacts.

Description: Forced contact (or positively guided contact) relays are designed so that their contacts are rigidly linked together. Assuming there are two sets of changeover contacts, *a* and *b*, if the normally open contact, *a*, welds, the normally closed contact, *b*, cannot close when the relay coil is next de-energised. Therefore, the monitoring of the closure of the normally closed contact *b* when the relay coil is de-energised may be used to prove that the normally open contact *a* has opened. Failure of normally closed contact *b* to close indicates a failure of contact *a*, so the monitoring circuit should ensure a safe shut-down, or ensure that shut-down is continued, for any machinery controlled by contact *a*.

References:

Zusammenstellung und Bewertung elektromechanischer Sicherheitsschaltungen für Verriegelungseinrichtungen. F. Kreutzkampf, W. Hertel, Sicherheitstechnisches Informations- und Arbeitsblatt 330212, BIA-Handbuch. 17. Lfg. X/91, Erich Schmidt Verlag, Bielefeld.
www.BGIA-HANDBUCHdigital.de/330212

A.1.3 Comparator

NOTE This technique/measure is referenced in Tables A.2, A.3, A.4 of IEC 61508-2.

Aim: To detect, as early as possible, (non-simultaneous) failures in an independent processing unit or in the comparator.

Description: The signals of independent processing units are compared cyclically or continuously by a hardware comparator. The comparator may itself be externally tested, or it may use self-monitoring technology. Detected differences in the behaviour of the processors lead to a failure message.

A.1.4 Majority voter

NOTE This technique/measure is referenced in Tables A.2, A.3 and A.4 of IEC 61508-2.

Aim: To detect and mask failures in one of at least three hardware channels.

Description: A voting unit using the majority principle (2 out of 3, 3 out of 3, or *m* out of *n*) is used to detect and mask failures. The voter may itself be externally tested, or it may use self-monitoring technology.

References:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.1.5 Idle current principle (de-energised to trip)

NOTE This technique/measure is referenced in Table A.16 of IEC 61508-2.

Aim: To execute the safety function if power is cut or lost.

Description: The safety function is executed if the contacts are open and no current flows. For example, if brakes are used to stop a dangerous movement of a motor, the brakes are opened by closing contacts in the safety-related system and are closed by opening the contacts in the safety-related system.

Reference:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.2 Electronic

Global objective: To control failure in solid-state components.

A.2.1 Tests by redundant hardware

NOTE This technique/measure is referenced in Tables A.3, A.15, A.16 and A.18 of IEC 61508-2.

Aim: To detect failures using hardware redundancy, i.e. using additional hardware not required to implement the process functions.

Description: Redundant hardware can be used to test at an appropriate frequency the specified safety functions. This approach is normally necessary for realising A.1.1 or A.2.2.

A.2.2 Dynamic principles

NOTE This technique/measure is referenced in Table A.3 of IEC 61508-2.

Aim: To detect static failures by dynamic signal processing.

Description: A forced change of otherwise static signals (internally or externally generated) helps to detect static failures in components. This technique is often associated with electromechanical components.

Reference:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld, 1993. http://www.bgia-handbuchdigital.de/330220

A.2.3 Standard test access port and boundary-scan architecture

NOTE This technique/measure is referenced in Tables A.3, A.15 and A.18 of IEC 61508-2.

Aim: To control and observe what happens at each pin of an IC.

Description: Boundary-scan test is an IC design technique which increases the testability of the IC by resolving the problem of how to gain access to the circuit test points within it. In a typical boundary-scan IC, comprised of core logic and input and output buffers, a shift-register stage is placed between the core logic and the input and output buffers adjacent to each IC pin. Each shift-register stage is contained in a boundary-scan cell. The boundary-scan cell can control and observe what happens at each input and output pin of an IC, via the standard test access port. Internal testing of the IC core logic is accomplished by isolating the on-chip core logic from stimuli received from surrounding components, and then performing an internal self-test. These tests can be used to detect failures in the IC.

Reference:

IEEE 1149-1:2001, *IEEE standard test access port and boundary-scan architecture, IEEE Computer Society*, 2001, ISBN: 0-7381-2944-5

A.2.4 (Not used)

A.2.5 Monitored redundancy

NOTE This technique/measure is referenced in Table A.3 of IEC 61508-2.

Aim: To detect failure, by providing several functional units, by monitoring the behaviour of each of these to detect failures, and by initiating a transition to a safe condition if any discrepancy in behaviour is detected.

Description: The safety function is executed by at least two hardware channels. The outputs of these channels are monitored and a safe condition is initiated if a fault is detected (i.e. if the output signals from all channels are not identical).

References:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld, 1993. http://www.bgia-handbuchdigital.de/330220

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

A.2.6 Electrical/electronic components with automatic check

NOTE This technique/measure is referenced in Table A.3 of IEC 61508-2.

Aim: To detect faults by periodic checking of the safety functions.

Description: The hardware is tested before starting the process, and is tested repeatedly at suitable intervals. The EUC continues to operate only if each test is successful.

References:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld, 1993. http://www.bgia-handbuchdigital.de/330220

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

A.2.7 Analogue signal monitoring

NOTE This technique/measure is referenced in Tables A.3 and A.13 of IEC 61508-2.

Aim: To improve confidence in measured signals.

Description: Wherever there is a choice, analogue signals are used in preference to digital on/off states. For example, trip or safe states are represented by analogue signal levels, usually with signal level tolerance monitoring. The technique provides continuity monitoring and a higher level of confidence in the transmitter, reducing the necessary proof-test frequency of the transmitter sensing function. External interfaces, for example impulse lines, will also require testing.

A.2.8 De-rating

NOTE This technique/measure is referenced in 7.4.2.13 of IEC 61508-2.

Aim: To increase the reliability of hardware components.

Description: Hardware components are operated at levels which are guaranteed by the design of the system to be well below the maximum specification ratings. De-rating is the practice of ensuring that under all normal operating circumstances, components are operated well below their maximum stress levels.

A.3 Processing units

Global objective: To recognise failures which lead to incorrect results in processing units.

A.3.1 Self-test by software: limited number of patterns (one-channel)

NOTE This technique/measure is referenced in Table A.4 of IEC 61508-2.

Aim: To detect, as early as possible, failures in the processing unit.

Description: The hardware is built using standard techniques which do not take any special safety requirements into account. The failure detection is realised entirely by additional software functions which perform self-tests using at least two complementary data patterns (for example 55hex and AAhex).

A.3.2 Self-test by software: walking bit (one-channel)

NOTE This technique/measure is referenced in Table A.4 of IEC 61508-2.

Aim: To detect, as early as possible, failures in the physical storage (for example registers) and instruction decoder of the processing unit.

Description: The failure detection is realised entirely by additional software functions which perform self-tests using a data pattern (for example walking-bit pattern) which tests the physical storage (data and address registers) and the instruction decoder. However, the diagnostic coverage is only 90 %.

A.3.3 Self-test supported by hardware (one-channel)

NOTE This technique/measure is referenced in Table A.4 of IEC 61508-2.

Aim: To detect, as early as possible, failures in the processing unit, using special hardware that increases the speed and extends the scope of failure detection.

Description: Additional special hardware facilities support self-test functions to detect failure. For example, this could be a hardware unit which cyclically monitors the output of a certain bit pattern according to the watch-dog principle.

A.3.4 Coded processing (one-channel)

NOTE This technique/measure is referenced in Table A.4 of IEC 61508-2.

Aim: To detect, as early as possible, failures in the processing unit.

Description: Processing units can be designed with special failure-recognising or failure-correcting circuit techniques. So far, these techniques have been applied only to relatively simple circuits and are not widespread; however, future developments should not be excluded.

References:

Le processeur codé: un nouveau concept appliqué à la sécurité des systèmes de transports. Gabriel, Martin, Wartski, Revue Générale des chemins de fer, No. 6, June 1990

Vital Coded Microprocessor Principles and Application for Various Transit Systems. P. Forin, IFAC Control Computers Communications in Transportation, 79-84, 1989

A.3.5 Reciprocal comparison by software

NOTE This technique/measure is referenced in Table A.4 of IEC 61508-2.

Aim: To detect, as early as possible, failures in the processing unit, by dynamic software comparison.

Description: Two processing units exchange data (including results, intermediate results and test data) reciprocally. A comparison of the data is carried out using software in each unit and detected differences lead to a failure message.

A.4 Invariable memory ranges

Global objective: The detection of information modifications in the invariable memory.

A.4.1 Word-saving multi-bit redundancy (for example ROM monitoring with a modified Hamming code)

NOTE 1 This technique/measure is referenced in Table A.5 of IEC 61508-2.

NOTE 2 See also A.5.6 “RAM monitoring with a modified Hamming code, or detection of data failures with error-detection-correction codes (EDC)” and C.3.2 “Error detecting and correcting codes”.

Aim: To detect all single-bit failures, all two-bit failures, some three-bit failures, and some all-bit failures in a 16-bit word.

Description: Every word of memory is extended by several redundant bits to produce a modified Hamming code with a Hamming distance of at least 4. Every time a word is read, checking of the redundant bits can determine whether or not a corruption has taken place. If a difference is found, a failure message is produced. The procedure can also be used to detect

addressing failures, by calculating the redundant bits for the concatenation of the data word and its address.

References:

Prüfbare und korrigierbare Codes. W. W. Peterson, München, Oldenburg, 1967

Error detecting and error correcting codes. R. W. Hamming, The Bell System Technical Journal 29 (2), 147-160, 1950

A.4.2 Modified checksum

NOTE This technique/measure is referenced in Table A.5 of IEC 61508-2.

Aim: To detect all odd-bit failures, i.e. approximately 50 % of all possible bit failures.

Description: A checksum is created by a suitable algorithm which uses all the words in a block of memory. The checksum may be stored as an additional word in ROM, or an additional word may be added to the memory block to ensure that the checksum algorithm produces a predetermined value. In a later memory test, a checksum is created again using the same algorithm, and the result is compared with the stored or defined value. If a difference is found, a failure message is produced.

A.4.3 Signature of one word (8-bit)

NOTE This technique/measure is referenced in Table A.5 of IEC 61508-2.

Aim: To detect all one-bit failures and all multi-bit failures within a word, as well as approximately 99,6 % of all possible bit failures.

Description: The contents of a memory block is compressed (using either hardware or software) using a cyclic redundancy check (CRC) algorithm into one memory word. A typical CRC algorithm treats the whole contents of the block as byte-serial or bit-serial data flow, on which a continued polynomial division is carried out using a polynomial generator. The remainder of the division represents the compressed memory contents – it is the "signature" of the memory – and is stored. The signature is computed once again in later tests and compared with one already stored. A failure message is produced if there is a difference.

A.4.4 Signature of a double word (16-bit)

NOTE This technique/measure is referenced in Table A.5 of IEC 61508-2.

Aim: To detect all one-bit failures and all multi-bit failures within a word, as well as approximately 99,998 % of all possible bit failures.

Description: This procedure calculates a signature using a cyclic redundancy check (CRC) algorithm, but the resulting value is at least two words in size. The extended signature is stored, recalculated and compared as in the single-word case. A failure message is produced if there is a difference between the stored and recalculated signatures.

A.4.5 Block replication (for example double ROM with hardware or software comparison)

NOTE This technique/measure is referenced in Table A.5 of IEC 61508-2.

Aim: To detect all bit failures.

Description: The address space is duplicated in two memories. The first memory is operated in the normal manner. The second memory contains the same information and is accessed in parallel to the first. The outputs are compared and a failure message is produced if a

difference is detected. In order to detect certain kinds of bit errors, the data must be stored inversely in one of the two memories and inverted once again when read.

A.5 Variable memory ranges

Global objective: Detecting failures during addressing, writing, storing and reading.

NOTE Soft-errors are listed in Table A.1, IEC 61508-2 as faults to be detected during operation or to be analysed in the derivation of the safe failure fraction. Causes of soft errors are: (1) Alpha particles from package decay, (2) Neutrons, (3) external EMI noise, (4) Internal cross-talk. External EMI noise is covered by other requirements of this international standard.

The effect of Alpha particles and Neutrons should be mastered by safety integrity measures at runtime. Safety integrity measures effective for hard errors may not be effective for soft errors, e.g. RAM tests, such as walk-path, galpat, etc. are not effective, whereas monitoring techniques such as Parity and ECC with recurring read of the memory cells are.

A soft error occurs when a radiation event causes enough of a charge disturbance to reverse or flip the data state of a low energized semiconductor memory cell, register, latch, or flip-flop. The error is called “soft” because the circuit itself is not permanently damaged by the radiation. Soft-errors are classified in Single Bit Upsets (SBU) or Single Event Upsets (SEU) and Multi-Bit Upsets (MBU).

If the disturbed circuit is a storage element like memory cell or flip-flop, the state is stored until the next (intended) write operation. The new data will be stored correctly. In a combinatory circuit the effect is rather a glitch because there is a continuous energy flow from the component driving this node. On connecting wires and communication lines the effect could also be a glitch. However due to the larger capacitance the effect by Alpha particles and Neutrons is considered negligible.

Soft-errors may be relevant to variable memory of any kind, i.e., to DRAM, SRAM, register banks in µP, cache, pipelines, configuration registers of devices such as ADC, DMA, MMU, Interrupt controller, complex timers. Sensitivity to alpha and neutron particles is a function of both core voltage and geometry. Smaller geometries at 2,5 V core voltage and especially below 1,8 V would require more evaluation and more effective protective measures.

The soft error rate has been reported (see a) and i) below) to be in a range of 700 Fit/MBit to 1 200 Fit/MBit for (embedded) memories. This is a reference value to be compared with data coming from the silicon process with which the device is implemented. Until recently SBU were considered to be dominant, but the latest forecast (see a) below) reports a growing percentage of MBU of the overall soft-error rate (SER) for technologies from 65 nm down.

The following literature and sources give details about soft-errors:

a) Altitude SEE Test European Platform (ASTEP) and First Results in CMOS 130 nm SRAM. J-L. Autran, P. Roche, C. Sudre et al. Nuclear Science, IEEE Transactions on Volume 54, Issue 4, Aug. 2007 Page(s):1002 - 1009

b) Radiation-Induced Soft Errors in Advanced Semiconductor Technologies, Robert C. Baumann, Fellow, IEEE, IEEE TRANSACTIONS ON DEVICE AND MATERIALS RELIABILITY, VOL. 5, NO. 3, SEPTEMBER 2005

c) Soft errors' impact on system reliability, Ritesh Mastipuram and Edwin C Wee, Cypress Semiconductor, 2004

d) Trends And Challenges In VLSI Circuit Reliability, C. Costantinescu, Intel, 2003, IEEE Computer Society

e) Basic mechanisms and modeling of single-event upset in digital microelectronics, P. E. Dodd and L. W. Massengill, IEEE Trans. Nucl. Sci., vol. 50, no. 3, pp. 583–602, Jun. 2003.

f) Destructive single-event effects in semiconductor devices and ICs, F. W. Sexton, IEEE Trans. Nucl. Sci., vol. 50, no. 3, pp. 603–621, Jun. 2003.

g) Coming Challenges in Microarchitecture and Architecture, Ronen, Mendelson, Proceedings of the IEEE, Volume 89, Issue 3, Mar 2001 Page(s):325 – 340

h) Scaling and Technology Issues for Soft Error Rates, A Johnston, 4th Annual Research Conference on Reliability Stanford University, October 2000

i) International Technology Roadmap for Semiconductors (ITRS), several papers.

A.5.1 RAM test "checkerboard" or "march"

NOTE This technique/measure is referenced in Table A.6 of IEC 61508-2.

Aim: To detect predominantly static bit failures.

Description: A checker-board type pattern of 0 s and 1 s is written into the cells of a bit-oriented memory. The cells are then inspected in pairs to ensure that the contents are the same and correct. The address of the first cell of such a pair is variable and the address of the second cell of the pair is formed by inverting bitwise the first address. In the first run, the address range of the memory is run towards higher addresses from the variable address, and in a second run towards lower addresses. Both runs are then repeated with an inverted pre-assignment. A failure message is produced if any difference occurs.

In a RAM test "march", the cells of a bit-oriented memory are initialised by a uniform bit stream. In the first run, the cells are inspected in ascending order: each cell is checked for the correct contents and its contents are inverted. The background, which is created in the first run, is treated in a second run in descending order and in the same manner. Both first runs are repeated with an inverted pre-assignment in a third or fourth run. A failure message is produced if a difference occurs.

A.5.2 RAM test "walkpath"

NOTE This technique/measure is referenced in Table A.6 of IEC 61508-2.

Aim: To detect static and dynamic bit failures and cross-talk between memory cells.

Description: The memory range to be tested is initialised by a uniform bit stream. The first cell is then inverted and the remaining memory area is inspected to ensure that the background is correct. After this, the first cell is re-inverted to return it to its original value, and the whole procedure is repeated for the next cell. A second run of the "wandering bit model" is carried out with an inverse background pre-assignment. A failure message is produced if a difference occurs.

A.5.3 RAM test "galpat" or "transparent galpat"

NOTE This technique/measure is referenced in Table A.6 of IEC 61508-2.

Aim: To detect static bit failures and a large proportion of dynamic couplings.

Description: In the RAM test "galpat", the chosen range of memory is first initialised uniformly (i.e. all 0 s or all 1 s). The first memory cell to be tested is then inverted and all the remaining cells are inspected to ensure that their contents are correct. After every read access to one of the remaining cells, the inverted cell is also checked. This procedure is repeated for each cell in the chosen memory range. A second run is carried out with the opposite initialisation. Any difference produces a failure message.

The "transparent galpat" test is a variation on the above procedure: instead of initialising all cells in the chosen memory range, the existing contents are left unchanged and signatures are used to compare the contents of sets of cells. The first cell to be tested in the chosen range is selected, and the signature S1 of all remaining cells in the range is calculated and stored. The cell to be tested is then inverted and the signature S2 of all the remaining cells is recalculated. (After every read access to one of the remaining cells, the inverted cell is also checked.) S2 is compared with S1, and any difference produces a failure message. The cell under test is re-inverted to re-establish the original contents, and the signature S3 of all the remaining cells is recalculated and compared with S1. Any difference produces a failure message. All memory cells in the chosen range are tested in the same manner.

A.5.4 RAM test "Abraham"

NOTE This technique/measure is referenced in Table A.6 of IEC 61508-2.

Aim: To detect all stuck-at and coupling failures between memory cells.

Description: The proportion of faults detected exceeds that of the RAM test "galpat". The number of operations required to perform the entire memory test is about 30 *n*, where *n* is the

number of cells in the memory. The test can be made transparent for use during the operating cycle by partitioning the memory and testing each partition in different time segments.

Reference:

Efficient Algorithms for Testing Semiconductor Random-Access Memories. R. Nair, S. M. Thatte, J. A. Abraham, IEEE Trans. Comput. C-27 (6), 572-576, 1978

A.5.5 One-bit redundancy (for example RAM monitoring with a parity bit)

NOTE This technique/measure is referenced in Table A.6 of IEC 61508-2.

Aim: To detect 50 % of all possible bit failures in the memory range tested.

Description: Every word of the memory is extended by one bit (the parity bit) which completes each word to an even or odd number of logical 1 s. The parity of the data word is checked each time it is read. If the wrong number of 1 s is found, a failure message is produced. The choice of even or odd parity should be made such that, whichever of the zero word (nothing but 0 s) and the one word (nothing but 1 s) is the more unfavourable in the event of a failure, then that word is not a valid code. Parity can also be used to detect addressing failures, when the parity is calculated for the concatenation of the data word and its address.

A.5.6 RAM monitoring with a modified Hamming code, or detection of data failures with error-detection-correction codes (EDC)

NOTE 1 This technique/measure is referenced in Table A.6 of IEC 61508-2.

NOTE 2 See also A.4.1 “Word-saving multi-bit redundancy (for example ROM monitoring with a modified Hamming code)”and C.3.2 “Error detecting and correcting codes”.

Aim: To detect all odd-bit failures, all two-bit failures, some three-bit and some multi-bit failures.

Description: Every access to memory is extended by several redundant bits to produce a modified Hamming code with a Hamming distance of at least 4. Every time data is read, one can determine whether a corruption has taken place by checking the redundant bits. If a difference is found, a failure message is produced. The procedure can also be used to detect addressing failure, when the redundant bits are calculated for the concatenation of the data and its address.

References:

Prüfbare und korrigierbare Codes. W. W. Peterson, München, Oldenburg, 1967

Error detecting and error correcting codes. R. W. Hamming, The Bell System Technical Journal 29 (2), 147-160, 1950

A.5.7 Double RAM with hardware or software comparison and read/write test

NOTE This technique/measure is referenced in Table A.6 of IEC 61508-2.

Aim: To detect all bit failures.

Description: The address space is duplicated in two memories. The first memory is operated in the normal manner. The second memory contains the same information and is accessed in parallel to the first. The outputs are compared and a failure message is produced if a difference is detected. In order to detect certain kinds of bit errors, the data must be stored inversely in one of the two memories and inverted once again when read.

A.6 I/O-units and interfaces (external communication)

Global objective: To detect failures in input and output units (digital, analogue, serial or parallel) and to prevent the sending of inadmissible outputs to the process.

A.6.1 Test pattern

NOTE This technique/measure is referenced in Tables A.7, A.13 and A.14 of IEC 61508-2.

Aim: To detect static failures (stuck-at failures) and cross-talk.

Description: This is a dataflow-independent cyclical test of input and output units. It uses a defined test pattern to compare observations with the corresponding expected values. The test pattern information, the test pattern reception, and test pattern evaluation must all be independent of each other. The EUC should not be inadmissibly influenced by the test pattern.

A.6.2 Code protection

NOTE This technique/measure is referenced in Tables A.7, A.15, A.16 and A.18 of IEC 61508-2.

Aim: To detect random hardware and systematic failures in the input/output dataflow.

Description: This procedure protects the input and output information from both systematic and random hardware failures. Code protection provides dataflow-dependent failure detection of the input and output units, based on information redundancy and/or time redundancy. Typically, redundant information is superimposed on input and/or output data. This then provides a means to monitor the correct operation of the input or output circuits. Many techniques are possible, for example a carrier frequency signal may be superimposed on the output signal of a sensor. The logic unit may then check for the presence of the carrier frequency or redundant code bits may be added to an output channel to allow the monitoring of the validity of a signal passing between the logic unit and final actuator.

A.6.3 Multi-channel parallel output

NOTE This technique/measure is referenced in Table A.7 of IEC 61508-2.

Aim: To detect random hardware failures (stuck-at failures), failures caused by external influences, timing failures, addressing failures, drift failures and transient failures.

Description: This is a dataflow-dependent multi-channel parallel output with independent outputs for the detection of random hardware failures. Failure detection is carried out via external comparators. If a failure occurs, the EUC is switched off directly. This measure is only effective if the dataflow changes during the diagnostic test interval.

A.6.4 Monitored outputs

NOTE This technique/measure is referenced in Table A.7 of IEC 61508-2.

Aim: To detect individual failures, failures caused by external influences, timing failures, addressing failures, drift failures (for analogue signals) and transient failures.

Description: This is a dataflow-dependent comparison of outputs with independent inputs to ensure compliance with a defined tolerance range (time, value). A detected failure cannot always be related to the defective output. This measure is only effective if the dataflow changes during the diagnostic test interval.

A.6.5 Input comparison/voting

NOTE This technique/measure is referenced in Tables A.7 and A.13 of IEC 61508-2.

Aim: To detect individual failures, failures caused by external influences, timing failures, addressing failures, drift failures (for analogue signals) and transient failures.

Description: This is a dataflow-dependent comparison of independent inputs to ensure compliance with a defined tolerance range (time, value). There will be 1 out of 2, 2 out of 3 or better redundancy. This measure is only effective if the dataflow changes during the diagnostic test interval.

A.7 Data paths (internal communication)

Global objective: To detect failures caused by a defect in the information transfer.

A.7.1 One-bit hardware redundancy

NOTE This technique/measure is referenced in Table A.8 of IEC 61508-2.

Aim: To detect all odd-bit failures, i.e. 50 % of all the possible bit failures in the data stream.

Description: The bus is extended by one line (bit) and this additional line (bit) is used to detect failures by parity checking.

A.7.2 Multi-bit hardware redundancy

NOTE This technique/measure is referenced in Table A.8 of IEC 61508-2.

Aim: To detect failures during the communication on the bus and in serial transmission links.

Description: The bus is extended by two or more lines (bits) and these additional lines (bits) are used in order to detect failures by Hamming code techniques.

A.7.3 Complete hardware redundancy

NOTE This technique/measure is referenced in Table A.8 of IEC 61508-2.

Aim: To detect failures during the communication by comparing the signals on two buses.

Description: The bus is doubled and the additional lines (bits) are used in order to detect failures.

A.7.4 Inspection using test patterns

NOTE This technique/measure is referenced in Table A.8 of IEC 61508-2.

Aim: To detect static failures (stuck-at failure) and cross-talk.

Description: This is a dataflow-independent cyclical test of data paths. It uses a defined test pattern to compare observations with the corresponding expected values.

The test pattern information, the test pattern reception, and test pattern evaluation must all be independent of each other. The EUC should not be inadmissibly influenced by the test pattern.

A.7.5 Transmission redundancy

NOTE This technique/measure is referenced in Table A.8 of IEC 61508-2.

Aim: To detect transient failures in bus communication.

Description: The information is transferred several times in sequence. The repetition is effective only against transient failures.

A.7.6 Information redundancy

NOTE This technique/measure is referenced in Table A.8 of IEC 61508-2.

Aim: To detect failures in bus communication.

Description: Data is transmitted in blocks, together with a calculated checksum for each block. The receiver then re-calculates the checksum of the received data and compares the result with the received checksum.

A.8 Power supply

Global objective: To detect or tolerate failures caused by a defect in the power supply.

A.8.1 Overvoltage protection with safety shut-off

NOTE This technique/measure is referenced in Table A.9 of IEC 61508-2.

Aim: To protect the safety-related system against overvoltage.

Description: Overvoltage is detected early enough that all outputs can be switched to a safe condition by the power-down routine or there is a switch-over to a second power unit.

Reference:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.8.2 Voltage control (secondary)

NOTE This technique/measure is referenced in Table A.9 of IEC 61508-2.

Aim: To monitor the secondary voltages and initiate a safe condition if the voltage is not in its specified range.

Description: The secondary voltage is monitored and a power-down is initiated, or there is a switch-over to a second power unit, if it is not in its specified range.

A.8.3 Power-down with safety shut-off

NOTE This technique/measure is referenced in Table A.9 of IEC 61508-2.

Aim: To shut off the power with all safety critical information stored.

Description: Overvoltage or undervoltage is detected early enough so that the internal state can be saved in non-volatile memory (if necessary), and so that all outputs can be set to a safe condition by the power-down routine, or that all outputs can be switched to a safe condition by the power-down routine, or there is a switch-over to a second power unit.

A.9 Temporal and logical program sequence monitoring

NOTE This group of techniques and measures is referenced in Tables A.15, A.16 and A.18 of IEC 61508-2.

Global objective: To detect a defective program sequence. A defective program sequence exists if the individual elements of a program (for example software modules, subprograms or

commands) are processed in the wrong sequence or period of time, or if the clock of the processor is faulty.

A.9.1 Watch-dog with separate time base without time-window

NOTE This technique/measure is referenced in Tables A.10 and A.11 of IEC 61508-2.

Aim: To monitor the behaviour and the plausibility of the program sequence.

Description: External timing elements with a separate time base (for example watch-dog timers) are periodically triggered to monitor the computer's behaviour and the plausibility of the program sequence. It is important that the triggering points are correctly placed in the program. The watch-dog is not triggered at a fixed period, but a maximum interval is specified.

A.9.2 Watch-dog with separate time base and time-window

NOTE This technique/measure is referenced in Tables A.10 and A.11 of IEC 61508-2.

Aim: To monitor the behaviour and the plausibility of the program sequence.

Description: External timing elements with a separate time base (for example watch-dog timers) are periodically triggered to monitor the computer's behaviour and the plausibility of the program sequence. It is important that the triggering points are correctly placed in the program. A lower and upper limit is given for the watch-dog timer. If the program sequence takes a longer or shorter time than expected, emergency action is taken.

A.9.3 Logical monitoring of program sequence

NOTE This technique/measure is referenced in Tables A.10 and A.11 of IEC 61508-2.

Aim: To monitor the correct sequence of the individual program sections.

Description: The correct sequence of the individual program sections is monitored using software (counting procedure, key procedure) or using external monitoring facilities. It is important that the checking points are placed in the program correctly.

A.9.4 Combination of temporal and logical monitoring of program sequences

NOTE This technique/measure is referenced in Tables A.10 and A.11 of IEC 61508-2.

Aim: To monitor the behaviour and the correct sequence of the individual program sections.

Description: A temporal facility (for example a watch-dog timer) monitoring the program sequence is retriggered only if the sequence of the program sections is also executed correctly.

A.9.5 Temporal monitoring with on-line check

NOTE This technique/measure is referenced in Tables A.10 and A.11 of IEC 61508-2.

Aim: To detect faults in the temporal monitoring.

Description: The temporal monitoring is checked at start-up, and a start is only possible if the temporal monitoring operates correctly. For example, a heat sensor could be checked by a heated resistor at start-up.

A.10 Ventilation and heating

NOTE This group of techniques and measures is referenced in Tables A.16 and A.18 of IEC 61508-2.

Global objective: To control failures in the ventilation or heating, and/or their monitoring, if this is safety-related.

A.10.1 Temperature sensor

Aim: To detect over- or under-temperature before the system begins to operate outside specification.

Description: A temperature sensor monitors temperature at the most critical points of the E/E/PE safety-related system. Before the temperature leaves the specified range, emergency action is taken.

A.10.2 Fan control

Aim: To detect incorrect operation of the fans.

Description: The fans are monitored for correct operation. If a fan is not working properly, maintenance (or ultimately, emergency) action is taken.

A.10.3 Actuation of the safety shut-off via thermal fuse

Aim: To shut off the safety-related system before the system works outside of its thermal specification.

Description: A thermal fuse is used to shut off the safety-related system. For a PES, the shut-off is introduced by a power-down routine which stores all information necessary for emergency action.

A.10.4 Staggered message from thermo-sensors and conditional alarm

Aim: To indicate that the safety-related system is working outside its thermal specification.

Description: The temperature is monitored and an alarm is raised if the temperature is outside of a specified range.

A.10.5 Connection of forced-air cooling and status indication

Aim: To prevent overheating by forced-air cooling.

Description: The temperature is monitored and forced-air cooling is introduced if the temperature is higher than a specified limit. The user is informed of the status.

A.11 Communication and mass-storage

Global objective: To control failures during communication with external sources and mass-storage.

A.11.1 Separation of electrical energy lines from information lines

NOTE This technique/measure is referenced in Table A.16 of IEC 61508-2.

Aim: To minimise cross-talk induced by high currents in the information lines.

Description: Electrical energy supply lines are separated from the lines carrying the information. The electrical field which could induce voltage spikes on the information lines decreases with distance.

A.11.2 Spatial separation of multiple lines

NOTE This technique/measure is referenced in Tables A.16 of IEC 61508-2.

Aim: To minimise cross-talk induced by high currents in multiple lines.

Description: Lines carrying duplicated signals are separated from each other. The electrical field which could induce voltage spikes on the multiple lines decreases with the distance. This measure also reduces common cause failures.

A.11.3 Increase of interference immunity

NOTE This technique/measure is referenced in Tables A.16 and A.18 of IEC 61508-2.

Aim: To minimise electromagnetic interference on the safety-related system.

Description: Design techniques such as shielding and filtering are used to increase the interference immunity of the safety-related system to electromagnetic disturbances which may be radiated or conducted on power or signal lines, or result from electrostatic discharges.

NOTE See [16] and [17] for immunity requirements for safety-related systems and for equipment intended to perform safety-related functions (functional safety) in industrial applications.

References:

IEC/TR 61000-5-2:1997, *Electromagnetic compatibility (EMC) – Part 5: Installation and mitigation guidelines – Section 2: Earthing and cabling*

Principles and Techniques of Electromagnetic Compatibility, Second Edition, C. Christopoulos, CRC Press, 2007, ISBN-10: 0849370353, ISBN-13: 978-0849370359

Noise Reduction Techniques in Electronic Systems. H. W. Ott, John Wiley Interscience, 2nd Edition, 1988

EMC for Product Designers. T. Williams, Newnes, 2007, ISBN 0750681705

Grounding and Shielding Techniques in Instrumentation, 3rd edition, R. Morrison . Wiley-Interscience, New York, 1986, ISBN-10: 0471838055, ISBN-13: 978-0471838050

A.11.4 Antivalent signal transmission

NOTE This technique/measure is referenced in Tables A.7 and A.16 of IEC 61508-2.

Aim: To detect the same induced voltages in multiple signal transmission lines.

Description: All duplicated information is transmitted with antivalent signals (for example logic 1 and 0). Common cause failures (for example by electromagnetic emission) can be detected by an antivalent comparator.

Reference:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch. 20. Lfg. V/93, Erich Schmidt Verlag, Bielefeld. http://www.bgia-handbuchdigital.de/330220

A.12 Sensors

Global objective: To control failures in the sensors of the safety-related system.

A.12.1 Reference sensor

NOTE This technique/measure is referenced in Table A.13 of IEC 61508-2.

Aim: To detect the incorrect operation of a sensor.

Description: An independent reference sensor is used to monitor the operation of a process sensor. All input signals are checked at suitable time intervals by the reference sensor to detect failures of the process sensor.

Reference:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.12.2 Positive-activated switch

NOTE This technique/measure is referenced in Table A.13 of IEC 61508-2.

Aim: To open a contact by a direct mechanical connection between switch cam and contact.

Description: A positive-activated switch opens its normally closed contacts by a direct mechanical connection between switch cam and contact. This ensures that whenever the switch cam is in the operated position, the switch contacts must be open.

Reference:

Verriegelung beweglicher Schutzeinrichtungen. F. Kreutzkampf, K. Becker, Sicherheitstechnisches Informations- und Arbeitsblatt 330210, BIA-Handbuch. 1. Lfg. IX/85, Erich Schmidt Verlag, Bielefeld

A.13 Final elements (actuators)

Global objective: To control failures in the final elements in the safety-related system.

A.13.1 Monitoring

NOTE This technique/measure is referenced in Table A.14 of IEC 61508-2.

Aim: To detect the incorrect operation of an actuator.

Description: The operation of the actuator is monitored (for example by the positively activated contacts of a relay, see monitoring of relay contacts in A.1.2). The redundancy introduced by this monitoring can be used to trigger emergency action.

References:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

Zusammenstellung und Bewertung elektromechanischer Sicherheitsschaltungen für Verriegelungseinrichtungen. F. Kreutzkampf, W. Hertel, Sicherheitstechnisches Informations- und Arbeitsblatt 330212, BIA-Handbuch. 17. Lfg. X/91, Erich Schmidt Verlag, Bielefeld

A.13.2 Cross-monitoring of multiple actuators

NOTE This technique/measure is referenced in Table A.14 of IEC 61508-2.

Aim: To detect faults in actuators by comparing the results.

Description: Each multiple actuator is monitored by a different hardware channel. If a discrepancy occurs, emergency action is taken.

A.14 Measures against the physical environment

NOTE This technique/measure is referenced in Tables A.16 and A.18 of IEC 61508-2

Aim: To prevent influences of the physical environment (water, dust, corrosive substances) causing failures.

Description: The enclosure of the equipment is designed to withstand the expected environment.

Reference:

IEC 60529:1989, *Degrees of protection provided by enclosures (IP Code)*

Annex B
(informative)

Overview of techniques and measures for E/E/PE safety related systems: avoidance of systematic failures (see IEC 61508-2 and IEC 61508-3)

NOTE Many techniques in this annex are applicable to software but have not been duplicated in Annex C.

B.1 General measures and techniques

B.1.1 Project management

NOTE This technique/measure is referenced in Tables B.1 to B.6 of IEC 61508-2.

Aim: To avoid failures by adoption of an organisational model and rules and measures for development and testing of safety-related systems.

Description: The most important and best measures are

- the creation of an organisational model, especially for quality assurance which is set down in a quality assurance handbook; and
- the establishment of regulations and measures for the creation and validation of safety-related systems in cross-project and project-specific guidelines.

A number of important basic principles are set down in the following:

- definition of a design organisation:
  - tasks and responsibilities of the organisational units,
  - authority of the quality assurance departments,
  - independence of quality assurance (internal inspection) from development;
- definition of a sequence plan (activity models):
  - determination of all activities which are relevant during execution of the project including internal inspections and their scheduling,
  - project update;
- definition of a standardised sequence for an internal inspection:
  - planning, execution and checking of the inspection (inspection theory),
  - releasing mechanisms for subproducts,
  - the safekeeping of repeat inspections;
- configuration management:
  - administration and checking of versions,
  - detection of the effects of modifications,
  - consistency inspections after modifications;
- introduction of a quantitative assessment of quality assurance measures:
  - requirement acquisition,
  - failure statistics;
- introduction of computer-aided universal methods, tools and training of personnel.

References:

ISO 9001:2008, *Quality management systems – Requirements*

ISO/IEC 15504 (all parts), *Information technology – Process assessment*

CMMI: Guidelines for Process Integration and Product Improvement, 2nd Edition. M. B. Chrissis, M. Konrad, S. Shrum, Addison-Wesley Professional, 2006, ISBN-10: 0-3212-7967-0, ISBN-13: 978-0-3212-7967-5

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

B.1.2 Documentation

NOTE 1 This technique/measure is referenced in Tables B.1 to B.6 of IEC 61508-2.

NOTE 2 See also Clause 5 and Annex A of IEC 61508-1.

Aim: To avoid failures and facilitate assessment of system safety, by documenting each step during development.

Description: The operational capacity and safety, as well as the care taken in development by all parties involved, has to be demonstrated during assessment. In order to be able to show the development care, and in order to guarantee the verification of the evidence of safety at any time, special importance is given to the documentation.

Important common measures are the introduction of guidelines and computer aid, i.e.

- guidelines, which
  - specify a grouping plan;
  - ask for checklists for the contents; and
  - determine the form of the document;
- administration of the documentation with the help of a computer-aided and organised project library.

Individual measures are:

- separation in the documentation
  - of the requirements,
  - of the system (user-documentation) and
  - of the development (including internal inspection);
- grouping of the development documentation according to the safety lifecycle;
- definition of standardised documentation modules, from which the documents can be compiled;
- clear identification of the constituent parts of the documentation;
- formalised revision update;
- selection of clear and intelligible means of description:
  - formal notation for determinations;
  - natural language for introductions, justifications and representations of intentions;

- graphical representations for examples;
- semantic definition of graphical elements; and
- directories of specialised words.

References:

IEC 61506:1997, *Industrial-process measurement and control – Documentation of application software*

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.1.3 Separation of E/E/PE system safety functions from non-safety functions

NOTE This technique/measure is referenced in Tables B.1 and B.6 of IEC 61508-2.

Aim: To prevent the non-safety-related part of the system from influencing the safety-related part in undesired ways.

Description: In the specification, it should be decided whether a separation of the safety-related systems and non-safety-related systems is possible. Clear specifications should be written for the interfacing of the two parts. A clear separation reduces the effort for testing the safety-related systems.

Reference:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.1.4 Diverse hardware

NOTE This technique/measure is referenced in Tables A.15, A.16 and A.18 of IEC 61508-2.

Aim: To detect systematic failures during operation of the EUC, using diverse components with different rates and types of failures.

Description: Different types of components are used for the diverse channels of a safety-related system. This reduces the probability of common cause failures (for example overvoltage, electromagnetic interference), and increases the probability of detecting such failures.

Existence of different means of performing a required function, for example different physical principles, offer other ways of solving the same problem. There are several types of diversity. Functional diversity employs the use of different approaches to achieve the same result.

Reference :

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.2 E/E/PE system design requirements specification

Global objective: To produce a specification which is, as far as possible, complete, free from mistakes, free from contradiction, and simple to verify.

B.2.1 Structured specification

NOTE This technique/measure is referenced in Tables B.1 and B.6 of IEC 61508-2.

Aim: To reduce complexity by creating a hierarchical structure of partial requirements. To avoid interface failures between the requirements.

Description: This technique structures the functional specification into partial requirements such that the simplest possible, visible relations exist between the latter. This analysis is successively refined until small clear partial requirements can be distinguished. The result of the final refinement is a hierarchical structure of partial requirements which provide a framework for the specification of the complete requirements. This method emphasises the interfaces of the partial requirements and is particularly effective for avoiding interface failures.

References:

ESA PSS 05-02, *Guide to the user requirements definition phase*, Issue 1, Revision 1, ESA Board for Software Standardisation and Control (BSSC), ESA, Paris, March 1995, ftp://ftp.estec.esa.nl/pub/wm/wme/bssc/PSS0502.pdf

Structured Analysis and System Specification. T. De Marco, Yourdon Press, Englewood Cliffs, 1979, ISBN-10: 0138543801, ISBN-13: 978-0138543808

B.2.2 Formal methods

NOTE 1 See C.2.4 for details of specific formal methods.

NOTE 2 This technique/measure is referenced in Tables B.1, B.2 and B.6 of IEC 61508-2.

Aim: Formal methods transfers the principles of mathematical reasoning to the specification and implementation of technical systems therefore increase the completeness, consistency or correctness of a specification or implementation.

Description: Formal methods provide a means of developing a description of a system during specification and/or implementation phase. These formal descriptions are mathematical models of the system function and/or structure.

Therefore unambiguous system description could be achieved (e.g. any state of an automaton is described by its initial state, inputs and the transition equations of the automaton) which increase understanding of the underlying system.

Choosing a suitable formal method is a difficult undertaking requiring full understanding of system, its development process and the range of mathematical models that could possibly be used (see following notes).

NOTE 3 The theorems of interest of the model (properties) represent guaranties about the system which provides far more confidence than simulation i.e. observing selected actions of the system.

NOTE 4 The disadvantages of formal methods can be:

– Fixed level of abstraction;
– limitations to capture all functionality that is relevant at the given stage;
– difficulty that implementation engineers have to understand the model;
– high efforts to develop, analyze and maintain model over the lifecycle of system;
– availability of efficient tools which support the building and analysis of model;
– availability of staff capable to develop and analyze model.

NOTE 5 The formal methods community's focus clearly was been the modeling of the target function of system often deemphasizing the fault robustness of a system. Therefore respective formal methods including system robustness has to be selected.

Reference:

Formal Specification: Techniques and Applications. N.Nissanke, Springer-Verlag Telos, 1999, ISBN-10: 1852330023

B.2.3 Semi-formal methods

NOTE 1 IEC 61508- 3 Table B.7 extends this Annex B list with other semi-formal software related techniques. art 3 lists:

– logic/function block diagrams: described in IEC 61131-3;
– sequence diagrams: described in IEC 61131-3;
– data flow diagrams: see C.2.2;
– finite state machines/state transition diagrams: see B.2.3.2;
– time Petri nets: see B.2.3.3;
– entity-relationship-attribute Data models: see B.2.4.4;
– message sequence charts: see C.2.14;
– decision/truth tables: see C.6.1.

Aim: To express parts of a specification unambiguously and consistently, so that some mistakes, omissions and wrong behaviour can be detected.

NOTE 2 This technique/measure is referenced in Tables B.1, B.2 and B.6 of IEC 61508-2 and in Tables A.1, A.2, A.4, B.7, C.1, C.2, C.4 and C.17 of IEC 61508-3.

B.2.3.1 General

Aim: To prove that the design meets its specification.

Description: Semi-formal methods provide a means of developing a description of a system at some stage in its development, i.e. specification, design or coding. The description can in some cases be analysed by machine or animated to display various aspects of the system behaviour. Animation can give extra confidence that the system meets the real requirement as well as the specified requirement.

Two semi-formal methods are described in the following subclauses.

B.2.3.2 Finite state machines / state transition diagrams

NOTE This technique/measure is referenced in Tables B.5, B.7, C.15 and C.17 of IEC 61508-3.

Aim: To model, verify, specify or implement the control structure of a system.

Description: Many systems can be described in terms of their states, their inputs, and their actions. Thus when in state S1, on receiving input I a system might carry out action A and move to state S2. By describing a system's actions for every input in every state we can describe a system completely. The resulting model of the system is called a finite state machine (or finite state automata). It is often drawn as a so-called state transition diagram showing how the system moves from one state to another, or as a matrix in which the dimensions are state and input, and the matrix cells contain the action and new state resulting from receiving the input when in the given state.

Where a system is complicated or has a natural structure, this can be reflected in a layered finite state machine. A Statechart is a type of state transition diagram in which nested states are allowed (the object state splits into two or more sub-states which can evolve in parallel, and possibly recombine into a single state at some point); this adds to the expressive power of the state transition notation but can add extra complexity which may be undesirable in a safety related system. Statecharts have a formal (mathematical) specification. State transition diagrams can apply to the whole system or to some object or element within it.

A specification or design expressed as a finite state machine can be checked for

- completeness (the system or object must have an action and new state for every input in every state);
- consistency (only one state transition is possible for each state/input pair); and
- reachability (whether or not it is possible to get from one state to another by any sequence of inputs); and
- absence of endless loops or dead-end states; etc.

These are important properties for critical systems. Tools to support these checks are easily developed and various models based on finite state automata (formal languages, Petri nets, Markov graphs, etc.) can be used. Algorithms also exist that allow the automatic generation of test cases for verifying a finite state machine implementation or for animating a finite state machine model. State transition diagrams and Statecharts are widely supported by tools which allow the diagrams to be drawn and checked, and which will generate code to implement the described state machine.

They can also be used for failure probability calculations, see B.6 and C.6.

References:

Introduction to Automata Theory, Languages, and Computation (3rd Edition). J. Hopcroft, R. Motwani, J. Ullman, Addison-Wesley Longman Publishing Co, 2006, ISBN: 0321462254

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.2.3.3 Time Petri nets

NOTE This technique/measure is referenced in Tables B.5, B.7, C.15 and C.17 of IEC 61508-3.

Aim: To model relevant aspects of the system behaviour and to assess and possibly improve safety and operational requirements through analysis and re-design.

Description: Petri nets are a particular case of finite state automata. They belong to a class of graph theoretic models which are suitable for representing information and control flow in systems that exhibit concurrency and have asynchronous behaviour.

A Petri net is a network of places and transitions. The places may be "marked" or "unmarked". A transition is "enabled" when all the input places to it are marked. When enabled, it is permitted (but not obliged) to "fire". If it fires, the input places to the transition become unmarked, and each output place from the transition is marked instead.

The potential hazards can be represented as particular states (markings) in the model. The Petri net model can be extended to allow for timing features of the system. Although "classical" Petri nets concentrate on control flow aspects, several extensions have been proposed to incorporate data flow into the model.

They also provide a very efficient support for performing Monte Carlo simulation in order to achieve failure probability calculations, see B.6.6.8.

References:

Timed Petri Nets: Theory and Application. Jiacun Wang, Springer, 1998, ISBN 0792382706

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.2.4 Computer-aided specification tools

NOTE This technique/measure is referenced in Tables B.1 and B.6 of IEC 61508-2 and in Tables A.1, A.2, C.1 and C.2 of IEC 61508-3.

B.2.4.1 General

Aim: To use formal specification techniques to facilitate automatic detection of ambiguity and completeness.

Description: The technique produces a specification in the form of a database that can be automatically inspected to assess consistency and completeness. The specification tool can animate various aspects of the specified system to the user. In general, the technique supports not only the creation of the specification but also of the design and other phases of the project lifecycle. Specification tools can be classified according to the following subclauses.

B.2.4.2 Tools oriented towards no specific method

Aim: To help the user write a good specification by providing prompts and links between related parts.

Description: The specification tool takes over some routine work from the user and supports the project management. It does not enforce any particular specification methodology. The relative independence with regard to method allows users a great deal of freedom but also gives them little of the specialised support necessary when creating specifications. This makes familiarisation with the system more difficult.

B.2.4.3 Model orientated procedure with hierarchical analysis

Aim: To avoid incompleteness, ambiguity and contradiction in the specification, e.g. supporting the user writing a good specification by ensuring consistency between descriptions of actions and data at various levels of abstraction.

Description: This method gives a functional representation of the desired system (structured analysis) at various levels of abstraction (degree of precision). There is a huge arsenal of such models: finite automata are a class of such models widely used to describe the evolution of discrete/digital systems. Differential equations are similar in spirit and aim at continuous/analogue systems. The analysis at various levels acts on both actions and data. Assessment of ambiguity and completeness is possible between hierarchical levels as well as between two functional units (modules) on the same level (e.g. any state of a system model is described by its initial state, inputs and the transition equations of the automaton).

NOTE Issues of model based descriptions may be the level of abstraction, limitations to capture all functionality that is relevant at the given stage, difficulty that practitioners have to understand the model (from reading the syntax to understanding), high efforts to develop, analyze and maintain a model over the lifecycle of a system, availability of efficient tools which support the building and analysis of model (development of such tools is certainly a high effort undertaking) and availability of staff capable to develop and analyze models.

Reference:

System requirements analysis. Jeffrey O. Grady, Academic Press, 2006, ISBN 012088514X, 9780120885145

B.2.4.4 Entity-relationship-attribute data models

Aim: To help the user write a good specification by focusing on entities within the system and relationships between them.

Description: The desired system is described as a collection of objects and their relationships. The tool enables one to determine which relationships can be interpreted by the

system. In general, the relationships permit a description of the hierarchical structure of the objects, the data flow, the relationships between the data, and which data are subject to certain manufacturing processes. The classical procedure has been extended for process control applications. Inspection capabilities and support for the user depend on the variety of relationships illustrated. On the other hand, a large number of representation possibilities makes the application of this technique complex.

Reference:

Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle. Karl Eugene Wiegers, Microsoft Press, 2003, ISBN 0735618798, 9780735618794

B.2.4.5 Incentive and answer

Aim: To help the user write a good specification by identifying stimulus-response relationships.

Description: The relationships between the objects of the system are specified in a notation of "incentives" and "answers". A simple and easily expanded language is used which contains language elements which represent objects, relationships, characteristics and structures.

B.2.5 Checklists

NOTE This technique/measure is referenced in Tables B.1, B.2 and B.6 of IEC 61508-2 and in Tables A.10, B.8, C.10 and C.18 of IEC 61508-3.

Aim: To draw attention to, and manage critical appraisal of, all important aspects of the system by safety lifecycle phase, ensuring comprehensive coverage without laying down exact requirements.

Description: A set of questions to be answered by the person performing the checklist. Many of the questions are of a general nature and the assessor must interpret them as seems most appropriate to the particular system being assessed. Checklists can be used for all phases of the overall, E/E/PE system safety and software safety lifecycles and are particularly useful as a tool to aid the functional safety assessment.

To accommodate wide variations in systems being validated, most checklists contain questions which are applicable to many types of system. As a result, there may well be questions in the checklist being used which are not relevant to the system being dealt with and which should be ignored. Equally there may be a need, for a particular system, to supplement the standard checklist with questions specifically directed at the system being dealt with.

In any case it should be clear that the use of checklists depends critically on the expertise and judgement of the engineer selecting and applying the checklist. As a result, the decisions taken by the engineer, with regard to the checklist(s) selected, and any additional or superfluous questions, should be fully documented and justified. The objective is to ensure that the application of the checklists can be reviewed and that the same results will be achieved unless different criteria are used.

The object in completing a checklist is to be as concise as possible. When extensive justification is necessary this should be done by reference to additional documentation. Pass, fail and inconclusive, or some similar restricted set of responses should be used to document the results for each question. This conciseness greatly simplifies the procedure of reaching an overall conclusion as to the results of the checklist assessment.

References:

IEC 60880:2006, *Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions*

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Software Quality Assurance: From Theory to Implementation. Daniel Galin, Pearson Education, 2004, ISBN 0201709457, 9780201709452

IEC 61346 (all parts), *Industrial systems, installations and equipment and industrial products – Structuring principles and reference designation*

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824

B.2.6 Inspection of the specification

NOTE This technique/measure is referenced in Tables B.1 and B.6 of IEC 61508-2.

Aim: To avoid incompleteness and contradiction in the specification.

Description: Inspection is a general technique in which various qualities of a specification document are assessed by an independent team. The inspection team puts questions to the creator, who must answer them satisfactorily. The examination should (if possible) be carried out by a team that was not involved in the creation of the specification. The required degree of independence is determined by the safety integrity levels demanded of the system. The independent inspectors should be able to reconstruct the operational function of the system in an indisputable manner without referring to any further specifications. They must also check that all relevant safety and technical aspects in the operational and organisational measures are covered. This procedure has proved itself to be very effective in practice.

References:

IEC 61160:2005, *Design review*

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Software Quality Assurance: From Theory to Implementation. D. Galin, Pearson Education, 2004, ISBN 0201709457, 9780201709452

B.3 E/E/PE system design and development

Global objective: To produce a stable design of the safety-related system in conformance with the specification.

B.3.1 Observance of guidelines and standards

NOTE This technique/measure is referenced in Table B.2 of IEC 61508-2.

Aim: To observe application sector standards (not specified in this standard).

Description: Guidelines should be complied with during the design of the safety-related system. These guidelines should firstly lead to safety-related systems which are practically

free from failures, and secondly facilitate the subsequent safety validation. They can be universally valid, specific to a project, or specific only to a single phase.

References:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.3.2 Structured design

NOTE This technique/measure is referenced in Tables B.2 and B.6 of IEC 61508-2.

Aim: To reduce complexity by creating a hierarchical structure of partial requirements. To avoid interface failures between the requirements. To simplify verification.

Description: When designing the hardware, specific criteria or methods should be used. For example, the following might be required:

– a hierarchically structured circuit design;

– use of manufactured and tested circuit parts.

Similarly, when designing the software, the use of structure charts enables an unambiguous structure of the software modules to be created. This structure shows how the modules relate to each other, the precise data which passes between modules, and the precise controls that exist between modules.

References:

IEC 61346 (all parts), *Industrial systems, installations and equipment and industrial products – Structuring principles and reference designation*

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Software Design. D. Budgen, Pearson Education, 2003, ISBN 0201722194, 9780201722192

An Overview of JSD, J. R. Cameron, IEEE Trans SE-12 No. 2, February 1986

Structured Development for Real-Time Systems (3 Volumes). P. T. Yourdon, P. T. Yourdon Press, 1985

Structured Development for Real-Time Systems (3 Volumes). P. T. Ward, S. J. Mellor, Yourdon Press, 1985

Applications and Extensions of SADT. D. T. Ross, Computer, 25-34, April 1985

Essential Systems Analysis. St. M. McMenamin, F. Palmer, Yourdon Inc, 1984

Structured Analysis (SA): A language for communicating ideas. D. T. Ross, IEEE Trans. Software Eng, Vol. SE-3 (1), 16-34

B.3.3 Use of well-tried components

NOTE This technique/measure is referenced in Tables B.2 and B.6 of IEC 61508-2.

Aim: To reduce the risk of numerous first time and undetected faults by the use of components with specific characteristics.

Description: The selection of well-tried components is carried out by the manufacturer, with regard to safety according to the reliability of the elements (for example the use of operationally tested physical units to meet high safety requirements, or the storing of safety-

related programs in safe memories only). The safety of memories can refer to unauthorised access as well as environmental influences (electromagnetic compatibility, radiation, etc) and the response of the elements in the event of a failure occurring.

References:

IEC 61163-1:2006, *Reliability stress screening – Part 1: Repairable assemblies manufactured in lots*

B.3.4 Modularisation

NOTE This technique/measure is referenced in Tables B.2 and B.6 of IEC 61508-2.

Aim: To reduce complexity and avoid failures, related to interfacing between subsystems.

Description: Every subsystem, at all levels of the design, is clearly defined and is of restricted size (only a few functions). The interfaces between subsystems are kept as simple as possible and the cross-section (i.e. shared data, exchange of information) is minimised. The complexity of individual subsystems is also restricted.

References:

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Software Reliability – Principles and Practices. G. J. Myers, Wiley-Interscience, New York, 1976, ISBN-10: 0471627658, ISBN-13: 978-0471627654

B.3.5 Computer-aided design tools

NOTE This technique/measure is referenced in Tables B.2 and B.6 of IEC 61508-2 and in Tables A.4 and C.4 of IEC 61508-3.

Aim: To carry out the design procedure more systematically. To include appropriate automatic construction elements which are already available and tested.

Description: Computer-aided design tools (CAD) should be used during the design of both hardware and software when available and justified by the complexity of the system. The correctness of such tools should be demonstrated by specific testing, by an extensive history of satisfactory use, or by independent verification of their output for the particular safety-related system that is being designed.

Support tools should be selected for their degree of integration. In this context, tools are integrated if they work co-operatively such that 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.

References:

Overview of Technology Computer-Aided Design Tools and Applications in Technology Development, *Manufacturing and Design*. W. Fichtner, Journal of Computational and Theoretical Nanoscience, Volume 5, Number 6, June 2008, pp. 1089-1105(17)

The Electromagnetic Data Exchange: Much more than a Common Data Format. P.E. Frandsen et al. In *Proceeding of the 2nd European Conference on Antennas and Propagation*. The Institution of Engineering and Technology (IET), 2007, ISBN 978-0-86341-842-6

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

B.3.6 Simulation

NOTE This technique/measure is referenced in Tables B.2, B.5 and B.6 of IEC 61508-2.

Aim: To carry out a systematic and complete inspection of an electrical/electronic circuit, of both the functional performance and the correct dimensioning of the components.

Description: The function of the safety-related system circuit is simulated on a computer via a software behavioural model. Individual components of the circuit each have their own simulated behaviour, and the response of the circuit in which they are connected is examined by looking at the marginal data of each component.

B.3.7 Inspection (reviews and analysis)

NOTE This technique/measure is referenced in Tables B.2 and B.6 of IEC 61508-2.

Aim: To reveal discrepancies between the specification and implementation.

Description: Specified functions of the safety-related system are examined and evaluated to ensure that the safety-related system conforms to the requirements given in the specification. Any points of doubt concerning the implementation and use of the product are documented so they may be resolved. In contrast to a walk-through, the author is passive and the inspector is active during the inspection procedure.

References:

IEC 61160:2005, *Design Review*

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

ANSI/IEE 1028:1997, *IEEE Standard for software reviews*

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.3.8 Walk-through

NOTE This technique/measure is referenced in Tables B.2 and B.6 of IEC 61508-2.

Aim: To reveal discrepancies between the specification and implementation.

Description: Specified functions of the safety-related system draft are examined and evaluated to ensure that the safety-related system complies with the requirements given in the specification. Doubts and potential weak points concerning the realisation and use of the product are documented so that they may be resolved. In contrast to an inspection, the author is active and the inspector is passive during the walk-through.

References:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

ANSI/IEEE 1028:1997, *IEEE Standard for software reviews*

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

Methodisches Testen von Programmen. G. J. Myers, Oldenbourg Verlag, München, Wien, 1987

B.4 E/E/PE system operation and maintenance procedures

Global objective: To develop procedures which help to avoid failures during the operation and maintenance of the safety-related system.

B.4.1 Operation and maintenance instructions

NOTE This technique/measure is referenced in Table B.4 of IEC 61508-2.

Aim: To avoid mistakes during operation and maintenance of the safety-related system.

Description: User instructions contain essential information on how to use and how to maintain the safety-related system. In special cases, these instructions will also include examples on how to install the safety-related system in general. All instructions must be easily understood. Figures and schematics should be used to describe complex procedures and dependencies.

Reference: *Guidelines for Safe Automation of Chemical Processes*. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.4.2 User friendliness

NOTE This technique/measure is referenced in Table B.4 of IEC 61508-2.

Aim: To reduce complexity during operation of the safety-related system.

Description: The correct operation of the safety-related system may depend to some degree on human operation. By considering the relevant system design and the design of the workplace, the safety-related system developer must ensure that

- the need for human intervention is restricted to an absolute minimum;
- the necessary intervention is as simple as possible;
- the potential for harm from operator error is minimised;
- the intervention facilities and indication facilities are designed according to ergonomic requirements;
- the operator facilities are simple, well labelled and intuitive to use;
- the operator is not overstrained, even in extreme situations;
- training on intervention procedures and facilities is adapted to the level of knowledge and motivation of the trainee user.

B.4.3 Maintenance friendliness

NOTE This technique/measure is referenced in Table B.4 of IEC 61508-2.

Aim: To simplify maintenance procedures of the safety-related system and to design the necessary means for effective diagnosis and repair.

Description: Preventive maintenance and repair is often carried out under difficult circumstances and under pressure from deadlines. Therefore, the safety-related system developer should ensure that

- safety-related maintenance measures are necessary as seldom as possible or even, ideally, not necessary at all;
- sufficient, sensible and easy-to-handle diagnosis tools are included for those repairs that are unavoidable – tools should include all necessary interfaces;
- if separate diagnosis tools have to be developed or obtained, then these should be available on time.

B.4.4 Limited operation possibilities

NOTE This technique/measure is referenced in Tables B.4 and B.6 of IEC 61508-2.

Aim: To reduce the operation possibilities for the normal user.

Description: This approach reduces the operation possibilities by

- limiting the operation within special operating modes, for example by key switches;
- limiting the number of operating elements;
- limiting the number of generally possible operating modes.

Reference:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.4.5 Operation only by skilled operators

NOTE This technique/measure is referenced in Tables B.4 and B.6 of IEC 61508-2.

Aim: To avoid operating failures caused by misuse.

Description: The safety-related system operator is trained to a level which is appropriate to the complexity and safety integrity level of the safety-related system. Training includes studying the background of the production process and knowing the relationship between the safety-related system and the EUC.

Reference:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.4.6 Protection against operator mistakes

NOTE This technique/measure is referenced in Tables B.4 and B.6 of IEC 61508-2.

Aim: To protect the system against all classes of operator mistakes.

Description: Wrong inputs (value, time, etc) are detected via plausibility checks or monitoring of the EUC. To integrate these facilities into the design, it is necessary to state at a very early stage which inputs are possible and which are permissible.

B.4.7 (Not used)

B.4.8 Modification protection

NOTE This technique/measure is referenced in Tables A.17 and A.18 of IEC 61508-2.

Aim: To protect the safety-related system against hardware modifications by technical means.

Description: Modifications or manipulations are detected automatically, for example by plausibility checks for the sensor signals, detection by the technical process and by automatic start-up tests. If a modification is detected, then emergency action is taken.

B.4.9 Input acknowledgement

NOTE This technique/measure is referenced in Tables A.17 and A.18 of IEC 61508-2.

Aim: A mistake during operation is detected by the operator himself before activating the EUC.

Description: An input to the EUC via the safety-related system is echoed to the operator before being sent to the EUC so that the operator has the possibility to detect and correct a mistake. As well as abnormal, unprovoked personnel action, the system design should consider top/bottom speed limits and direction of human reaction. This would avoid, for example, the operator pressing keys faster than expected, causing the system to read a double keystroke as a single one, or a key to be pressed twice because the system (display) was too slow to react to the first instance. The same key stroke should not be valid more than once in succession for critical data entry; pressing the "enter" or "yes" key unlimited times must not lead to an unsafe action of the system.

Time-out procedures should be included with multiple choice questions (yes/no, etc.) to cater for when the operator may not make up his mind and leave the system waiting.

Ability to reboot a safety-related PES makes the system vulnerable unless both software/hardware are designed with such occasions in mind.

B.5 E/E/PE system integration

Global objective: To avoid failures during the integration phase and to reveal any failures that are made during this and previous phases.

B.5.1 Functional testing

NOTE This technique/measure is referenced in Tables B.3 and B.5 of IEC 61508-2 and in Tables A.5, A.6, A.7, C.5, C.6 and C.7 of IEC 61508-3.

Aim: To reveal failures during the specification and design phases. To avoid failures during implementation and the integration of software and hardware.

Description: During the functional tests, reviews are carried out to see whether the specified characteristics of the system have been achieved. The system is given input data which adequately characterises the normally expected operation. The outputs are observed and their response is compared with that given by the specification. Deviations from the specification and indications of an incomplete specification are documented.

Functional testing of electronic components designed for a multi-channel architecture usually involves the manufactured components being tested with pre-validated partner components. In addition to this, it is recommended that the manufactured components be tested in combination with other partner components of the same batch, in order to reveal common mode faults which would otherwise have remained masked.

Also, the working capacity of the system has to be sufficient, see guidance in C.5.20.

References:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Practical Software Testing: A Process-oriented Approach. I. Burnstein, Springer, 2003, ISBN 0387951318, 9780387951317

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.5.2 Black-box testing

NOTE This technique/measure is referenced in Tables B.3, B.5 and B.6 of IEC 61508-2 and in Tables A.5, A.6, A.7, C.5, C.6 and C.7 of IEC 61508-3.

Aim: To check the dynamic behaviour under real functional conditions. To reveal failures to meet functional specification, and to assess utility and robustness.

Description: The functions of a system or program are executed in a specified environment with specified test data which is derived systematically from the specification according to established criteria. This exposes the behaviour of the system and permits a comparison with the specification. No knowledge of the internal structure of the system is used to guide the testing. The aim is to determine whether the functional unit carries out correctly all the functions required by the specification. The technique of forming equivalence classes is an example of the criteria for blackbox test data. The input data space is subdivided into specific input value ranges (equivalence classes) with the aid of the specification. Test cases are then formed from the

– data from permissible ranges;
– data from inadmissible ranges;
– data from the range limits;
– extreme values;
– and combinations of the above classes.

Other criteria can be effective in order to select test cases in the various test activities (module test, integration test and system test). For example, the criterion "extreme operational conditions" is relied upon for the system test within the framework of a validation.

References:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844

Essentials of Software Engineering. Frank F. Tsui, Orlando Karam. Jones & Bartlett, 2006. ISBN 076373537X, 9780763735371

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Systematic Software Testing. Rick D. Craig, Stefan P. Jaskiel. Artech House, 2002. ISBN 1580535089, 9781580535083

B.5.3 Statistical testing

NOTE This technique/measure is referenced in Tables B.3, B.5 and B.6 of IEC 61508-2.

Aim: To check the dynamic behaviour of the safety-related system and to assess its utility and robustness.

Description: This approach tests a system or program with input data selected according to the expected statistical distribution of the real operating inputs – the operational profile.

References:

A discussion of statistical testing on a safety-related application. S Kuball, J H R May, Proc. IMechE Vol. 221 Part O: J. Risk and Reliability, Institution of Mechanical Engineers, 2007

Practical Reliability Engineering. P. O'Connor, D. Newton, R. Bromley, John Wiley and Sons, 2002, ISBN 0470844639, 9780470844632

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

B.5.4 Field experience

NOTE 1 This technique/measure is referenced in Tables B.3, B.5 and B.6 of IEC 61508-2.

NOTE 2 See also C.2.10 for a similar measure and Annex D for a statistical approach, both in the context of software.

Aim: To use field experience from different applications as one of the measures to avoid faults either during E/E/PE system integration and/or during E/E/PE system safety validation.

Description: Use of components or subsystems, which have been shown by experience to have no, or only unimportant, faults when used, essentially unchanged, over a sufficient period of time in numerous different applications. Particularly for complex components with a multitude of possible functions (for example operating system, integrated circuits), the developer shall pay attention to which functions have actually been tested by the field experience. For example, consider self-test routines for fault detection: if no break-down of the hardware occurs within the operating period, the routines cannot be said to have been tested, since they have never performed their fault detection function.

For field experience to apply, the following requirements must have been fulfilled:

– unchanged specification;
– 10 systems in different applications;
– 10^5 operating hours and at least one year of service history.

NOTE 3 Sector standards may specify different numbers.

The field experience is demonstrated through documentation of the vendor and/or operating company. This documentation must contain at least

– the exact designation of the system and its component, including version control for hardware;
– the users and time of application;
– the operating hours;
– the procedures for the selection of the systems and applications procured to the proof;

– the procedures for fault detection and fault registration as well as fault removal.

References:

IEC 60300-3-2:2004, *Dependability management – Part 3-2: Application guide – Collection of dependability data from the field*

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.6 E/E/PE system safety validation

Global objective: To prove that the E/E/PE safety-related system conforms to the E/E/PE system safety requirements specification and E/E/PE system design requirements specification.

B.6.1 Functional testing under environmental conditions

NOTE This technique/measure is referenced in Table B.5 of IEC 61508-2.

Aim: To assess whether the safety-related system is protected against typical environmental influences.

Description: The system is put under various environmental conditions (for example according to the standards in the IEC 60068 series or the IEC 61000 series), during which the safety functions are assessed for their reliability (and compatibility with the standards mentioned).

References:

IEC 60068-1:1988, *Environmental testing – Part 1: General and guidance*
Amendment 1(1992)

IEC 61000-4-1:2006, *Electromagnetic compatibility (EMC) – Part 4-1: Testing and measurement techniques – Overview of IEC 61000-4 series*

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.2 Interference surge immunity testing

NOTE This technique/measure is referenced in Tables B.5 and B.6 of IEC 61508-2.

Aim: To check the capacity of the safety-related system to handle peak surges.

Description: The system is loaded with a typical application program, and all the peripheral lines (all digital, analogue and serial interfaces as well as the bus connections and power supply, etc.) are subjected to standard noise signals. In order to obtain a quantitative statement, it is sensible to approach the surge limit carefully. The chosen class of noise is not complied with if the function fails.

References:

IEC 61000-4-5:2005, *Electromagnetic compatibility (EMC) – Part 4-5: Testing and measurement techniques – Surge immunity test*

C37.90.1-2002, *IEEE Standard for Surge Withstand Capability (SWC) Tests for Relays and Relay Systems Associated with Electric Power Apparatus*

B.6.3 (Not used)

B.6.4 Static analysis

NOTE This technique/measure is referenced in Tables B.5 and B.6 of IEC 61508-2 and in Tables A.9, B.8, C.9 and C.18 of IEC 61508-3.

Aim: To avoid systematic faults that can lead to breakdowns in the system under test, either early or after many years of operation.

Description: This systematic and possibly computer-aided approach inspects specific static characteristics of the prototype system to ensure completeness, consistency, lack of ambiguity regarding the requirement in question (for example construction guidelines, system specifications, and an appliance data sheet). A static analysis is reproducible. It is applied to a prototype which has reached a well-defined stage of completion. Some examples of static analysis, for hardware and software, are

– consistency analysis of the data flow (such as testing if a data object is interpreted everywhere as the same value);
– control flow analysis (such as path determination, determination of non-accessible code);
– interface analysis (such as investigation of variable transfer between various software modules);
– dataflow analysis to detect suspicious sequences of creating, referencing and deleting variables;
– testing adherence to specific guidelines (for example creepage distances and clearances, assembly distance, physical unit arrangement, mechanically sensitive physical units, exclusive use of the physical units which were introduced).

References:

Static Analysis and Software Assurance. D. Wagner, Lecture Notes in Computer Science, Volume 2126/2001, Springer, 2001, ISBN 978-3-540-42314-0

An Industrial Perspective on Static Analysis. B A Wichmann, A A Canning, D L Clutterbuck, L A Winsborrow, N J Ward and D W R Marsh. Software Engineering Journal., 69 – 75, March 1995

Dependability of Critical Computer Systems 3. P. G. Bishop et al,. Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.5 Dynamic analysis and testing

NOTE This technique/measure is referenced in Tables B.5 and B.6 of IEC 61508-2 and in Tables A.5, A.9, B.2, C.5, C.9 and C.12 of IEC 61508-3.

Aim: To detect specification failures by inspecting the dynamic behaviour of a prototype at an advanced state of completion.

Description: The dynamic analysis of a safety-related system is carried out by subjecting a near-operational prototype of the safety-related system to input data which is typical of the intended operating environment. The analysis is satisfactory if the observed behaviour of the safety-related system conforms to the required behaviour. Any failure of the safety-related system must be corrected and the new operational version must then be reanalysed.

References:

The Concept of Dynamic Analysis. T. Ball, ESEC/FSE ’99, Lecture Notes in Computer Science, Springer, 1999, ISBN 978-3-540-66538-0

Dependability of Critical Computer Systems 3. P. G. Bishop et al,. Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.6 Failure analysis

NOTE This technique/measure is referenced in Tables B.5 and B.6 of IEC 61508-2.

B.6.6.1 Failure modes and effects analysis (FMEA)

Aim: To analyse a system design, by examining systematically all possible sources of failure of a system's components and determining the effects of these failures on the behaviour and safety of the system.

Description: The analysis usually takes place through a meeting of engineers. Each component of a system is analysed in turn to give a set of failure modes for the component, their causes and effects (locally and at overall system level), detection procedures and recommendations. If the recommendations are acted upon, they are documented as remedial action taken.

References:

IEC 60812:2006, *Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824

Reliability Technology. A. E. Green, A. J. Bourne, Wiley-Interscience, 1972, ISBN 0471324809

B.6.6.2 Cause consequence diagrams

NOTE This technique/measure is referenced in Tables B.3, B.4, C.13 and C.14 of IEC 61508-3.

Aim: To analyse and model, in a compact diagrammatic form, the sequence of events that can develop in a system as a consequence of combinations of basic events.

Description: The technique can be regarded as a combination of fault tree and event tree analysis. It starts from a critical (initiating) event and the consequence graph is traced forwards by using YES/NO gates describing success and failure of some operations. This allows building event sequences leading either to an accident or to a mastered situation. Then cause graphs (i.e. fault trees) are built for each failure. Then starting from an accidental situation and going in the backward direction gives a global fault tree with this accidental situation as top event. In the forward direction the possible consequences arising from an event are determined. The graph can contain vertex symbols which describe the conditions for propagation along different branches from the vertex. Time delays can also be included. These conditions can also be described with fault trees. The lines of propagation can be combined with logical symbols, to make the diagram more compact. A set of standard symbols is defined for use in cause consequence diagrams. The diagrams can be used for generating fault trees and to compute the probability of occurrence of certain critical consequences. It can also be used to produce event trees.

References:

IEC 62502, *Analysis techniques for dependability – Event tree analysis (ETA)*[1]

The Cause Consequence Diagram Method as a Basis for Quantitative Accident Analysis. B. S. Nielsen, Danish Atomic Energy Commission, Riso-M-1374, 1971

[1] Under consideration.

B.6.6.3 Event tree analysis (ETA)

NOTE This technique/measure is referenced in Tables B.4 and C.14 of IEC 61508-3.

Aim: To model, in a diagrammatic form, the sequence of events that can develop in a system after an initiating event, and thereby indicate how serious consequences can occur. An event tree is difficult to build from scratch and using consequence diagram is helpful.

Description: On the top of the diagram is written the sequence conditions that are relevant in the progression of events that follow the initiating event. Starting under the initiating event, which is the target of the analysis, a line is drawn to the first condition in the sequence. There the diagram branches off into "yes" and "no" branches, describing how future events depend on the condition. For each of these branches, one continues to the next condition in a similar way. Not all conditions are, however, relevant for all branches. One continues to the end of the sequence, and each branch of the tree constructed in this way represents a possible consequence. Provided the conditions in the sequences are independent, the event tree can be used to compute the probability of the various consequences, based on the probability and number of conditions in the sequence. As conditions are rarely fully independent, such calculation shall be considered cautiously and performed by skilled analysts.

References:

IEC 62502, *Analysis techniques for dependability – Event tree analysis (ETA)*[2]

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824

B.6.6.4 Failure modes, effects and criticality analysis (FMECA)

NOTE Failure analysis is referenced in Tables A.10, B.4, C.10 and C.14 of IEC 61508-3.

Aim: To rank the criticality of components which could result in injury, damage or system degradation through single-point failures, in order to determine which components might need special attention and necessary control measures during design or operation.

Description: This method is comparable to FMEA, but there are one or more columns for indicating the criticality, which can be ranked in many ways. The most laborious method is described by the Society for Automotive Engineers (SAE) in ARP 926. In this procedure, the criticality number for any component is indicated by the number of failures of a specific type expected during each million operations occurring in a critical mode. The criticality number is a function of nine parameters, most of which have to be measured. A very simple method for criticality determination is to multiply the probability of component failure by the damage that could be generated; this method is similar to simple risk factor assessment.

References:

IEC 60812:2006, *Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*

Software criticality analysis of COTS/SOUP. P.Bishop, T.Clement, S.Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003

Software FMEA techniques. P.L.Goddard. In Proc Annual 2000 Reliability and Maintainability Symposium, IEEE, 2000, ISBN: 0-7803-5848-1

[2] Under consideration.

B.6.6.5 Fault tree analysis (FTA)

NOTE This technique/measure is referenced in Tables B.4 and C.14 of IEC 61508-3.

Aim: To aid in the analysis of events, or combinations of events, that will lead to a hazard or serious consequence and to perform the probability calculation of the top event.

Description: Starting at an event which would be the immediate cause of a hazard or serious consequence (the "top event"), analysis is carried out in order to identify the causes of this event. This is done in several steps through the use of logical operators (and, or, etc). Intermediate causes are analysed in the same way, and so on, back to basic events where analysis stops.

The method is graphical, and a set of standardised symbols are used to draw the fault tree. At the end of the analysis, the fault tree represents the logical function linking the basic events (generally components failures) to the top event (the overall system failure).The technique is mainly intended for the analysis of hardware systems, but there have also been attempts to apply this approach to software failure analysis. This technique can be used qualitatively for failure analysis (identification failure scenarios: minimal cut sets or prime implicants), semi-quantitatively (by ranking scenarios according to their probabilities) and quantitatively for probabilistic calculations of the top event (see C.6).

References:

IEC 61025:2006, *Fault tree analysis (FTA)*

From safety analysis to software requirements. K.M. Hansen, A.P. Ravn, A.P, V Stavridou. IEEE Trans Software Engineering, Volume 24, Issue 7, Jul 1998

B.6.6.6 Markov models

NOTE See B.1 of IEC 61508-6 for the use of this technique against reliability block diagrams, in the context of analysing hardware safety integrity.

Aim: To model the behaviour of the system by a state transition graph and to evaluate probabilistic system parameters (e.g, un-reliability, un-availability, *MTTF*, *MUT*, *MDT*, etc.) of a system.

Description: It is a finite state automaton (see B.2.3.2) represented by a directed graph. The nodes (circles) represent the states and the edges (arrows) between nodes represent the transitions (failure, repairs, etc.) occurring between the states. Edges are weighted with the corresponding failure rates or repair rates. The fundamental property of homogeneous Markov processes is that the future depends only of the present: a change of state, *N*, to a subsequent state, *N*+1, is independent of the previous state, *N*-1. This implies that all the probabilistic laws of the models are exponential.

The failure events, states and rates can be detailed in such a way that a precise description of the system is obtained, for example detected or undetected failures, manifestation of a larger failure, etc. Proof test intervals may also be modelled properly by using the so-called multi-phase Markov processes where the probabilities of the states at the end of one phase (e.g. just before a proof test) can be used to calculate the initial conditions for the next phase (e.g. the probabilities of the various states after a proof test has been performed).

The Markov technique is suitable for modelling multiple systems in which the level of redundancy varies with time due to component failure and repair. Other classical methods, for example, FMEA and FTA, cannot readily be adapted to modelling the effects of failures throughout the lifecycle of the system since no simple combinatorial formulae exist for calculating the corresponding probabilities.

In the simplest cases, the formulae which describe the probabilities of the system are readily available in the literature or can be calculated manually and some methods of simplification (i.e. reducing the number of states) also exist to handle more complex cases.

Nevertheless, mathematically speaking, a homogeneous Markov graph is only a simple and common set of linear differential equations with constant coefficients. This has been analysed for a long time and powerful algorithms have been developed and are available to handle them. Therefore when the size of the model increases it is very efficient to use the above algorithms which are implemented in various computer software packages.

It has to be noted that the size of the graph increases exponentially with the number of components: this is the so-called combinatorial explosion. Therefore this technique is usable without approximations only for small systems.

When non-exponential laws have to be handled -semi-Markov models- then Monte Carlo simulation (see B.6.6.8) should be used.

References:

IEC 61165:2006, *Application of Markov techniques*

The Theory of Stochastic Processes. R. E. Cox and H. D. Miller, Methuen and Co. Ltd., London, UK, 1963

Finite MARKOV Chains. J. G. Kemeny and J. L. Snell. D. Van Nostrand Company Inc, Princeton, 1959

The Theory and Practice of Reliable System Design. D. P. Siewiorek and R. S. Swarz, Digital Press, 1982

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.6.7 Reliability block diagrams (RBD)

NOTE 1 This technique/measure is used in Annex B of IEC 61508-6.

NOTE 2 See also C.6.4 “Reliability block diagrams”.

Aim: To model, in a diagrammatic form, the set of events that must take place and conditions which must be fulfilled for a successful operation of a system or a task. It is more a method of representation than a method of analysis.

Description: The target of the analysis is represented as a success path consisting of blocks, lines and logical junctions. A success path starts from one side of the diagram and continues via the blocks and junctions to the other side of the diagram. A block represents a condition or an event, and the path can pass it if the condition is true or the event has taken place. If the path comes to a junction, it continues if the logic of the junction is fulfilled. If it reaches a vertex, it may continue along all outgoing lines. If there exists at least one success path through the diagram, the target of the analysis is operating correctly.

A RBD is a structural representation of the modelled system. It is a kind of electrical circuit: when the current find a path from the input to the output, that means that the modelled system is working properly, when the circuit is cut that means that the modelled system is failed. This lead to the concept of minimal cut sets which represent the combinations of failures (i.e. places where the RBD is "cut") leading to the failure of the modelled system.

Mathematically a RBD is similar to a fault tree. It represents the logical function linking the states of the individual components (failed or working) to the state of the whole system (failed or working). Therefore the calculations are similar as those described for fault trees.

References:

IEC 61078:2006, *Analysis techniques for dependability – Reliability block diagram and boolean methods*

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.6.8 Monte-Carlo simulation

NOTE This technique/measure is referenced in Table B.4 of IEC 61508-3 and used in IEC 61508-6 Annex B.

Aim: To simulate real world phenomena by generating random numbers when analytical methods are not practicable.

Description: Monte-Carlo simulations are used to solve two classes of problems:

– probabilistic, where random numbers are used to generate stochastic phenomena; and

– deterministic, which are mathematically translated into an equivalent probabilistic problem (e.g. integral calculations).

The principle of Monte-Carlo simulation is to use random numbers to animate a behavioural functional and dysfunctional model of the system under study. Such behavioural models are provided by state transition models (Markov graph, Petri nets, formal languages, etc.). The Monte-Carlo simulation is run to produce a large statistical sample from which statistical results are obtained.

When using Monte-Carlo simulations care must be taken to ensure that the biases, tolerances or noise have reasonable values. This shall be managed through the confidence interval which is easily obtained from the simulations. Contrary to analytical methods, Monte Carlo simulation is self approximating. Negligible events just not appear without need to identify them to simplify the model.

A general principle of Monte-Carlo simulations is to restate and reformulate the problem so that the results obtained are as accurate as possible rather than tackling the problem as initially stated.

In the context of this standard, Monte Carlo simulation may be used for SIL calculations and to take into consideration the reliability data uncertainties. With present time computers, SIL4 calculations are easily achieved.

References:

Monte Carlo Methods. J. M. Hammersley, D. C. Handscomb, Chapman & Hall, 1979

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.6.9 Fault tree models

NOTE 1 See IEC 61508-6 for the use of this technique in the context of analysing hardware safety integrity.

NOTE 2 Fault trees have already been described above as safety validation means (see B.6.6.5). They are also widely used for failure analysis and probabilistic calculations.

Aim: To build by a systematic topdown graphical (effect-cause) approach, the logical function linking the basic events (failure modes) to the top event (unwanted event).

Description: This is both a method of analysis helping the analyst to develop the model step by step and a mathematical model for probabilistic calculations. It allows performing:

– qualitative analysis by identifying and sorting failure scenarios (minimal cut sets or prime implicants),

– semi quantitative analysis by ranking scenarios according to their probabilities of occurrence,

– quantitative analysis by calculating the probability of the top event.

Like RBD (Reliability Block Diagrams), a fault tree represents the logical (boolean) function linking the states of the individual components (failed or working) to the state of the whole system (failed or working). Therefore when the components are independent, probabilistic calculations may be performed just by applying the basic properties of probabilities applied on logical function. This is not so easy because this is a static model which basically works only with genuine (i.e. constant) probabilities). Time dependent probabilities shall be handled cautiously. For example the PFD_{avg} of safety systems comprising periodically proof tested components cannot be calculated straightforwardly and this is even more difficult for *PFH* of safety systems working in continuous mode. Therefore only reliability engineers with a sound understanding of the underlying mathematics should perform un-availability/*PFD* and un-reliability/*PFH* calculations with this method.

Calculations may be done by hand for very simple fault trees but a lot of algorithms have been developed and implemented to handle complex logical equations over the last 50 past years. The state of the art at the present time is using BDD (Binary Decision Diagrams) which is a technique of compact encoding of the logical equation into a computer memory. It is, at the present time, the only method able to perform the probabilistic calculations without approximations on industrial size systems. It is also sufficiently fast to allow handling uncertainties by Monte Carlo simulation.

Reference:

IEC 61025:2006, *Fault tree analysis (FTA)*

B.6.6.10 Generalised Stochastic Petri net models (GSPN)

NOTE 1 See IEC 61508-6 for the use of this technique in the context of analysing hardware safety integrity.

NOTE 2 Petri nets have already been mentioned as semi-formal method (see B.2.3.3). They can also efficiently used in the context of hardware safety integrity.

Aim: To graphically build a functionnal and dysfunctional model behaving as close as possible as the actual modelled system in order to provide an efficient support for Monte carlo simulation.

Description: This is an asynchronous finite state automata as described in B.2.3.3, except that the good property tracked when performing semi-formal validation are no longer existing when modelling the dysfunctionnal behaviour of a safety system. The so-called places (pictured by circles) represent the potental states and the so-called transitions (pictured by rectangles) represents the events likely to occur. In addition of the marking of the places (see B.2.3.3) messages or predicates may be used to validate (enable) the transitions and the delay elapsing from the validation of a transition to its firing may be deterministic or stochastic. This is why those Petri Nets are called "generalised stochastic" Petri nets.

Petri nets constitute flexible behavioural models which prove to be very efficient as Monte Carlo simulation support (see B.6.6.8). Except the accuracy of the Monte Carlo simulation itself which, anyway, is always known, all the limitations of other methods (dependancies,

combinatory explosion, non-exponentail laws, etc.) are overcome. With present time computer this is no longer a problem even for SIL4 evaluations.

References:

IEC 62551, *Analysis techniques for dependability – Petri net modelling (CD1)*[3]

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.7 Worst-case analysis

NOTE This technique/measure is referenced in Tables B.5 and B.6 of IEC 61508-2.

Aim: To avoid systematic failures which arise from unfavourable combinations of the environmental conditions and the component tolerances.

Description: The operational capacity of the system and the component dimensioning is examined or calculated on a theoretical basis. The environmental conditions are changed to their highest permissible marginal values. The most essential responses of the system are inspected and compared with the specification.

B.6.8 Expanded functional testing

NOTE This technique/measure is referenced in Tables B.5 and B.6 of IEC 61508-2.

Aim: To reveal failures during the specification and design and development phases. To check the behaviour of the safety-related system in the event of rare or unspecified inputs.

Description: Expanded functional testing reviews the functional behaviour of the safety-related system in response to input conditions which are expected to occur only rarely (for example major failure), or which are outside the specification of the safety-related system (for example incorrect operation). For rare conditions, the observed behaviour of the safety-related system is compared with the specification. Where the response of the safety-related system is not specified, one should check that the plant safety is preserved by the observed response.

References:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Dependability of Critical Computer Systems 3. P. G. Bishop et al,. Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.9 Worst-case testing

NOTE This technique/measure is referenced in Tables B.5 and B.6 of IEC 61508-2.

Aim: To test the cases specified during worst-case analysis.

Description: The operational capacity of the system and the component dimensioning is tested under worst-case conditions. The environmental conditions are changed to their highest permissible marginal values. The most essential responses of the system are inspected and compared with the specification.

---

[3] Under consideration.

B.6.10 Fault insertion testing

NOTE This technique/measure is referenced in Tables B.5 and B.6 of IEC 61508-2.

Aim: To introduce or simulate faults in the system hardware and document the response.

Description: This is a qualitative method of assessing dependability. Preferably, detailed functional block, circuit and wiring diagrams are used in order to describe the location and type of fault and how it is introduced; for example: power can be cut from various modules; power, bus or address lines can be open/short-circuited; components or their ports can be opened or shorted; relays can fail to close or open, or do it at the wrong time, etc. Resulting system failures are classified, as in Tables 1 and 2 of IEC 60812, for example. In principle, single steady-state faults are introduced. However, in case a fault is not revealed by the built-in diagnostic tests or otherwise does not become evident, it can be left in the system and the effect of a second fault considered. The number of faults can easily increase to hundreds.

The work is done by a multidisciplinary team and the vendor of the system should be present and consulted. The mean operating time between failure for faults that have grave consequences should be calculated or estimated. If the calculated time is low, modifications should be made.

References:

IEC 60812:2006, *Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*

IEC 61069-5:1994, *Industrial-process measurement and control – Evaluation of system properties for the purpose of system assessment – Part 5: Assessment of system dependability*

Annex C
(informative)

Overview of techniques and measures for achieving software safety integrity (see IEC 61508-3)

C.1 General

The overview of techniques contained in this annex should not be regarded as either complete or exhaustive.

C.2 Requirements and detailed design

NOTE Relevant techniques and measures are found in B.2.

C.2.1 Structured diagrammatic methods

NOTE This technique/measure is referenced in Tables A.2 and A.4 of IEC 61508-3.

C.2.1.1 General

Aim: The main aim of structured methods is to promote the quality of software development by focusing attention on the early parts of the lifecycle. The methods aim to achieve this through both precise and intuitive procedures and notations (assisted by computers), to determine and document requirements and implementation features in a logical order and a structured manner.

Description: A range of structured methods exists. Some are designed for traditional data-processing and transaction processing functions, while others are more oriented to process control and real-time applications (which tend to be more safety critical). UML (see C.3.12) contains many examples of structured notations.

Structured methods are essentially "thought tools" for systematically perceiving and partitioning a problem or system. Their main features are the following:

– a logical order of thought, breaking a large problem into manageable stages;

– analysis and documentation of the total system, including the environment as well as the required system;

– decomposition of data and function in the required system;

– checklists, i.e. lists of the sort of things that need analysis;

– low intellectual overhead – simple, intuitive, pragmatic;

– often with a strong emphasis on developing a diagrammatic model of the intended system, and CASE tool support for the overall method.

The supporting notations for analysing and documenting problems and system entities (for example processes and data flows) tend to be precise, but notations for expressing the processing functions performed by these entities tend to be more informal. However, some methods do make partial use of (mathematically) formal notations (for example, regular expressions or finite state machines). Increased precision not only reduces the scope for misunderstanding, it provides scope for automatic processing.

Another benefit of structured notations is their visibility, enabling a specification or design to be checked intuitively by a user, against his powerful but unstated knowledge.

This overview describes several structured methods in more detail.

Reference:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Software Design. D. Budgen, Pearson Education, 2003, ISBN 0201722194, 9780201722192

C.2.1.2 Controlled Requirements Expression (CORE)

Aim: To ensure that all the requirements are determined and expressed.

Description: This approach is intended to bridge the gap between the customer/end user and the analyst. It is not mathematically rigorous but aids communication – CORE is designed for requirements expression rather than specification. The approach is structured, and the expression goes through various levels of refinement. The CORE method encourages a wider view of the problem, bringing in a knowledge of the environment in which the system will be used and the differing viewpoints of the various types of user. CORE includes guidelines and tactics for recognising departures from the "grand design". Departures can be corrected or explicitly identified and documented. Thus specifications may not be complete, but unresolved problems and high-risk areas are detected and have to be considered in the subsequent design.

Reference:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794

C.2.1.3 Jackson System Development (JSD)

Aim: A development method covering the development of software systems from requirements through to code, with special emphasis on real-time systems.

Description: JSD is a staged development procedure in which the developer models the real world behaviour upon which the system functions are to be based, determines the required functions and inserts them into the model, and transforms the resulting specification into one that is realisable in the target environment. It therefore covers the traditional phases of specification and design and development but takes a somewhat different view from the traditional methods in not being top-down.

Moreover, it places great emphasis on the early stage of discovering the entities in the real world that are the concern of the system being built and on modelling them and what can happen to them. Once this analysis of the "real world" has been done and a model created, the system's required functions are analysed to determine how they can fit into this real-world model. The resulting system model is augmented with structured descriptions of all the processes in the model and the whole is then transformed into programs that will operate in the target software and hardware environment.

References:

Systems Analysis and Design. D. Yeates, A. Wakefield. Pearson Education, 2003, ISBN 0273655361, 9780273655367

An Overview of JSD. J. R. Cameron. IEEE Transactions on Software Engineering, SE-12, No. 2, February 1986

C.2.1.4 Real-time Yourdon

Aim: The specification and design of real-time systems.

Description: The development scheme underlying this technique assumes a three-stage evolution of a system being developed. The first stage involves the building of an "essential model", one that describes the behaviour required by the system. The second involves the building of an implementation model which describes the structures and mechanisms that, when implemented, embody the required behaviour. The third stage involves the actual building of the system in hardware and software. The three stages correspond roughly to the traditional specification and design and development phases but lay greater emphasis on the fact that at each stage the developer is engaged in a modelling activity.

The essential model is in two parts:

– the environmental model, containing a description of the boundary between the system and its environment and a description of the external events to which the system must respond; and

– the behavioural model, which contains schemes describing the transformation carried out by the system in response to events and a description of the data the system must hold in order to respond.

The implementation model also divides into submodels, covering the allocation of individual processes to processors and the decomposition of the processes into software modules.

To capture these models, the technique combines a number of other well-known techniques: data-flow diagrams, transformation graphs, structured English, state transition diagrams and Petri nets. Additionally, the method contains techniques for simulating a proposed system design, either on paper or mechanically from the models that are drawn up.

References:

Real-time Systems Development. R. Williams. Butterworth-Heinemann, 2006, ISBN 0750664711, 9780750664714

Structured Development for Real-Time Systems (3 Volumes). P. T. Ward and S. J. Mellor. Yourdon Press, 1985

C.2.2 Data flow diagrams

NOTE This technique/measure is referenced in Tables B.5 and B.7 of IEC 61508-3.

Aim: To describe the data flow through a program in a diagrammatic form.

Description: Data flow diagrams document how data input is transformed to output, with each stage in the diagram representing a distinct transformation.

Data flow diagrams have three aspects:

– annotated arrows – represent data flow in and out of the transformation centres, with the annotations documenting what the data is;

– annotated bubbles – represent transformation centres, with the annotation documenting the transformation;

– operators (and, xor) – these operators are used to link the annotated arrows.

Each bubble in a data flow diagram can be considered as a stand-alone black box which, as soon as its inputs are available, transforms them to its outputs. One of the principal advantages is that they show transformations without making any assumptions about how these transformations are implemented. A pure data flow diagram does not include control

information or sequencing information, but this is catered for by real-time extensions to the notation, as in real-time Yourdon (see C.2.1.4).

The preparation of data flow diagrams is best approached by considering system inputs and working towards system outputs. Each bubble must represent a distinct transformation – its output should, in some way, be different from its input. There are no rules for determining the overall structure of the diagram and constructing a data flow diagram is one of the creative aspects of system design. Like all design, it is an iterative procedure with early attempts refined in stages to produce the final diagram.

References:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

ISO 5807:1985, *Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts*

ISO/IEC 8631:1989, *Information technology – Program constructs and conventions for their representation*

C.2.3 Structure diagrams

NOTE This technique/measure is referenced in Table B.5 of IEC 61508-3.

Aim: To show the structure of a program diagrammatically.

Description: Structure diagrams are a notation which complements data flow diagrams. They describe the programming system and a hierarchy of parts and display this graphically, as a tree. They document how elements of a data flow diagram can be implemented as a hierarchy of program units.

A structure chart shows relationships between program modules without including any information about the order of activation of these units. They are drawn using the following four symbols:

- a rectangle annotated with the name of the module;
- a line connecting these rectangles creating structure;
- a circled arrow (circle empty), annotated with the name of data passed to and from elements in the structure chart (normally, the circled arrow is drawn parallel to the line connecting the rectangles in the chart);
- a circled arrow (circle filled), annotated with the name of the control signal passing from one module to another in the structure chart, again the arrow is drawn parallel to the line connecting the two modules.

From any non-trivial data flow diagram, it is possible to derive a number of different structure charts.

Data flow diagrams depict the relationship between information and functions in the system. Structure charts depict the way elements of the system are implemented. Both techniques present valid, though different, views of the system.

References:

Software Design & Development. G. Lancaster. Pascal Press, 2001, ISBN 1741251753, 9781741251753

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

C.2.4 Formal methods

NOTE This technique/measure is referenced in Tables A.1, A.2, A.4 and B.5 of IEC 61508-3.

C.2.4.1 General

Aim: The development of software in a way that is based on mathematics. This includes formal design and formal coding techniques.

Description: Formal methods provide a means of developing a description of a system at some stage in its specification, design or implementation. The resulting description is in a strict notation that can be subjected to mathematical analysis to detect various classes of inconsistency or incorrectness. Moreover, the description can in some cases be analysed by machine with a rigour similar to the syntax checking of a source program by a compiler, or animated to display various aspects of the behaviour of the system described. Animation can give extra confidence that the system meets the real requirement as well as the formally specified requirement, because it improves human recognition of the specified behaviour.

A formal method will generally offer a notation (generally some form of discrete mathematics being used), a technique for deriving a description in that notation, and various forms of analysis for checking a description for different correctness properties.

Several formal methods are described in the following subclauses of this overview : CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z. Note that other techniques, such as finite state machines and Petri nets (see Annex B), may be considered as formal methods, depending on how strictly the techniques, as used, conform to a rigorous mathematical basis.

References:

Formal Specification: Techniques and Applications. N.Nissanke, Springer-Verlag Telos, 1999, ISBN 1852330023

The Practice of Formal Methods in Safety-Critical Systems. S. Liu, V. Stavridou, B. Dutertre, J. Systems Software 28, 77-87, Elsevier, 1995

Formal Methods: Use and Relevance for the Development of Safety-Critical Systems. L. M. Barroca, J. A. McDermid, The Computer Journal 35 (6), 579-599, 1992

How to Produce Correct Software – An Introduction to Formal Specification and Program Development by Transformations. E. A. Boiten et al, The Computer Journal 35 (6), 547-554, 1992

C.2.4.2 Calculus of Communicating Systems (CCS)

Aim: CCS is a means of describing and reasoning about the behaviour of systems of concurrent, communicating processes.

Description: CCS is a mathematical calculus concerned with the behaviour of systems. The system design is modelled as a network of independent processes operating sequentially or in parallel. Processes can communicate via ports (similar to CSP's channels), the communication only taking place when both processes are ready. Non-determinism can be modelled. Starting from a high-level abstract description of the entire system (known as a trace), it is possible to carry out a step-wise refinement of the system into a composition of communicating processes whose total behaviour is that required of the whole system. Equally, it is possible to work in a bottom-up fashion, combining processes and deducing the properties of the resulting system using inference rules related to the composition rules.

Reference:

Communication and Concurrency. R. Milner. Pearson Education, 1989, ISBN 9780131150072

C.2.4.3 Communicating Sequential Processes (CSP)

Aim: CSP is a technique for the specification of concurrent software systems, i.e. systems of communicating processes operating concurrently.

Description: CSP provides a language for the specification of systems of processes and proof for verifying that the implementation of processes satisfies their specifications (described as a trace, i.e. a permissible sequence of events).

A system is modelled as a network of independent processes, composed sequentially or in parallel. Each process is described in terms of all of its possible behaviours. Processes can communicate (synchronise or exchange data) via channels, the communication only taking place when both processes are ready. The relative timing of events can be modelled.

The theory behind CSP was directly incorporated into the architecture of the INMOS transputer, and the OCCAM language allows a CSP-specified system to be directly implemented on a network of transputers.

Reference:

Communicating Sequential Processes: The First 25 Years. A. Abdallah, C. Jones, J. Sanders (Eds.). Springer, 2004, ISBN 3540258132, 9783540258131

C.2.4.4 Higher Order Logic (HOL)

Aim: This is a formal language intended as a basis for hardware specification and verification.

Description: HOL refers to a particular logic notation and its machine support system, both of which were developed at the University of Cambridge computer laboratory. The logic notation is mostly taken from Church's simple theory of types and the machine support system is based upon the logic of computable functions (LCF) system.

Reference:

Higher-Order Computational Logic. J. Lloyd. In Computational Logic: Logic Programming and Beyond, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, ISBN 978-3-540-43959-2

C.2.4.5 LOTOS

Aim: LOTOS is a means for describing and reasoning about the behaviour of systems of concurrent, communicating processes.

Description: LOTOS (language for temporal ordering specification) is based on CCS with additional features from the related algebras CSP and CIRCAL (circuit calculus). It overcomes the weakness of CCS in the handling of data structures and value expressions by combining it with aspects of the abstract data type language ACT ONE. The process description aspect of LOTOS could, however, be used with other formalisms for the description of abstract data types.

References:

Model Checking for Software Architectures. R. Mateescu. In Software Architecture, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2004, ISBN 978-3-540-22000-8

ISO 8807:1989, *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*

C.2.4.6 OBJ

Aim: To provide a precise system specification with user feed-back and system validation prior to implementation.

Description: OBJ is an algebraic specification language. Users specify requirements in terms of algebraic equations. The behavioural, or constructive, aspects of the system are specified in terms of operations acting on abstract data types (ADT). An ADT is like an ADA package where the operator behaviour is visible whilst the implementation details are "hidden".

An OBJ specification, and subsequent step-wise implementation, is amenable to the same formal proof techniques as other formal approaches. Moreover, since the constructive aspects of the OBJ specification are machine-executable, it is straightforward to achieve system validation from the specification itself. Execution is essentially the evaluation of a function by equation substitution (rewriting) which continues until specific output value is obtained. This executability allows end-users of the envisaged system to gain a "view" of the eventual system at the system specification stage without the need to be familiar with the underlying formal specification techniques.

As with all other ADT techniques, OBJ is only applicable to sequential systems, or to sequential aspects of concurrent systems. OBJ has been used for the specification of both small- and large-scale industrial applications.

References:

Software Engineering with OBJ: Algebraic Specification in Action. J. Goguen, G. Malcolm. Springer, 2000, ISBN 0792377575, 9780792377573

C.2.4.7 Temporal logic

Aim: Direct expression of safety and operational requirements and formal demonstration that these properties are preserved in the subsequent development steps.

Description: Standard first-order predicate logic contains no concept of time. Temporal logic extends first-order logic by adding modal operators (for example "henceforth" and "eventually"). These operators can be used to qualify assertions about the system. For example, safety properties might be required to hold "henceforth", whilst other desired system states might be required to be attained "eventually" from some other initiating state. Temporal formulas are interpreted on sequences of states (behaviours). What constitutes a "state" depends on the chosen level of description. It can refer to the whole system, a system element or the computer program.

Quantified time intervals and constraints are not handled explicitly in temporal logic. Absolute timing has to be handled by creating additional time states as part of the state description.

Reference:

Mathematical Logic for Computer Science. M. Ben-Ari. Springer, 2001, ISBN 1852333197, 9781852333195

C.2.4.8 VDM, VDM++ – Vienna Development Method

Aim: The systematic specification and implementation of sequential (VDM) and concurrent real-time (VDM++) programs.

Description: VDM is a mathematically based specification technique and a technique for refining implementations in a way that allows proof of their correctness with respect to the specification.

The specification technique is model-based in that the system state is modelled in terms of set-theoretic structures on which are described invariants (predicates), and operations on that state are modelled by specifying their pre- and post-conditions in terms of the system state. Operations can be proved to preserve the system invariants.

The implementation of the specification is done by the reification of the system state in terms of data structures in the target language and by refinement of the operations in terms of the program in the target language. Reification and refinement steps give rise to proof obligations that establish their correctness. Whether or not these obligations are carried out is determined by the designer.

VDM is principally used in the specification stage but can be used in the design and implementation stages leading to source code. It can only be applied to sequentially structured programs or the sequential processes in concurrent systems.

The object-oriented and concurrent real-time extension of VDM, VDM++, is a formal specification language based on the ISO language VDM-SL and on the object-oriented language Smalltalk.

VDM++ provides a wide range of constructs such that a user can formally specify concurrent real-time systems in an object-oriented fashion. In VDM++ a complete formal specification consists of a collection of class specifications and optionally a workspace.

Real-time provisions of VDM++ are:

– temporal expressions are provided to denote both the current moment and the method invocation moment within a method body;

– a timed post expression can be added to a method to specify the upper (or lower) bounds of the execution time for correct implementations;

– time continuous variables have been introduced. With assumption and effect clauses one can specify relations (for example differential equations) between these functions of time. This feature has proven to be very useful in the specification of requirements of systems which operate in a time continuous environment. Refinement steps lead to discrete software solutions for these kinds of systems.

References:

ISO/IEC 13817-1:1996, *Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language*

Systematic Software Development using VDM. C. B. Jones. Prentice-Hall. 2nd Edition, 1990

Conformity Clause for VDM-SL, G. I. Parkin and B. A. Wichmann, Lecture Notes in Computer Science 670, FME'93 Industrial-Strength Formal Methods, First International Symposium of Formal Methods in Europe. Editors: J. C. P. Woodcock and P. G. Larsen. Springer Verlag, 501-520

C.2.4.9 Z

Aim: Z is a specification language notation for sequential systems and a design technique that allows the developer to proceed from a Z specification to executable algorithms in a way that allows proof of their correctness with respect to the specification.

Z is principally used in the specification stage but a method has been devised to go from specification into a design and an implementation. It is best suited to the development of data-oriented, sequential systems.

Description: Like VDM, the specification technique is model-based in that the system state is modelled in terms of set-theoretic structures on which are described invariants (predicates), and operations on that state are modelled by specifying their pre- and post-conditions in terms of the system state. Operations can be proved to preserve the system invariants thereby demonstrating their consistency. The formal part of a specification is divided into schemas which allow the structuring of specifications through refinement.

Typically, a Z specification is a mixture of formal Z and informal explanatory text in natural language. (Formal text on its own can be too terse for easy reading and often its purpose needs to be explained, while the informal natural language can easily become vague and imprecise.)

Unlike VDM, Z is a notation rather than a complete method. However, an associated method (called B) has been developed which can be used in conjunction with Z. The B method is based on the principle of step-wise refinement.

References:

Formal Specification using Z, 2nd Edition. D. Lightfoot. Palgrave Macmillan, 2000, ISBN 9780333763278

The B-Method. S. Schneider. Palgrave Macmillan, 2001, ISBN 9780333792841

C.2.5 Defensive programming

NOTE This technique/measure is referenced in Table A.4 of IEC 61508-3.

Aim: To produce programs which detect anomalous control flow, data flow or data values during their execution and react to these in a predetermined and acceptable manner.

Description: Many techniques can be used during programming to check for control or data anomalies. These can be applied systematically throughout the programming of a system to decrease the likelihood of erroneous data processing.

There are two overlapping areas of defensive techniques. Intrinsic error-safe software is designed to accommodate its own design shortcomings. These shortcomings may be due to mistakes in design or coding, or to erroneous requirements. The following lists some of the defensive techniques:

– variables should be range checked;

– where possible, values should be checked for plausibility;

– parameters to procedures should be type, dimension and range checked at procedure entry.

These first three recommendations help to ensure that the numbers manipulated by the program are reasonable, both in terms of the program function and physical significance of the variables.

Read-only and read-write parameters should be separated and their access checked. Functions should treat all parameters as read-only. Literal constants should not be write-accessible. This helps detect accidental overwriting or mistaken use of variables.

Fault tolerant software is designed to "expect" failures in its own environment or use outside nominal or expected conditions, and behave in a predefined manner. Techniques include the following.

– Input variables and intermediate variables with physical significance should be checked for plausibility.
– The effect of output variables should be checked, preferably by direct observation of associated system state changes.
– The software should check its configuration, including both the existence and accessibility of expected hardware and also that the software itself is complete – this is particularly important for maintaining integrity after maintenance procedures.

Some of the defensive programming techniques, such as control flow sequence checking, also cope with external failures.

References:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Dependability of Critical Computer Systems: Guidelines Produced by the European Workshop on Industrial Computer Systems, Technical Committee 7 (EWICS TC7, Systems Reliability, Safety, and Security). Elsevier Applied Science, 1989, ISBN 1851663819, 9781851663811

C.2.6 Design and coding standards

NOTE This technique/measure is referenced in Table A.4 of IEC 61508-3.

C.2.6.1 General

Aim: To facilitate verifiability, to encourage a team-centred, objective approach and to enforce a standard design method.

Description: The rules to be adhered to are agreed at the outset of the project between the participants. These rules comprise the design and development methods to be followed (for example JSP, Petri nets, etc.) and the related coding standards (see C.2.6.2).

These rules are made to allow for ease of development, verification, assessment and maintenance. Therefore they should take into account available tools, in particular analysers and reverse engineering tools.

References:

IEC 60880:2006, *Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions*

Verein Deutscher Ingenieure. Software-Zuverlassigkeit – Grundlagen, Konstruktive Mass-nahmen, Nachweisverfahren. VDI-Verlag, 1993, ISBN 3-18-401185-2

C.2.6.2 Coding standards

NOTE This technique/measure is referenced in Table B.1 of IEC 61508-3.

Aim: To reduce the likelihood of errors in the safety-related code and to facilitate its verification.

Description: The following principles indicate how safety-related coding rules (for any programming language) can assist in complying with the IEC 61508-3 normative requirements and in achieving the informative "desirable properties" (see Annex F). Account should be taken of available support tools.

IEC 61508-3 Requirements & Recommendations	*Coding Standards Suggestions*
Modular approach (Table A.2-7, Table A.4-4)	Software module size limit (Table B.9–1) and software complexity control (Table B.9–2). Examples: • Specification of "local" size and complexity metrics and limits (for modules) • Specification of "global" complexity metrics and limits (for overall modules organisation) • Parameter number limit / fixed number of subprogram parameters (Table B.9–4) Information hiding/encapsulation (Table B.9–3): e.g., incentives for using particular language features. Fully defined interface (Table B.9–6). Examples: • Explicit specification of function signatures • Failure assertion programming (Table A.2-3a) and data verification (7.9.2.7), with explicit specification of pre-conditions and post-conditions for functions, of assertions, of data types invariants
Code understandability • Promote code understandability (7.4.4.13) • Readable, understandable and testable (7.4.6)	Naming conventions promoting meaningful, unambiguous names. Example: avoidance of names that could be confounded (e.g., IO and I0). Symbolic names for numeric values. Procedures and guidelines for source code documentation (7.4.4.13). For example: • Explain why's and meanings (and not only what), • Caveats • Side effects Where practicable, the following information shall be contained in the source code (7.4.4.13): • Legal entity (for example: company, author(s), etc.) • Description • Inputs and outputs • Configuration management history (See also Modular Approach)

IEC 61508-3 Requirements & Recommendations	*Coding Standards Suggestions*
Verifiability and testability • Facilitate verification and testing (7.4.4.13) • Facilitate the detection of design or programming mistakes (7.4.4.10) • Formal verification (Table A.5 - 9) • Formal proof (Table A.9 - 1)	• Wrappers for "critical" library functions, to check pre- and post-conditions • Incentives for using language features that can express restrictions on the use of particular data elements or functions (e.g., const) • For tool supported verification: rules for complying with the limitations of the selected tools (provided this does not impair more essential goals) • Limited use of recursion (Table B.1 – 6) and other forms of circular dependencies (See also Modular Approach)
Static verification of conformance to the specified design (7.9.2.12)	Coding guidelines for the implementation of specific design concepts or constraints. For example: • Coding guidelines for cyclic behaviour, with guaranteed maximum cycle time (Table A.2-13a) • Coding guidelines for time-triggered architecture (Table A.2-13b) • Coding guidelines for event-driven architecture, with guaranteed maximum response time (Table A.2-13c) • Loops with a statically determined maximum number of iterations (except for the infinite loop of the cyclic design) • Coding guidelines for static resource allocation (Table A.2-14) and avoidance of dynamic objects (Table B.1–2) • Coding guidelines for static synchronisation of access to shared resources (Table A.2-15) • Coding guidelines to comply with limited use of interrupts (Table B.1–4) • Coding guidelines to avoid dynamic variables (Table B.1–3a) • Online checking of the installation of dynamic variables (Table B.1–3b) • Coding guidelines to ensure compatibility with other programming languages used (7.4.4.10) Guidelines to facilitate traceability with design

IEC 61508-3 Requirements & Recommendations	*Coding Standards Suggestions*
Language subset (Table A.3 - 3) • Proscribe unsafe language features (7.4.4.13) • Use only defined language features (7.4.4.10) • Structured programming (Table A.4 - 6) • Strongly typed programming language (Table A.3 - 2) • No automatic type conversion (Table B.1 - 8)	Exclusion of language features leading to unstructured designs. E.g., • Limited use of pointers (Table B.1–5) • Limited use of recursion (Table B.1–6) • Limited use of C-like unions • Limited use of Ada or C++-like exceptions • No unstructured control flow in programs in higher level languages (Table B.1–7) • One entry/one exit point in subroutines and functions (Table B.9-5) • No automatic type conversion • Limited use of side effects not apparent from functions signatures (e.g., of static variables). No side effects in evaluation of conditions and all forms of assertions. Limited or documented-only use of compiler-specific features. Limited use of potentially misleading language constructs. Rules to be applied when these language features are used nonetheless.
Good programming practice (7.4.4.13)	When applicable: • Coding guidelines to ensure that, when necessary, floating point expressions are evaluated in the right order (e.g., “a-b+c” is not always equal to “a+c-b”) • In floating point comparisons: use only inequalities (less than, less or equal to, greater than, greater or equal to) instead of strict equality • Guidelines regarding conditional compilation and “pre-processing” • Systematic checking of return conditions (success / failure) Documentation, and, when possible, automation of the production of executable code (makefiles). Avoidance of side effects not apparent from functions signatures. When such side effects exist, guidelines to document them. Bracketing when operators precedence is not absolutely obvious. Catching of supposedly impossible situations (e.g., a “default” case in C “switches”). Use of “wrappers” for critical modules, in particular to check pre- and post-conditions and return conditions. Coding guidelines to comply with known compiler errors and limits set by compiler assessment.

C.2.6.3 No dynamic variables or dynamic objects

NOTE This technique/measure is referenced in Tables A.2 and B.1 of IEC 61508-3.

Aim: To exclude

– unwanted or undetected overlay of memory;

– bottlenecks of resources during (safety-related) run-time.

Description: In the case of this measure, dynamic variables and dynamic objects are those variables and objects that have their memory allocated and absolute addresses determined at run-time. The value of allocated memory and its addresses depend on the state of the system at the moment of allocation, which means that it cannot be checked by the compiler or any other off-line tool.

Because the number of dynamic variables and objects, and the existing free memory space for allocating new dynamic variables or objects, depends on the state of the system at the moment of allocation, it is possible for faults to occur when allocating or using the variables or objects. For example, when the amount of free memory at the location allocated by the system is insufficient, the memory contents of another variable can be inadvertently overwritten. If dynamic variables or objects are not used, these faults are avoided.

Restrictions on the use of dynamic objects are needed where the dynamic behaviour cannot be accurately predicted by some static analysis (i.e. in advance of the program execution), and therefore predictable program execution cannot be guaranteed.

C.2.6.4 On-line checking during creation of dynamic variables or dynamic objects

NOTE 1 This technique/measure is referenced in Table B.1 of IEC 61508-3.

Aim: To check that the memory to be allocated to dynamic variables and objects is free before allocation takes place, ensuring that the allocation of dynamic variables and objects during run-time does not impact existing variables, data or code.

Description: In the case of this measure, dynamic variables are those variables that have their memory allocated and absolute addresses determined at run-time (variables in this sense are also the attributes of object instances).

By means of hardware or software, the memory is checked to ensure it is free before a dynamic variable or object is allocated to it (for example, to avoid stack overflow). If allocation is not allowed (for example if the memory at the determined address is not sufficient), appropriate action must be taken. After a dynamic variable or object has been used (for example, after exiting a subroutine) the whole memory which was allocated to it must be freed.

NOTE 2 An alternative is to demonstrate statically that memory will be adequate in all cases.

C.2.6.5 Limited use of interrupts

NOTE This technique/measure is referenced in Table B.1 of IEC 61508-3.

Aim: To keep software verifiable and testable.

Description: The use of interrupts should be restricted. Interrupts may be used if they simplify the system. Software handling of interrupts should be inhibited during critical parts (for example time critical, critical to data changes) of the executed functions. If interrupts are used, then parts not interruptible should have a specified maximum computation time, so that the maximum time for which an interrupt is inhibited can be calculated. Interrupt usage and masking should be thoroughly documented.

C.2.6.6 Limited use of pointers

NOTE This technique/measure is referenced in Table B.1 of IEC 61508-3.

Aim: To avoid the problems caused by accessing data without first checking range and type of the pointer. To support modular testing and verification of software. To limit the consequence of failures.

Description: In the application software, pointer arithmetic may be used at source code level only if pointer data type and value range (to ensure that the pointer reference is within the correct address space) are checked before access. Inter-task communication of the application software should not be done by direct reference between the tasks. Data exchange should be done via the operating system.

C.2.6.7 Limited use of recursion

NOTE This technique/measure is referenced in Table B.1 of IEC 61508-3.

Aim: To avoid unverifiable and untestable use of subroutine calls.

Description: If recursion is used, there must be a clear criterion which makes predictable the depth of recursion.

C.2.7 Structured programming

NOTE This technique/measure is referenced in Table A.4 of IEC 61508-3.

Aim: To design and implement the program in a way that it is practical to analyse without it being executed. The program may contain only an absolute minimum of statistically untestable behaviour.

Description: The following principles should be applied to minimise structural complexity:

– divide the program into appropriately small software modules, ensuring they are decoupled as far as possible and all interactions are explicit;

– compose the software module control flow using structured constructs, that is sequences, iterations and selection;

– keep the number of possible paths through a software module small, and the relation between the input and output parameters as simple as possible;

– avoid complicated branching and, in particular, avoid unconditional jumps (goto) in higher level languages;

– where possible, relate loop constraints and branching to input parameters;

– avoid using complex calculations as the basis of branching and loop decisions.

Features of the programming language which encourage the above approach should be used in preference to other features which are (allegedly) more efficient, except where efficiency takes absolute priority (for example some safety critical systems).

References:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

A Discipline of Programming. E. W. Dijkstra. Englewood Cliffs NJ, Prentice-Hall, 1976

C.2.8 Information hiding/encapsulation

NOTE This technique/measure is referenced in Table B.9 of IEC 61508-3.

Aim: To prevent unintended access to data or procedures and thereby support a good program structure.

Description: Data that is globally accessible to all software elements can be accidentally or incorrectly modified by any of these elements. Any changes to these data structures may require detailed examination of the code and extensive modifications.

Information hiding is a general approach for minimising these difficulties. The key data structures are "hidden" and can only be manipulated through a defined set of access procedures. This allows the internal structures to be modified or further procedures to be added without affecting the functional behaviour of the remaining software. For example, a name directory might have access procedures "insert", "delete" and "find". The access procedures and internal data structures could be re-written (for example to use a different look-up method or to store the names on a hard disk) without affecting the logical behaviour of the remaining software using these procedures.

In this connection, the concept of abstract data types should be used. If direct support is not provided, then it may be necessary to check that the abstraction has not been inadvertently broken.

References:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

On the Design and Development of Program Families. D. L. Parnas. IEEE Trans SE-2, March 1976

C.2.9 Modular approach

NOTE This technique/measure is referenced in Tables A.4 and B.9 of IEC 61508-3.

Aim: Decomposition of a software system into small comprehensible parts in order to limit the complexity of the system.

Description: A modular approach or modularisation contains several rules for the design, coding and maintenance phases of a software project. These rules vary according to the design method employed during design. Most methods contain the following rules:

- a software module (or equivalently, subprogram) should have a single well-defined task or function to fulfil;
- connections between software modules should be limited and strictly defined, coherence in one software module shall be strong;
- collections of subprograms should be built providing several levels of software modules;
- subprogram sizes should be restricted to some specified value, typically two to four screen sizes;
- subprograms should have a single entry and a single exit only;
- software modules should communicate with other software modules via their interfaces – where global or common variables are used they should be well structured, access should be controlled and their use should be justified in each instance;
- all software module interfaces should be fully documented;
- any software module's interface should contain only those parameters necessary for its function. However, this recommendation is complicated by the possibility that a programming language may permit default parameters, or that an object-oriented approach is used.

References:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

C.2.10 Use of trusted/verified software elements

NOTE This technique/measure is referenced in Tables A.2, C.2, A.4 and C.4 of IEC 61508-3.

Aim: To avoid the need for software designs and elements to be extensively revalidated or redesigned for each new application. To take advantage of designs which have not been formally or rigorously verified, but for which considerable operational history is available. To take advantage of a pre-existing software element which has been verified for a different application and for which a body of verification evidence exists.

Description: This measure verifies that the software elements are sufficiently free from systematic design faults and/or operational failures.

It is generally impractical to build a complex system from the most rudimentary parts. It is generally necessary to make use of major subassemblies (“elements”, see IEC 61508-4, 3.4.5 and 3.2.8) that have been previously developed to provide some useful function and which can be reused to implement some part of the new system.

Well-designed and structured PESs are made up of a number of software elements which are clearly distinct and which interact with each other in clearly specified ways. Building up a library of such generally applicable software elements which can be reused in several applications allows much of the resource necessary for validating the designs to be shared by more than one application.

However, for safety related applications it is essential to have sufficient confidence that the new system incorporating these pre-existing elements has the required safety integrity, and that safety is not compromised by some incorrect behaviour of the pre-existing element.

Two viewpoints are possible in order to gain confidence that the behaviour of a pre-existing element can be accurately known:

- to analyse a comprehensive operational history of the element to demonstrate that the element has been “proven-in-use”;

- to assess a body of verification evidence that has been gathered on the behaviour of the element to determine if the element meets the requirements of this standard.

C.2.10.1 Proven-in-use

Only in rare cases will “proven-in-use” (see IEC 61508-4, 3.8.18) be a sufficient argument that a trusted software element achieves the necessary safety integrity. For complex elements with many possible functions (e.g. an operating system), it is essential to establish which functions of the element are actually sufficiently proven-in-use. For example, where a self-test routine is provided to detect faults, if no failure occurs within the operating period, one cannot consider the self-test routine for fault detection as being proven-in-use.

A software element can be considered to be proven-in-use if it fulfils the following criteria:

– unchanged specification;
– systems in different applications;
– at least one year of service history;
– operating time according to the safety integrity level or suitable number of demands; demonstration of a non-safety-related failure rate of less than
  – 10^{-2} per demand (year) with a confidence of 95 % requires 300 operational runs (years),
  – 10^{-5} per demand (year) with a confidence of 99,9 % requires 690 000 operational runs (years);

NOTE 1 See Annex D for some mathematical aspects supporting the above numerical estimates. See also B.5.4 for a similar measure and statistical approach.

– all of the operating experience must relate to a known demand profile of the functions of the software element, to ensure that increased operating experience genuinely leads to an increased knowledge of the behaviour of the software element relative to that demand profile;
– no safety-related failures.

NOTE 2 A failure which may not be safety critical in one context can be safety critical in another, and vice versa.

To enable verification that software element fulfils the criteria, the following must be documented:

– exact identification of each system and its elements, including version numbers (for both software and hardware);
– identification of users, and time of application;
– operating time;
– procedure for the selection of the user-applied systems and application cases;
– procedures for detecting and registering failures, and for removing faults.

C.2.10.2 Assess a body of verification evidence

A pre-existing software element (see IEC 61508-4, 3.2.8) is one that already exists and has not been developed specifically for the current project or SRS. The pre-existing software could be a commercially available product, or it could have been developed by some organisation for a previous product or system. Pre-existing software may or may not have been developed in accordance with the requirements of this standard.

In order to assess the safety integrity of the new system incorporating the pre-existing software, a body of verification evidence is needed to determine the behaviour of the pre-existing element. This may be derived (1) from the element supplier's own documentation and records of the development process of the element, or (2) it may be created or supplemented by additional qualification activities undertaken by the developer of the new safety related system, or by third parties. This is the "Safety Manual for compliant items" that defines the capabilities and limitations of the potentially reusable software element.

In any case, a Safety Manual for compliant items must exist (or must be created) that is adequate to make possible an assessment of the integrity of a specific Safety Function that depends wholly or partly on the reused element. If not, a conservative conclusion must be drawn that the element has not been justified for safety-related reuse. (This is not to say that the element cannot be justified in any case, but simply that insufficient evidence was found in this particular case.)

This standard has specific requirements for the contents of the Safety Manual for compliant items, see IEC 61508-2 Annex D and IEC 61508-3 Annex D, and IEC 61508-3 7.4.2.12 and 7.4.2.13.

As a very brief indication of content, the Safety Manual for compliant items will address the following:

- that the element's design is known and documented;
- the element has been subject to verification and validation using a systematic approach with documented testing and review of all parts of the element's design and code;
- that unused and unneeded functions of the element will not prevent the new system from meeting its safety requirements;
- that all credible failure mechanisms of the element in the new system have been identified and that appropriate mitigation has been implemented.

A functional safety assessment of the new system must establish that the reused element is applied strictly within the limits of capability that have been justified by the evidence and assumptions in the element's Compliant Item Safety Manual.

References:

Component-Based Software Development: Case Studies. Kung-Kiu Lau. World Scientific, 2004, ISBN 9812388281, 9789812388285

Software Reuse and Reverse Engineering in Practice. P. A. V. Hall (ed.), Chapman & Hall, 1992, ISBN 0-412-39980-6

Software criticality analysis of COTS/SOUP. P.Bishop, T.Clement, S.Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003

C.2.11 Traceability

Aim: To maintain consistency between lifecycle stages.

Description: In order to ensure that the software that results from lifecycle activities meets the requirements for correct operation of the safety-related system, it is essential to ensure consistency between the lifecycle stages. A key concept here is that of "traceability" between activities. This is essentially an impact analysis to check (1) that decisions made at an earlier stage are adequately implemented in later stages (forward traceability), and (2) that decisions made at a later stage are actually required and mandated by earlier decisions.

Forward traceability is broadly concerned with checking that a requirement is adequately addressed in later lifecycle stages. Forward traceability is valuable at several points in the safety lifecycle:

- from the system safety requirements to the software safety requirements;
- from the Software Safety Requirements Specification, to the software architecture;
- from the Software Safety Requirements Specification, to the software design;
- from the Software Design Specification, to the module and integration test specifications;
- from the system and software design requirements for hardware/software integration, to the hardware/software integration test specifications;
- from the Software Safety Requirements Specification, to the software safety validation plan;
- from the Software Safety Requirements Specification, to the software modification plan (including reverification and revalidation);
- from the Software Design Specification, to the software verification (including data verification) plan;

- from the requirements of IEC 61508-3 Clause 8, to the plan for software functional safety assessment.

Backward traceability is broadly concerned with checking that every implementation (interpreted in a broad context, and not confined to code implementation) decision is clearly justified by some requirement. If this justification is absent, then the implementation contains something unnecessary that will add to the complexity but not necessarily address any genuine requirement of the safety-related system. Backward traceability is valuable at several points in the safety lifecycle:

- from the safety requirements, to the perceived safety needs;
- from the software architecture, to the Software Safety Requirements Specification;
- from the software detailed design to the software architecture;
- from the software code to the software detailed design;
- from the software safety validation plan, to the Software Safety Requirements Specification;
- from the software modification plan, to the Software Safety Requirements Specification;
- from the software verification (including data verification) plan, to the Software Design Specification.

Reference:

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794

C.2.12 Stateless software design (or limited state design)

NOTE This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: To limit the complexity of software behaviour.

Description: Consider a software program that processes a sequence of transactions: it receives a sequence of inputs and produces an output in response to each input. The program may also memorise some or all of its history in a "state", and may take this state into account when calculating how to respond to future inputs.

Where the program's output in response to a specific input is completely determined by that input, then the program is said to be memoryless or stateless. Each input/output transaction is complete in the sense that no transaction is influenced by any earlier transaction, and a specific input always results in the same associated output.

In contrast, where the program takes account of both its specific input and also its memorised state in calculating its output, then the program is capable of more complex behaviour because it can deliver different outputs in response to the same input on different occasions. The response to a specific input may depend on the context (i.e. the previous inputs and outputs) in which the input is received. A further consideration that is relevant to some applications (typically communications) is that the program's behaviour can be particularly sensitive to changes in the stored state, whether inadvertently or maliciously introduced.

Stateless (or limited state) design is a general approach that aims to minimise the potential complexity of software behaviour by avoiding or minimising the use of state information in the software design.

References:

Introduction to Automata Theory, Languages, and Computation (3rd Edition). J. Hopcroft, R. Motwani, J. Ullman, Addison-Wesley Longman Publishing Co, 2006, ISBN:0321462254

Stateless connections. T. Aura, P Nikander. In Proc International Conference on Information and Communications Security (ICICS'97), ed Yongfei Han. Springer, 1997, ISBN 354063696X, 9783540636960

C.2.13 Offline numerical analysis

NOTE This technique/measure is referenced in Table A.9 of IEC 61508-3.

Aim: To ensure the accuracy of numerical calculations.

Description: Numerical inaccuracy may arise in the calculation of a mathematical function as a consequence of using finite representations of ideal functions and numbers. Truncation error is introduced when a function is approximated by a finite number of terms of an infinite series such as a Fourier series. Rounding error is introduced by the finitely accurate representation of real numbers in a physical computer. When anything but the simplest calculation is performed in floating point, the validity of the calculation must be checked to ensure that the accuracy required by the application is actually achieved.

Reference:

Guide to Scientific Computing. P.R. Turner. CRC Press, 2001, ISBN 0849312426, 9780849312427

C.2.14 Message sequence charts

NOTE This technique/measure is referenced in Tables B.7 and C.17 of IEC 61508-3.

Aim: To assist the capture of system requirements in the early design stages of software development including requirements and software architectural design. In UML, the name "System Sequence Diagram" is used for this notation.

Description: The Message Sequence Chart is a diagrammatic mechanism for describing the behaviour of a system in terms of the communication that takes place between the system actors (and actor may be to a human being, a computer system, or a software element or object, depending on the design phase). For each actor, a vertical "lifeline" is drawn on the diagram and arrows between the lifelines are used to represent messages. Actions upon receipt of messages can be optionally shown on the diagrams as boxes. A collection of scenarios (describing both desirable and undesirable behaviour) is built up as a specification of the required system behaviour. These scenarios have several uses. They can be animated to demonstrate the system behaviour to end-users. They can be transformed into an executable implementation of the system. They can form the basis of test data.

UML contains extensions to the orginal concept of the Message Squence Chart in the form of selection and iteration constructs which allow scenarios to branch and loop, providing a more compact notation. Sub-diagrams can also be defined which can be referenced from a number of higher level sequence diagrams. Timer and external events can also be represented.

References:

"*Message Sequence charts*", D. Harel, P. Thiagarajan. In UML for Real: Design of Embedded Real-Time Systems. ed. L. Lavagno. Springer, 2003, ISBN 1402075014, 9781402075018

ISO/IEC 19501:2005, *Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2*

C.3 Architecture design

C.3.1 Fault detection and diagnosis

NOTE This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: To detect faults in a system, which might lead to a failure, thus providing the basis for counter-measures in order to minimise the consequences of failures.

Description: Fault detection is the activity of checking a system for erroneous states (caused by a fault within the (sub)system to be checked). The primary goal of fault detection is to inhibit the effect of wrong results. A system which acts in combination with parallel elements, relinquishing control when it detects its own results are incorrect, is called self-checking.

Fault detection is based on the principles of redundancy (mainly to detect hardware faults – see IEC 61508-2 Annex A) and diversity (software faults). Some sort of voting is needed to decide on the correctness of results. Special methods applicable are: assertion programming, N-version programming and the diverse monitor technique; and for hardware: introducing additional sensors, control loops, error checking codes, etc.

Fault detection may be achieved by checks in the value domain or in the time domain on different levels, especially physical (temperature, voltage etc), logical (error detecting codes), functional (assertions) or external (plausibility checks). The results of these checks may be stored and associated with the data affected to allow failure tracking.

Complex systems are composed of subsystems. The efficiency of fault detection, diagnosis and fault compensation depends on the complexity of the interactions among the subsystems, which influences the propagation of faults.

Fault diagnosis should be applied at the smallest subsystem level, since smaller subsystems allow a more detailed diagnosis of faults (detection of erroneous states).

Integrated enterprise-wide information systems can routinely communicate the status of safety systems, including diagnostic testing information, to other supervisory systems. If an anomaly is detected, it can be highlighted and used to trigger corrective action before a hazardous situation develops. Lastly, if an incident does occur, documentation of such anomalies can aid the subsequent investigation.

Reference: *Dependability of Critical Computer Systems 1*. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

C.3.2 Error detecting and correcting codes

NOTE This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: To detect and correct errors in sensitive information.

Description: For an information of *n* bits, a coded block of *k* bits is generated which enables *r* errors to be detected and corrected. Two example types are Hamming codes and polynomial codes.

It should be noted that in safety-related systems it will normally be necessary to discard faulty data rather than try to correct it, since only a predetermined fraction of errors may be corrected properly.

Reference:

Fundamentals of Error-correcting Codes, W. Huffman, V. Pless. Cambridge University Press, 2003, ISBN 0521782805, 9780521782807

C.3.3 Failure assertion programming

NOTE This technique/measure is referenced in Table A.17 of IEC 61508-2, and Tables A.2 and C.2 of IEC 61508-3.

Aim: To detect residual software design faults during execution of a program, in order to prevent safety critical failures of the system and to continue operation for high reliability.

Description: The assertion programming method follows the idea of checking a pre-condition (before a sequence of statements is executed, the initial conditions are checked for validity) and a post-condition (results are checked after the execution of a sequence of statements). If either the pre-condition or the post-condition is not fulfilled, the processing reports the error.

For example,

```
assert < pre-condition>;
    action 1;
       :
       :
    action x;
assert < post-condition>;
```

References:

Exploiting Traces in Program Analysis. A. Groce, R. Joshi. Lecture Notes in Computer Science vol 3920, Springer Berlin / Heidelberg, 2006, ISBN 978-3-540-33056-1

Software Development – A Rigorous Approach. C. B. Jones, Prentice-Hall, 1980

C.3.4 Diverse monitor

NOTE This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: To protect against residual specification and implementation faults in software which adversely affect safety.

Description: Two monitoring approaches can be distinguished: (1) the monitor and the monitored function in the same computer, with some guarantee of independence between them; and (2) the monitor and the monitored function in separate computers.

A diverse monitor is an external monitor, implemented on an independent computer to a different specification. This diverse monitor is solely concerned with ensuring that the main computer performs safe, not necessarily correct, actions. The diverse monitor continuously monitors the main computer. The diverse monitor prevents the system from entering an unsafe state. In addition, if it detects that the main computer is entering a potentially hazardous state, the system has to be brought back to a safe state either by the diverse monitor or the main computer.

Hardware and software of the diverse monitor should be classified and qualified according to the appropriate SIL.

Reference:

Requirements based Monitors for Real-Time Systems, D. Peters, D. Parnas. IEEE Transactions on Software Engineering, vol. 28, no. 2, 2002

C.3.5 Software diversity (diverse programming)

NOTE This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: Detect and mask residual software design and implementation faults during execution of a program, in order to prevent safety critical failures of the system, and to continue operation for high reliability.

Description: In diverse programming a given program specification is designed and implemented *N* times in different ways. The same input values are given to the *N* versions, and the results produced by the *N* versions are compared. If the result is considered to be valid, the result is transmitted to the computer outputs.

An essential requirement is that the *N* versions be independent of each other in some sense, so that they do not all fail simultaneously due to the same cause. In practice it may be very difficult to achieve and to demonstrate the version independence that is the foundation of the *N*-version approach.

The *N* versions can run in parallel on separate computers, alternatively all versions can be run on the same computer and the results subjected to an internal vote. Different voting strategies can be used on the *N* versions, depending on the application requirements, as follows.

– If the system has a safe state, then it is feasible to demand complete agreement (all *N* agree) otherwise an output value is used that will cause the system to reach the safe state. For simple trip systems the vote can be biased in the safe direction. In this case the safe action would be to trip if either version demanded a trip. This approach typically uses only two versions (*N*=2).

– For systems with no safe state, majority voting strategies can be employed. For cases where there is no collective agreement, probabilistic approaches can be used in order to maximise the chance of selecting the correct value, for example, taking the middle value, temporary freezing of outputs until agreement returns, etc.

This technique does not eliminate residual software design faults, nor does it avoid errors in the interpretation of the specification, but it provides a measure to detect and mask before they can affect safety.

References:

Modelling software design diversity – a review, B. Littlewood, P. Popov, L. Strigini. ACM Computing Surveys, vol 33, no 2, 2001

The N-Version Approach to Fault-Tolerant Software, A.Avizienis, IEEE Transactions on Software Engineering, vol. SE-11, no. 12 pp.1491-1501, 1985

An experimental evaluation of the assumption of independence in multi-version programming, J.C. Knight, N.G. Leveson. IEEE Transactions on Software Engineering, vol. SE-12, no 1, 1986

In Search of Effective Diversity: a Six Language Study of Fault-Tolerant Flight Control Software. A. Avizienis, M. R. Lyu and W. Schutz. 18th Symposium on Fault-Tolerant Computing, Tokyo, Japan, 27-30 June 1988, IEEE Computer Society Press, 1988, ISBN 0-8186-0867-6

C.3.6 Backward recovery

NOTE This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: To provide correct functional operation in the presence of one or more faults.

Description: If a fault has been detected, the system is reset to an earlier internal state, the consistency of which has been proven before. This method implies saving of the internal state frequently at so-called well-defined checkpoints. This may be done globally (for the complete database) or incrementally (changes only between checkpoints). Then the system has to compensate for the changes which have taken place in the meantime by using journalling (audit trail of actions), compensation (all effects of these changes are nullified) or external (manual) interaction.

References:

Looking into Compensable Transactions. Jing Li, Huibiao Zhu, Geguang Pu, Jifeng He. In Software Engineering Workshop, 2007. SEW 2007. IEEE, 2007, ISBN 978-0-7695-2862-5

Software Fault Tolerance (Trends in Software, No. 3), M. R. Lyu (ed.), John Wiley & Sons, April 1995, ISBN 0471950688

C.3.7 Re-try fault recovery mechanisms

NOTE This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: To attempt functional recovery from a detected fault condition by re-try mechanisms.

Description: In the event of a detected fault or error condition, attempts are made to recover the situation by re-executing the same code. Recovery by re-try can be as complete as a reboot and a re-start procedure or a small re-scheduling and re-starting task, after a software time-out or a task monitoring action. Re-try techniques are commonly used in communication fault or error recovery, and re-try conditions could be flagged from a communication protocol error (checksum, etc.) or from a communication acknowledgement response time-out.

Reference:

Reliable Computer Systems: Design and Evaluation, D.P. Siewiorek, R.S. Schwartz. A.K. Peters Ltd., 1998, ISBN 156881092X, 9781568810928

C.3.8 Graceful degradation

NOTE This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: To maintain the more critical system functions available, despite failures, by dropping the less critical functions.

Description: This technique gives priorities to the various functions to be carried out by the system. The design ensures that if there is insufficient resources to carry out all the system functions, the higher priority functions are carried out in preference to the lower ones. For example, error and event logging functions may be lower priority than system control functions, in which case system control would continue if the hardware associated with error logging were to fail. Further, should the system control hardware fail, but not the error logging hardware, then the error logging hardware would take over the control function.

This is predominantly applied to hardware but is applicable to the total system including software. It must be taken into account from the topmost design phase.

References:

Towards the Integration of Fault, Resource, and Power Management, T. Siridakis. In Computer Safety, Reliability, and Security: 23rd International Conference, SAFECOMP 2004. Eds. Maritta Heisel et. al. Springer, 2004, ISBN 3540231765, 9783540231769

Achieving Critical System Survivability Through Software Architectures, J.C. Knight, E.A. Strunk. Springer Berlin / Heidelberg, 2004, 978-ISBN 3-540-23168-4

The Evolution of Fault-Tolerant Computing. Vol. 1 of Dependable Computing and Fault-Tolerant Systems, Edited by A. Avizienis, H. Kopetz and J. C. Laprie, Springer Verlag, 1987, ISBN 3-211-81941-X

Fault Tolerance, Principle and Practices. T. Anderson and P. A. Lee, Vol. 3 of Dependable Computing and Fault-Tolerant Systems, Springer Verlag, 1987, ISBN 3-211-82077-9

C.3.9 Artificial intelligence fault correction

NOTE 1 This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: To be able to react to possible hazards in a very flexible way by introducing a combination of methods and process models and some kind of on-line safety and reliability analysis.

Description: Fault forecasting (calculating trends), fault correction, maintenance and supervisory actions may be supported by artificial intelligence (AI) based systems in a very efficient way in diverse channels of a system, since the rules might be derived directly from the specifications and checked against these. Certain common faults which are introduced into specifications, by implicitly already having some design and implementation rules in mind, may be avoided effectively by this approach, especially when applying a combination of models and methods in a functional or descriptive manner.

The methods are selected in such a way that faults may be corrected and the effects of failures be minimised, in order to meet the desired safety integrity.

NOTE 2 See C.3.2 for warning about correcting faulty data, and item 5, Table A.2 of IEC 61508-3 for negative recommendations concerning this technique.

Reference:

Fault Diagnosis: Models, Artificial Intelligence, Applications. J. Korbicz, J. Koscielny, Z. Kowalczuk, W. Cholewa. Springer, 2004, ISBN 3540407677, 9783540407676

C.3.10 Dynamic reconfiguration

NOTE This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: To maintain system functionality despite an internal fault.

Description: The logical architecture of the system has to be such that it can be mapped onto a subset of the available resources of the system. The architecture needs to be capable of detecting a failure in a physical resource and then remapping the logical architecture back onto the restricted resources left functioning. Although the concept is more traditionally restricted to recovery from failed hardware units, it is also applicable to failed software units if there is sufficient "run-time redundancy" to allow a software re-try or if there is sufficient redundant data to make the individual and isolated failure be of little importance.

This technique must be considered at the first system design stage.

Reference:

Dynamic Reconfiguration of Software Architectures Through Aspects. C. Costa et al. Lecture Notes in Computer Science, Volume 4758/2007, Springer Berlin / Heidelberg, 2007, ISBN 978-3-540-75131-1

C.3.11 Safety and Performance in real time: Time-Triggered Architecture

NOTE 1 This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: Composability and transparent implementation of fault-tolerance into safety-critical real-time systems with predictable behaviour.

Description: In a Time-Triggered Architecture (TTA) system, all system activities are initiated and based on the progression of a globally synchronised time-base. Each application is assigned a fixed time slot on the time-triggered bus, which contains the messages exchanged between the jobs of each application which can therefore be exchanged only according to a defined schedule. In event-driven systems, system activities are triggered by arbitrary events at unpredictable points in time. The key advantages of a TTA are (see reference Scheidler, Heiner et. al.):

– composability, which greatly reduces the effort required for testing and certifying the system;
– transparent implementation of fault-tolerance, which makes the architecture highly recommendable for safety-critical applications;
– provision of a globally synchronised time-base, which facilitates the design of distributed real-time systems.

Communication between nodes is done using the *Time-Triggered Protocol TTP/C* (see reference Kopetz, Hexel et. al.) according to a static schedule, deciding when to transmit a message and whether a received message is relevant for the particular electronic module or not. Access to the bus is controlled by a cyclic *time-division multiple access (TDMA)* schema derived from the global notion of time.

The TTP/C protocol guarantees (see reference Rushby) four basic services (core services) in a network of TTA nodes (see reference Kopetz, Bauer):

– Deterministic and timely message transport: Transport of messages from the output port of the sending element to the input ports of the receiving elements within an a priori known time bound. A fault-tolerant transport service is offered by a time-triggered communication service that is available via the temporal firewall interface which eliminates control error propagation by design and minimises coupling between elements. The timely transport of messages with minimal latency and jitter is crucial for the achievement of control stability in real-time applications.
– Fault-tolerant Clock Synchronization: The communication controller generates a fault-tolerant synchronised global time base (with a precision within a few clock tics) that is provided to the host subsystem.
– Consistent Diagnosis of Failing Nodes (Membership Service): The communication controller informs every SRU (“smallest replaceable unit”) about the state of every other SRU in a cluster with a latency of less than one TDMA round.
– Strong Fault Isolation: A maliciously faulty host subsystem (including its software) can produce erroneous data outputs, but can never interfere in any other way with the correct operation of the rest of a TTP/C cluster. Fail silence in the temporal domain is guaranteed by the time-triggered behaviour of the communication controller.

NOTE 2 Other time-triggered protocols are FlexRay and TT-Ethernet (time-triggered Ethernet).

References:

Time-Triggered Architecture (TTA). C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, C. Temple. In Advances in Information Technologies: The Business Challenge, ed. J-Y. Roger. IOS Press, 1998, ISBN 9051993854, 9789051993851

A Synchronisation Strategy for a TTP/C Controller. H. Kopetz, R. Hexel, A. Krueger, D. Millinger, A. Schedl. SAE paper 960120, Application of Multiplexing Technology SP 1137, Detroit, SAE Press, Warrendale, 1996

The Time-Triggered Architecture. H. Kopetz, G. Bauer. Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software, October 2002

An Overview of Formal Verification for the Time-Triggered Architecture. J. Rushby: Invited paper, Oldenburg, Germany, September 9-12, 2002. Proceedings FTRTFT 2002, Springer LNCS 2469, 2002, ISBN 978-3-540-44165-6

C.3.12 UML

NOTE This technique/measure is referenced in Table B.7 of IEC 61508-3.

Aim: To provide a comprehensive set of notations for modelling the desired behaviour of complex systems

Description: UML is, as the name implies, a collection of requirements and design notations which is intended to provide comprehensive support for software development. Some parts of UML are based upon notations first introduced in other methods (such as system sequence diagrams and state transition diagrams) and other notations are unique to UML. UML is strongly biased towards object-oriented concepts although some of the notations can be used without any necessity to proceed to an object-oriented programming. UML is supported by a number of commercially available CASE tools, many of which are capable of automatically generating code from the UML models.

The UML notations which are most generally applicable to the specification and design of safety related systems are the following:

- Class diagrams
- Use cases
- Activity diagrams
- State transition diagrams (Statecharts)
- System sequence diagrams

Other UML notations are relevant to the expression of software architectural design (software structure) but are not listed specifically here.

State transition diagrams are described in B.2.3.2 and system sequence diagrams in C.2.14. The other notations are described in the following three subsections.

C.3.12.1 Class diagrams

Class diagrams define the classes of objects with which the software has to deal. They are based upon earlier entity-relationship-attribute diagrams but are adapted for object oriented design. Each class (of which there will be one or more instances known as objects at run time) is represented as a rectangle and the various relationships between the classes are shown as lines or arrows. The operations or methods offered by each class, and the data attributes of each class, can be added to the diagram. The relationships which can be represented consist of both reference relationships with their cardinality (an instance of class

A may refer to one or many instances of class B) and specialisation relationships (Class X is a refinement of class Y) with possibly additional methods and attributes. Multiple inheritance can be depicted.

C.3.12.2 Use cases

Use cases provide a textual description of the desired behaviour of the system in reponse to a particular scenario, usually from the point of view of external actors including human users of the system and external systems. Alternative sub-scenarios within a given use case can be used to represent optional behaviour, especially in error response cases. A collection of use cases is developed to provide a sufficiently complete specification of the system requirements. Use clauses can be the starting point for the development more rigorous models such as system sequence diagrams and activity diagrams.

Use case diagrams provide a pictorial representation of the system and the actors who are involved in the use cases, but are not rigorous and only the text of the use case is important for specification.

C.3.12.3 Activity diagrams

An activity diagram shows the intended sequence of actions carried out by a software element (often an object in an object-oriented design) including sequential and iterative behaviour (some aspects look remarkably like a flowchart). Activity diagrams however allow the actions of a number of elements to be described in parallel, with the interactions between the elements shown by arrows on the diagram. Synchronisation points where an activity must wait for one or more inputs from other activities before it can proceed are shown by a symbol similar to a Petri net node.

Reference:

ISO/IEC 19501:2005, *Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2*

C.4 Development tools and programming languages

C.4.1 Strongly typed programming languages

NOTE This technique/measure is referenced in Table A.3 of IEC 61508-3.

Aim: Reduce the probability of faults by using a language which permits a high level of checking by the compiler.

Description: When a strongly typed programming language is compiled, many checks are made on how variable types are used, for example in procedure calls and external data access. Compilation will fail and an error message be produced for any usage that does not conform to predefined rules.

Such languages usually allow user-defined data types to be defined from the basic language data types (such as integer, real). These types can then be used in exactly the same way as the basic type. Strict checks are imposed to ensure the correct type is used. These checks are imposed over the whole program, even if this is built from separately compiled units. The checks also ensure that the number and the type of procedure arguments match even when referenced from separately compiled software modules.

Strongly typed languages usually support other aspects of good software engineering practice such as easily analysable control structures (for example if.. then.. else, do.. while, etc.) which lead to well-structured programs.

Typical examples of strongly typed languages are Pascal, Ada and Modula 2.

Reference:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

C.4.2 Language subsets

NOTE This technique/measure is referenced in Table A.3 of IEC 61508-3.

Aim: To reduce the probability of introducing programming faults and increase the probability of detecting any remaining faults.

Description: The language is examined to determine programming constructs which are either error-prone or difficult to analyse, for example, using static analysis methods. A language subset is then defined which excludes these constructs.

References:

Practical Experiences of Safety- and Security-Critical Technologies, P. Amey, A.J. Hilton. Ada User Journal, June, 2004

Safer C: Developing Software for High-integrity and Safety-critical Systems. L. Hatton, McGraw-Hill, 1994, ISBN 0077076400, 9780077076405

Requirements for programming languages in safety and security software standard. B. A. Wichmann. Computer Standards and Interfaces. Vol. 14, pp 433-441, 1992

C.4.3 Certified tools and certified translators

NOTE This technique/measure is referenced in Table A.3 of IEC 61508-3.

Aim: Tools are necessary to help developers in the different phases of software development. Wherever possible, tools should be certified so that some level of confidence can be assumed regarding the correctness of the outputs.

Description: The certification of a tool will generally be carried out by an independent, often national, body, against independently set criteria, typically national or international standards. Ideally, the tools used in all development phases (specification, design, coding, testing and validation) and those used in configuration management, should be subject to certification.

To date, only compilers (translators) are regularly subject to certification procedures; these are laid down by national certification bodies and they exercise compilers (translators) against international standards such as those for Ada and Pascal.

It is important to note that certified tools and certified translators are usually certified only against their respective language or process standards. They are usually not certified in any way with respect to safety.

References:

The certification of software tools with respect to software standards, P. Bunyakiati et al. In IEEE International Conference on Information Reuse and Integration, IRI 2007, IEEE, 2007, ISBN 1-4244-1500-4

Certified Testing of C Compilers for Embedded Systems. O. Morgan. In: 3rd Institution of Engineering and Technology Conference on Automotive Electronics. IEEE, 2007, ISBN 978-0-86341-815-0

The Ada Conformity Assessment Test Suite (ACATS), version 2.5, Ada Conformity Assessment Authority, http://www. ic.org/compilerstesting.html , Apr. 2002

C.4.4 Tools and translators: increased confidence from use

NOTE This technique/measure is referenced in Table A.3 of IEC 61508-3.

Aim: To avoid any difficulties due to translator failures which can arise during development, verification and maintenance of a software package.

Description: A translator is used, where there has been no evidence of improper performance over many prior projects. Translators without operating experience or with any serious known faults should be avoided unless there is some other assurance of correct performance (for example, see C.4.4.1).

If the translator has shown small deficiencies, the related language constructs are noted down and carefully avoided during a safety related project.

Another version to this way of working is to restrict the usage of the language to only its commonly used features.

This recommendation is based on the experience from many projects. It has been shown that immature translators are a serious handicap to any software development. They make a safety-related software development generally infeasible.

It is also known, presently, that no method exists to prove the correctness for all tool or translator parts.

C.4.4.1 Comparison of source program and executable code

Aim: To check that the tools used to produce a PROM image have not introduced any errors into the PROM image.

Description: The PROM image is reverse-engineered to obtain the constituent "object" modules. These "object" modules are reverse-engineered into assembly language files. Using suitable techniques the reverse generated assembly language files are compared with the actual source files originally used to produce the PROM.

The major advantage of the technique is that the tools (compilers, linkers etc.) used to produce the PROM image do not have to be validated for all programs. The technique verifies that source file used for the particular safety-related system are correctly transformed.

References:

Demonstrating Equivalence of Source Code and PROM Contents. D. J. Pavey and L. A. Winsborrow. The Computer Journal Vol. 36, No. 7, 1993

Formal demonstration of equivalence of source code and PROM contents: an industrial example. D. J. Pavey and L. A. Winsborrow. Mathematics of Dependable Systems, Ed. C. Mitchell and V. Stavridou, Clarendon Press, 1995, ISBN 0-198534-91-4

Assuring Correctness in a Safety Critical Software Application. L. A. Winsborrow and D. J. Pavey. High Integrity Systems, Vol. 1, No. 5, pp 453-459, 1996

C.4.5 Suitable programming languages

NOTE This technique/measure is referenced in Table A.3 of IEC 61508-3.

Aim: To support the requirements of this International Standard as much as possible, in particular defensive programming, strong typing, structured programming and possibly assertions. The programming language chosen should lead to an easily verifiable code with a minimum of effort and facilitate program development, verification and maintenance.

Description: The language should be fully and unambiguously defined. The language should be user- or problem-orientated rather than processor/platform machine-orientated. Widely used languages or their subsets are preferred to special purpose languages.

In addition to the already referenced features the language should provide for

– block structure;
– translation time checking; and
– run-time type and array bound checking.

The language should encourage

– the use of small and manageable software modules;
– restriction of access to data in specific software modules;
– definition of variable subranges; and
– any other type of error-limiting constructs.

If safe operation of the system is dependent upon real-time constraints, then the language should also provide for exception/interrupt handling.

It is desirable that the language is supported by a suitable translator, appropriate libraries of pre-existing software modules, a debugger and tools for both version control and development.

Currently, at the time of developing this standard, it is not clear whether object-oriented languages are to be preferred to other conventional ones.

Features which make verification difficult and therefore should be avoided are

– unconditional jumps excluding subroutine calls;
– recursion;
– pointers, heaps or any type of dynamic variables or objects;
– interrupt handling at source code level;
– multiple entries or exits of loops, blocks or subprograms;
– implicit variable initialisation or declaration;
– variant records and equivalence; and
– procedural parameters.

Low-level languages, in particular assembly languages, present problems due to their processor/platform machine-orientated nature.

A desirable language property is that its design and use should result in programs whose execution is predictable. Given a suitably defined programming language, there is a subset which ensures that program execution is predictable. This subset cannot (in general) be statically determined, although many static constraints may assist in ensuring predictable execution. This would typically require a demonstration that array indices are within bounds, and that numeric overflow cannot arise, etc.

Table C.1 gives recommendations for specific programming languages.

References:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

IEC 60880:2006, *Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions*

IEC 61131-3:2003, *Programmable controllers – Part 3: Programming languages*

ISO/IEC 1539-1:2004, *Information technology – Programming languages – Fortran – Part 1:Base language*

ISO/IEC 7185:1990, *Information technology – Programming languages – Pascal*

ISO/IEC 8652:1995, *Information technology – Programming languages – Ada*

ISO/IEC 9899:1999, *Programming languages – C*

ISO/IEC 10206:1991, *Information technology – Programming languages – Extended Pascal*

ISO/IEC 10514-1:1996, *Information technology – Programming languages – Part 1: Modula-2, Base Language*

ISO/IEC 10514-3:1998, *Information technology – Programming languages – Part 3: Object Oriented Modula-2*

ISO/IEC 14882:2003, *Programming languages – C++*

ISO/IEC/TR 15942:2000, *Information technology — Programming languages — Guide for the use of the Ada programming language in high integrity systems*

Table C.1 – Recommendations for specific programming languages

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

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

NOTE 1 The recommendationsHR, R and – NR are explained in Annex A of IEC 61508-3.

NOTE 2 System software includes the operating system, drivers, embedded functions and software modules provided as part of the system. The software is typically provided by the safety system vendor. The language subset should be carefully selected to avoid complex structures which may result in implementation faults. Checks should be performed to check for proper use of the language subset.

NOTE 3 The application software is the software developed for a specific safety application. In many cases this software is developed by the end user or by an application oriented contractor. Where a number of programming languages have the same recommendation, the developer should select one which is commonly used by personnel in the industry or facility. The language subset should be carefully selected to avoid complex structures which may result in implementation faults. Checks should be performed to check for proper use of the language subset.

NOTE 4 If a specific language is not listed in the table, it must not be assumed that it is excluded. It should conform to this International Standard.

NOTE 5 There are a number of extensions to the Pascal language including Free Pascal. References to Pascal include these extensions.

NOTE 6 Java is designed to have a run-time garbage collector. A subset of Java can be defined which does not require garbage collection. Some Java implementations provide progressive garbage collection which recovers free memory as the program executes and prevent execution stopping for a period when available memory is exhausted. Hard real time applications should not use any form of garbage collection.

NOTE 7 If the Java implementation requires a run-time interpreter of Java intermediate code, then the interpreter must be treated as part of the safety related software and treated in accordance with the requirements of IEC 61508-3.

NOTE 8 For entries 15-24, see IEC 61131-3.

C.4.6 Automatic software generation

NOTE This technique/measure is referenced in Table A.2 of IEC 61508-3.

Aim: To automate the more error-prone tasks of software implementation.

Description: The system design is described by a model (an executable specification) at a higher level of abstraction than the traditional executable code. The model is transformed automatically by a code generator into executable form. The aim is to improve software quality by eliminating the error-prone manual tasks of coding. A further potential benefit is that more complex designs can be undertaken at the higher abstract level.

References:

Embedded Software Generation from System Level Design Languages, H Yu, R. Domer, D. Gajski. In “ASP-DAC 2004: Proceedings of the ASP-Dac 2004 Asia and South Pacific Design Automation Conference, 2004”, IEEE Circuits and Systems Society. IEEE, 2004, ISBN 0780381750, 9780780381759

Transforming Process Algebra Models into UML State Machines: Bridging a Semantic Gap?. M.F. van Amstel et. al. In Theory and Practice of Model Transformations: First International Conference, ICMT”. ed. A. Vallecillo. Springer, 2008, ISBN 3540699260, 9783540699262

C.4.7 Test management and automation tools

NOTE This technique/measure is referenced in Table A.5 of IEC 61508-3.

Aim: To encourage a systematic and thorough approach to software and system testing.

Description: The use of appropriate support tools mechanises the more labour-intensive and error-prone tasks in system development and brings the capability for a systematic approach to test management. The availability of support encourages a more thorough approach to both normal and regression testing.

Reference:

Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. R.Black, John Wiley and Sons, 2002, ISBN 0471223980, 9780471223986

C.5 Verification and modification

C.5.1 Probabilistic testing

NOTE This technique/measure is referenced in Tables A.5, C.15, A.7 and C.17 of IEC 61508-3.

Aim: To gain a quantitative figure about the reliability properties of the investigated software.

Description: The method produces a statistical estimate of software reliability. This quantitative figure may take into account the related levels of confidence and significance and can give

– a failure probability per demand;
– a failure probability during a certain period of time; and
– a probability of error containment.

From these figures, other parameters may be derived such as:

– probability of failure free execution;
– probability of survival;
– availability;
– MTBF or failure rate; and
– probability of safe execution.

Probabilistic considerations are either based on a probabilistic test or on operating experience. Usually, the number of test cases or observed operating cases is very large. Typically, the testing of the demand mode of operation involves considerably less elapsed time than the continuous mode of operation.

Automated testing tools are normally employed to provide test data and supervise test outputs. Large tests are run on large host computers with the appropriate process simulation periphery. Test data is selected both according to systematic and random hardware viewpoints. The overall test control, for example, guarantees a test data profile, while random selection can govern individual test cases in detail.

Individual test harnesses, test executions and test supervisions are determined by the detailed test aims as described above. Other important conditions are given by the mathematical prerequisites that must be fulfilled if the test evaluation is to meet its intended test aim.

Probabilistic figures about the behaviour of any test object may also be derived from operating experience. Provided the same conditions are met, the same mathematics can be applied as for the evaluation of test results.

In practice, it is very difficult to demonstrate ultra-high levels of reliability using these techniques.

References:

A discussion of statistical testing on a safety-related application. S Kuball, J H R May, Proc IMechE Vol. 221 Part O: J. Risk and Reliability, Institution of Mechanical Engineers, 2007

Estimating the Probability of Failure when Testing Reveals No Failures, W.K. Miller, L.J. Morell, et al.. IEEE Transactions on Software Engineering, VOl. 18, NO.1, pp33-43, January 1992

Reliability estimation from appropriate testing of plant protection software, J. May, G. Hughes, A.D. Lunn. IEE Software Engineering Journal, v10 n6 pp 206-218, Nov 1995 (ISSN: 0268-6961)

Validation of ultra high dependability for software based systems, B. Littlewood and L. Strigini. Comm. ACM 36 (11), 69-80, 1993

C.5.2 Data recording and analysis

NOTE This technique/measure is referenced in Tables A.5 and A.8 of IEC 61508-3.

Aim: To document all data, decisions and rationale in the software project to allow for easier verification, validation, assessment and maintenance.

Description: Detailed documentation is maintained during a project, which could include

- testing performed on each software module;
- decisions and their rationale;
- problems and their solutions.

During and at the conclusion of the project this documentation can be analysed to establish a wide variety of information. In particular, data recording is very important for the maintenance of computer systems as the rationale for certain decisions made during the development project is not always known by the maintenance engineers.

Reference:

Dependability of Critical Computer Systems 2. F. J. Redmill, Elsevier Applied Science, 1989, ISBN ISBN 1851663819, 9781851663811

C.5.3 Interface testing

NOTE This technique/measure is referenced in Table A.5 of IEC 61508-3.

Aim: To detect errors in the interfaces of subprograms.

Description: Several levels of detail or completeness of testing are feasible. The most important levels are tests for

- all interface variables at their extreme values;

– all interface variables individually at their extreme values with other interface variables at normal values;

– all values of the domain of each interface variable with other interface variables at normal values;

– all values of all variables in combination (this will only be feasible for small interfaces);

– the specified test conditions relevant to each call of each subroutine.

These tests are particularly important if the interfaces do not contain assertions that detect incorrect parameter values. They are also important after new configurations of pre-existing subprograms have been generated.

C.5.4 Boundary value analysis

NOTE This technique/measure is referenced in Tables B.2, B.3 and B.8 of IEC 61508-3.

Aim: To detect software errors occurring at parameter limits or boundaries.

Description: The input domain of the program is divided into a number of input classes according to the equivalence relation (see C.5.7). The tests should cover the boundaries and extremes of the classes. The tests check that the boundaries in the input domain of the specification coincide with those in the program. The use of the value zero, in a direct as well as in an indirect translation, is often error-prone and demands special attention:

– zero divisor;

– blank ASCII characters;

– empty stack or list element;

– full matrix;

– zero table entry.

Normally the boundaries for input have a direct correspondence to the boundaries for the output range. Test cases should be written to force the output to its limited values. Consider also if it is possible to specify a test case which causes the output to exceed the specification boundary values.

If the output is a sequence of data, for example a printed table, special attention should be paid to the first and the last elements and to lists containing none, one and two elements.

References:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

C.5.5 Error guessing

NOTE This technique/measure is referenced in Tables B.2 and B.8 of IEC 61508-3.

Aim: To remove common programming mistakes.

Description: Testing experience and intuition combined with knowledge and curiosity about the system under test may add some uncategorised test cases to the designed test case set.

Special values or combinations of values may be error-prone. Some interesting test cases may be derived from inspection checklists. It may also be considered whether the system is robust enough. For example: can the buttons be pushed on the front-panel too fast or too often? What happens if two buttons are pushed simultaneously?

Reference:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

C.5.6 Error seeding

NOTE This technique/measure is referenced in Table B.2 of IEC 61508-3.

Aim: To ascertain whether a set of test cases is adequate.

Description: Some known types of mistake are inserted (seeded) into the program, and the program is executed with the test cases under test conditions. If only some of the seeded errors are found, the test case set is not adequate. The ratio of found seeded errors to the total number of seeded errors is an estimate of the ratio of found real errors to total number errors. This gives a possibility of estimating the number of remaining errors and thereby the remaining test effort.

$$\frac{\text{Found seeded errors}}{\text{Total number of seeded errors}} = \frac{\text{Found real errors}}{\text{Total number of real errors}}$$

The detection of all the seeded errors may indicate either that the test case set is adequate, or that the seeded errors were too easy to find. The limitations of the method are that in order to obtain any usable results, the types of mistake as well as the seeding positions must reflect the statistical distribution of real errors.

References:

Software Fault Injection: Inoculating Programs Against Errors. J. Voas, G. McGraw. Wiley Computer Pub., 1998, ISBN 0471183814, 9780471183815

Faults, Injection Methods, and Fault Attacks. Chong Hee Kim, Jean-Jacques Quisquater, IEEE Design and Test of Computers, vol. 24, no. 6, pp. 544-545, Nov., 2007

Fault seeding for software reliability model validation. A. Pasquini, E. De Agostino. Control Engineering Practice, Volume 3, Issue 7, July 1995. Elsevier Science Ltd

C.5.7 Equivalence classes and input partition testing

NOTE This technique/measure is referenced in Tables B.2 and B.3 of IEC 61508-3.

Aim: To test the software adequately using a minimum of test data. The test data is obtained by selecting the partitions of the input domain required to exercise the software.

Description: This testing strategy is based on the equivalence relation of the inputs, which determines a partition of the input domain.

Test cases are selected with the aim of covering all the partitions previously specified. At least one test case is taken from each equivalence class.

There are two basic possibilities for input partitioning which are

– equivalence classes derived from the specification – the interpretation of the specification may be either input orientated, for example the values selected are treated in the same way, or output orientated, for example the set of values lead to the same functional result;

– equivalence classes derived from the internal structure of the program – the equivalence class results are determined from static analysis of the program, for example the set of values leading to the same path being executed.

References:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

Static Analysis and Software Assurance. D. Wagner, Lecture Notes in Computer Science, Volume 2126/2001, Springer, 2001, ISBN 978-3-540-42314-0

C.5.8 Structure-based testing

NOTE This technique/measure is referenced in Table B.2 of IEC 61508-3.

Aim: To apply tests which exercise certain subsets of the program structure.

Description: Based on analysis of the program, a set of input data is chosen so that a large (and often prespecified target) percentage of the program code is exercised. Measures of code coverage will vary as follows, depending upon the level of rigour required. In all cases, 100 % of the selected coverage metric should be the aim; if it is not possible to achieve 100 % coverage, the reasons why 100 % cannot be achieved should be documented in the test report (for example, defensive code which can only be entered if a hardware problem arises). The first four techniques in the following list are mentioned specifically in the recommendations in Table B.3 of IEC 61508-3 and are widely supported by testing tools; the remaining techniques could also be considered.

– **Entry point (call graph) coverage**: ensure that every subprogram (subroutine or function) has been called at least once (this is the least rigorous structural coverage measurement).

  NOTE In object-oriented languages, there can be several subprograms of the same name which apply to different variants of a polymorphic type (overriding subprograms) which can be invoked by dynamic dispatching. In these cases every such overriding subprogram should be tested.

– **Statements**: ensure that all statements in the code have been executed at least once.

– **Branches**: both sides of every branch should be checked. This may be impractical for some types of defensive code.

– **Compound conditions**: every condition in a compound conditional branch (i.e. linked by AND/OR) is exercised. See MCDC (modified condition decision coverage, ref. DO178B).

– **LCSAJ**: a linear code sequence and jump is any linear sequence of code statements, including conditional statements, terminated by a jump. Many potential subpaths will be infeasible due to constraints on the input data imposed by the execution of earlier code.

– **Data flow**: the execution paths are selected on the basis of data usage; for example, a path where the same variable is both written and read.

– **Basis path**: one of a minimal set of finite paths from start to finish, such that all arcs are included. (Overlapping combinations of paths in this basis set can form any path through that part of the program.) Tests of all basis path has been shown to be efficient for locating errors.

References:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

RTCA, Inc. document DO-178B and EUROCAE document ED-12B, *Software Considerations in Airborne Systems and Equipment Certification*, dated December 1, 1992

C.5.9 Control flow analysis

NOTE This technique/measure is referenced in Table B.8 of IEC 61508-3.

Aim: To detect poor and potentially incorrect program structures.

Description: Control flow analysis is a static testing technique for finding suspect areas of code that do not follow good programming practice. The program is analysed producing a directed graph which can be further analysed for

– inaccessible code, for instance unconditional jumps which leaves blocks of code unreachable;

– knotted code. Well-structured code has a control graph which is reducible by successive graph reductions to a single node. In contrast, poorly structured code can only be reduced to a knot composed of several nodes.

References:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

C.5.10 Data flow analysis

NOTE This technique/measure is referenced in Table B.8 of IEC 61508-3.

Aim: To detect poor and potentially incorrect program structures.

Description: Data flow analysis is a static testing technique that combines the information obtained from the control flow analysis with information about which variables are read or written in different portions of code. The analysis can check for

– variables that may be read before they are assigned a value – this can be avoided by always assigning a value when declaring a new variable;

– variables that are written more than once without being read – this could indicate omitted code;

– variables that are written but never read – this could indicate redundant code.

A data flow anomaly will not always directly correspond to a program fault, but if anomalies are avoided the code is less likely to contain faults.

References:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

C.5.11 Symbolic execution

NOTE This technique/measure is referenced in Table B.8 of IEC 61508-3.

Aim: To show the agreement between the source code and the specification.

Description: The program variables are evaluated after substituting the left-hand side by the right- hand side in all assignments. Conditional branches and loops are translated into Boolean expressions. The final result is a symbolic expression for each program variable. This expression is a formula for the value that the program would calculate if given real data. This can be checked against the expected expression.

A related use of symbolic execution is as a systematic way of generating test data for the paths through the program logic. The symbolic execution facility may be incorporated into an integrated toolset to provide a facility for software element test and code analysis.

References:

Using symbolic execution for verifying safety-critical systems. A. Coen-Porisini, G. Denaro, C. Ghezzi, M. Pezzé. Proceedings of the 8th European software engineering conference, and 9th ACM SIGSOFT international symposium on Foundations of software engineering. ACM, 2001, ISBN:1-58113-390-1

Using symbolic execution to guide test generation. G. Lee, J. Morris, K. Parker, G. Bundell, P. Lam. In Software Testing, Verification and Reliability, vol 15, no 1, 2005. John Wiley & Sons, Ltd

C.5.12 Formal proof (verification)

NOTE This technique/measure is referenced in Tables A.5 and A.9 of IEC 61508-3.

Aim: To prove the correctness of a program with respect to some abstract model of the program, using theoretical and mathematical models and rules.

Description: Testing is a common way to examine the correctness of a program. However, exhaustive testing is generally unachievable given the complexity of programs of practical value, and therefore only a fraction of the possible program behaviour can be examined in this way. In contrast, formal verification applies mathematical operations to a mathematical representation of a program in order to establish that the program behaves as defined for all possible inputs.

Formal verification of a system requires an abstract model of the program and of its required behaviour (i.e. a specification) in a language with a precise mathematical meaning. The specification may be complete, or it may be restricted to specific program properties:

– functional correctness properties, i.e. the program should exhibit a particular functionality.
– safety (i.e. some bad behaviour will never occur) and liveness (i.e. some good behaviour will occur eventually) properties.
– timing properties, i.e. some behaviour will occur at a particular time.

The outcome of formal verification is a rigorous argument that the abstract model of the program is correct with respect to the specification for all possible inputs i.e. the model satisfies the specified properties.

However, the correctness of the model does not directly prove the correctness of the actual program, and a further necessary step is to show that the model is an accurate abstraction of the actual program for the properties of interest. Some program properties of practical interest cannot be formalised mathematically (e.g. most timing and scheduling, or subjective properties such as a “clear and simple” user interface, or indeed any property or design

objective that cannot readily be expressed in a formal language). Formal verification therefore does not completely replace simulation and testing, but instead complements these techniques by providing further evidence to support the program's correct operation for all inputs. While formal verification can ensure the correctness of an abstract model of a program, testing ensures that the actual program behaves as expected.

The use of formal verification at the design phase may significantly reduce development time by discovering significant errors and oversights early in the design phase, and thus reducing the time required iterating between design and testing.

Several formal methods in practical use are described in C.2.4: for instance, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z.

C.5.12.1 Model checking

Model checking is a method for the formal verification of reactive and concurrent systems. Given a finite state structure which describes the behaviors of the system, a property written as a temporal logic formula is checked if it holds or not against the structure. Efficient algorithms (e.g. SPIN, SMV, and UPPAAL) are employed to traverse the whole states of the structure automatically and exhaustively. When the property does not hold, a counterexample is generated. It shows how the property is violated in the structure, and contains very useful information to investigate the system. Model checking can detect "deep bugs" that could escape from the traditional inspection and testing.

Note that model checking is helpful in analysing subtle complexity. This may be useful in some low-SIL applications but caution is needed if subtle complexity exists in high-SIL applications.

References:

Is Proof More Cost-Effective Than Testing?. S. King, R. Chapman, J. Hammond, A. Pryor. IEEE Transactions on Software Engineering, vol. 26 no. 8, August 2000

Model Checking. E. M. Clarke, O. Grumberg, and D. A. Peled. MIT Press, 1999, ISBN 0262032708, 9780262032704

Systems and Software Verification: Model-Checking Techniques and Tools. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. Mckenzie, Springer, 2001, ISBN 3-540-41523-8

Logic in Computer Science: Modelling and Reasoning about Systems. M.Huth and M. Ryan. Cambridge University Press, 2000, ISBN 0521652006, 0521656028

The Spin Model Checker: Primer and Reference Manual. G. J. Holzmann. Addison-Wesley, 2003, ISBN 0321228626, 9780321228628

C.5.12.2 (void)

C.5.13 Complexity metrics

NOTE This technique/measure is referenced in Tables B.9 and C.19 of IEC 61508-3.

Aim: To predict the attributes of programs from properties of the software itself or from its development or test history.

Description: These models evaluate some structural properties of the software and relate this to a desired attribute such as reliability or complexity. Software tools are required to evaluate most of the measures. Some of the metrics which can be applied are given below:

- graph theoretic complexity – this measure can be applied early in the lifecycle to assess trade-offs, and is based on the complexity of the program control graph, represented by its cyclomatic number;
- number of ways to activate a certain software module (accessibility) – the more a software module can be accessed, the more likely it is to be debugged;
- Halstead type metrics science – this measure computes the program length by counting the number of operators and operands; it provides a measure of complexity and size that forms a baseline for comparison when estimating future development resources;
- number of entries and exits per software module – minimising the number of entry/exit points is a key feature of structured design and programming techniques.

Reference:

Metrics and Models in Software Quality Engineering. S.H. Kan. Addison-Wesley, 2003, ISBN 0201729156, 9780201729153

C.5.14 Formal inspections

NOTE This technique/measure is referenced in Table B.8 of IEC 61508-3.

Aim: To reveal defects in a software element.

Description: Formal inspection is a structured process to inspect software material that is carried out by peers of the person producing the material to find defects and to enable the producer to improve the material. The producer should take no part in the inspection process, other than to brief the inspectors during the familiarization stage. Formal inspections may be carried out on specific software elements produced at any phase of the software development life-cycle.

Prior to the inspection taking place the inspectors should become familiar with the materials to be inspected. The inspectors' roles in the inspection process should be clearly defined. An inspection agenda should be prepared. Entry and exit criteria should be defined based on the properties required for the software element. Entry criteria are the criteria or requirements which must be met prior to the inspection taking place. Exit criteria are the criteria or requirements which must be met to complete a specific process.

During the inspection the findings of the inspection should be formally recorded by the moderator, whose role is to facilitate the inspection. A consensus on the findings should be reached by all inspectors. Defects should be classified as either a) requiring rectification prior to acceptance or b) requiring rectification by a given time / milestone. Defects identified should be referred to the producer for subsequent rectification after completion of the inspection. Dependent on the number and scope of identified defects, the moderator may determine it to be necessary for a further inspection of the software material.

References:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Fagan, M. *Design and Code Inspections to Reduce Errors in Program Development*. IBM Systems Journal 15, 3 (1976): 182-211

C.5.15 Walk-through (software)

NOTE This technique/measure is referenced in Table B.8 of IEC 61508-3.

Aim: To reveal discrepancies between the specification and implementation.

Description: Walk-through is an informal technique, carried out by the producer of a software element in the presence of his peers with the objective of finding defects in the software element. They may be carried out on specific software elements produced at any phase of the software development life-cycle.

Specified functions of the safety-related system are examined and evaluated to ensure that the safety-related system conforms to the requirements given in the specification. Any points of doubt concerning the implementation and use of the product are documented so they may be resolved. In contrast with a formal inspection, the author is active during the walkthrough procedure.

References:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

C.5.16 Design review

NOTE This technique/measure is referenced in Table B.8 of IEC 61508-3.

Aim: To reveal defects in the design of the software.

Description: A design review is a formal, documented, comprehensive and systematic examination of the software design to evaluate the design requirements and the capability of the design to meet these requirements and identify problems and propose solutions.

Design Reviews provide the means to assess the status of the design against the input requirements, and the means to identify opportunities for further improvement. As the development life-cycle activities progress, and major detailed design milestones are met, Design Reviews should be held to review all interface aspects, ensure that the design can be verified to ensure that the design meets its requirements, and ensure that the most appropriate design is consistent with the safety requirements. Such a review is primarily intended to verify the work of the designers and should be treated as a confirmation and refining activity.

A rigorous inspection technique such as “sneak circuit analysis” may be used to detect incorrect software behaviour such as an unexpected path or logic flow, unintended outputs, incorrect timing, undesired actions.

References:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

IEC 61160:2005, *Design review*

Space Product Assurance, Sneak analysis - Part 2: Clue list. ECSS-Q-40-04A Part 2. ESA Publications Division, Noordwijk, 1997, ISSN 1028-396X, http://www.everyspec.com/ESA/ECSS-Q-40-04A_Part-2_14981/

C.5.17 Prototyping/animation

NOTE This technique/measure is referenced in Tables B.3 and B.5 of IEC 61508-3.

Aim: To check the feasibility of implementing the system against the given constraints. To communicate the specifier's interpretation of the system to the customer, in order to locate misunderstandings.

Description: A subset of system functions, constraints, and performance requirements are selected. A prototype is built using high-level tools. At this stage, constraints such as the target computer, implementation language, program size, maintainability, reliability and availability need not be considered. The prototype is evaluated against the customer's criteria and the system requirements may be modified in the light of this evaluation.

Reference:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.18 Process simulation

NOTE This technique/measure is referenced in Tables A.7, C.7, B.3 and C.13 of IEC 61508-3.

Aim: To test the function of a software system, together with its interface to the outside world, without allowing it to modify the real world in any way.

Description: The creation of a system, for testing purposes only, which mimics the behaviour of the equipment under control (EUC).

The simulation may be software only or a combination of software and hardware. It must

– provide inputs, equivalent to the inputs which will exist when the EUC is actually installed;

– respond to outputs from the software being tested in a way which faithfully represents the controlled plant;

– have provision for operator inputs to provide any perturbations with which the system under test is required to cope.

When software is being tested the simulation may be a simulation of the target hardware with its inputs and outputs.

References:

EmStar: An Environment for Developing Wireless Embedded Systems Software. J Elson et al. http://cens.ucla.edu/TechReports/9_emstar.pdf

A hardware-software co-simulator for embedded system design and debugging. A. Ghosh et al. In Proceedings of the IFIP International Conference on Computer Hardware Description Languages and Their Applications, IFIP International Conference on Very Large Scale Integration, 1995. IEEE, 1995, ISBN 4930813670, 9784930813671

C.5.19 Performance requirements

NOTE This technique/measure is referenced in Table B.6 of IEC 61508-3.

Aim: To establish demonstrable performance requirements of a software system.

Description: An analysis is performed of both the system and the software requirements specifications to specify all general and specific, explicit and implicit performance requirements.

Each performance requirement is examined in turn to determine

- the success criteria to be obtained;
- whether a measure against the success criteria can be obtained;
- the potential accuracy of such measurements;
- the project stages at which the measurements can be estimated; and
- the project stages at which the measurements can be made.

The practicability of each performance requirement is then analysed in order to obtain a list of performance requirements, success criteria and potential measurements. The main objectives are:

- each performance requirement is associated with at least one measurement;
- where possible, accurate and efficient measurements are selected which can be used as early in the development as possible;
- essential and optional performance requirements and success criteria are specified; and
- where possible, advantage should be taken of the possibility of using a single measurement for more than one performance requirement.

Reference:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.20 Performance modelling

NOTE This technique/measure is referenced in Tables B.2 and B.5 of IEC 61508-3.

Aim: To ensure that the working capacity of the system is sufficient to meet the specified requirements.

Description: The requirements specification includes throughput and response requirements for specific functions, perhaps combined with constraints on the use of total system resources. The proposed system design is compared against the stated requirements by

- producing a model of the system processes, and their interactions;
- determining the use of resources by each process, for example, processor time, communications bandwidth, storage devices, etc;
- determining the distribution of demands placed upon the system under average and worst-case conditions;
- computing the mean and worst-case throughput and response times for the individual system functions.

For simple systems an analytic solution may be sufficient, while for more complex systems some form of simulation may be more appropriate to obtain accurate results.

Before detailed modelling, a simpler "resource budget" check can be used which sums the resources requirements of all the processes. If the requirements exceed designed system capacity, the design is infeasible. Even if the design passes this check, performance modelling may show that excessive delays and response times occur due to resource starvation. To avoid this situation, engineers often design systems to use some fraction (for example 50 %) of the total resources so that the probability of resource starvation is reduced.

Reference:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.21 Avalanche/stress testing

NOTE This technique/measure is referenced in Table B.6 of IEC 61508-3.

Aim: To burden the test object with an exceptionally high workload in order to show that the test object would stand normal workloads easily.

Description: There are a variety of test conditions which can be applied for avalanche/stress testing. Some of these test conditions are:

– if working in a polling mode then the test object gets much more input changes per time unit as under normal conditions;

– if working on demands then the number of demands per time unit to the test object is increased beyond normal conditions;

– if the size of a database plays an important role then it is increased beyond normal conditions;

– influential devices are tuned to their maximum speed or lowest speed respectively;

– for the extreme cases, all influential factors, as far as is possible, are put to the boundary conditions at the same time.

Under these test conditions, the time behaviour of the test object can be evaluated. The influence of load changes can be observed. The correct dimension of internal buffers or dynamic variables, stacks, etc. can be checked.

Reference:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.22 Response timing and memory constraints

NOTE This technique/measure is referenced in Table B.6 of IEC 61508-3.

Aim: To ensure that the system will meet its temporal and memory requirements.

Description: The requirements specification for the system and the software includes memory and response requirements for specific functions, perhaps combined with constraints on the use of total system resources.

An analysis is performed to determine the distribution demands under average and worst-case conditions. This analysis requires estimates of the resource usage and elapsed time of each system function. These estimates can be obtained in several ways, for example comparison with an existing system or the prototyping and benchmarking of time critical systems.

C.5.23 Impact analysis

NOTE This technique/measure is referenced in Table A.8 of IEC 61508-3.

Aim: To determine the effect that a change or an enhancement to a software system will have to other software modules in that software system as well as to other systems.

Description: Prior to a modification or enhancement being performed on the software, an analysis should be undertaken to determine the impact of the modification or enhancement on the software, and to also determine which software systems and software modules are affected.

After the analysis has been completed a decision is required concerning the reverification of the software system. This depends on the number of software modules affected, the criticality of the affected software modules and the nature of the change. Possible decisions are:

– only the changed software module is reverified;

– all affected software modules are reverified; or

– the complete system is reverified.

Reference:

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794

C.5.24 Software configuration management

NOTE This technique/measure is referenced in Table A.8 of IEC 61508-3.

Aim: Software configuration management aims to ensure the consistency of groups of development deliverables as those deliverables change. Configuration management in general applies to both hardware and software development.

Description: Software configuration management is a technique used throughout development (see IEC 61508-3, 6.2.3). In essence, it requires documenting the production of every version of every significant deliverable and of every relationship between different versions of the different deliverables. The resulting documentation allows the developer to determine the effect on other deliverables of a change to one deliverable (especially one of its elements). In particular, systems or subsystems can be reliably re-built from consistent sets of element versions.

References:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

Software Configuration Management: Coordination for Team Productivity. W.A. Babich. Addison-Wesley, 1986, ISBN 0201101610, 9780201101614

CMMI: guidelines for process integration and product improvement, Mary Beth Chrissis, Mike Konrad, Sandy Shrum, Addison-Wesley, 2003, ISBN 0321154967, 9780321154965

C.5.25 Regression validation

NOTE This technique/measure is referenced in Table A.8 of IEC 61508-3.

Aim: To ensure that valid conclusions are drawn from regression testing.

Description: Complete regression testing of a large or complex system will usually require much effort and resource. Where possible, it is desirable to restrict the regression testing to cover only the system aspects of direct interest at that point in the system development. In this partial regression testing it is essential to have a clear understanding of the scope of the partial testing and to draw only valid conclusions regarding the tested state of the system.

Reference:

Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. R.Black, John Wiley and Sons, 2002, ISBN 0471223980, 9780471223986

C.5.26 Animation of specification and design

NOTE This technique/measure is referenced in Table A.9 of IEC 61508-3.

Aim: To guide the software verification by means of a systematic examination of the specification

Description: A representation of the software that is more abstract than the executable code (i.e. a specification or a high level design) is examined to determine the behaviour of the eventual executable software. The examination is automated in some way (depending on the possibilities afforded by the nature and level of abstraction of the higher level representation) so as to simulate the behaviour and outputs of the executable software. One application of this approach is to generate tests (or “oracles”) that can be later applied to the executable software, thus automating to some degree the testing process. Another application is to animate a user interface so that non-technical end-users can gain some appreciation of the detailed meaning of the specification to which the software developers will work. This provides a valuable method of communication between the two groups.

References:

Supporting the Software Testing Process through Specification Animation. T.Miller, P.Strooper. In Proceedings of the First International Conference on Software Engineering and Formal Methods (SEFM'03), ed. P.Lindsay. IEEE Computer Society, IEEE Computer Society, 2003, ISBN 0769519490, 9780769519494

B model animation for external verification. H.Waeselynck, S.Behnia, In Proceedings. of the Second International Conference on Formal Engineering Methods, 1998. IEEE Computer Society, 1998, ISBN 0-8186-9198-0

C.5.27 Model based testing (test case generation)

NOTE This technique/measure is referenced in Table A.5 of IEC 61508-3.

Aim: To facilitate efficient automatic test case generation from system models and to generate highly repeatable test suites.

Description: Model-based Testing (MBT) is a black-box approach in which common testing tasks such as test case generation (TCG) and test results evaluation are based on a model of the system (application) under test (SUT). Typically, but not only, the systems data and user behaviour are modelled using Finite state machines, Markov processes, decision tables or the like (El-Far, 2001, generalized). Additionally, model-based testing can be combined with source code level test coverage measurement, and functional models can be based on existing source code.

Model-based testing is the automatic generation of efficient test cases/procedures using models of system requirements and specified functionality (SoftwareTech, 2009).

Since testing is very expensive, there is a huge demand for automatic test case generation tools. Therefore, model-based testing is currently a very active field of research, resulting in a

large number of available Test Case Generation (TCG) tools. These tools typically extract a test suite from the behavioural part of the model, guaranteeing to meet certain coverage requirements.

The model is an abstract, partial representation of the desired behaviour of the system under test (SUT). From this model, test models are derived, building an abstract test suite. Test cases are derived from this abstract test suite and executed against the system, and tests can be run against the system model as well. MBT with TCG is based on and strongly related to use of formal methods, so recommendations are similar with respect to safety integrity levels (SIL): HR (highly recommended) for higher SILs, and not required for lower SILs.

The specific activities in general are:

– build the model (from system requirements)
– generate expected inputs
– generate expected outputs
– run tests
– compare actual outputs with expected outputs
– decide on further action (modify model, generate more tests, estimate reliability/quality of the software)

Tests can be derived with different methods and techniques for expressing models of user/system behaviour, e.g.

– by using decision tables
– by using finite state machines
– by using grammars
– by using Markov Chain models
– by using state charts
– by theorem proving
– by constraint logic programming
– by model checking
– by symbolic execution
– by using an event-flow model
– reactive system tests: parallel hierarchical finite automaton
– etc.

Model-based testing is specifically targeting recently the safety critical domain. It allows for early exposure of ambiguities in specification and design, provides the capability to automatically generate many non-repetitive efficient tests, to evaluate regression test suites and to assess software reliability and quality, and eases updating of test suites.

A thorough overview is provided by ElFar (2001) and SoftwareTech 2009 (see references), other details and domain specific issues are discussed in the other references.

References:

T. Bauer, F. Böhr, D. Landmann, T. Beletski, R. Eschbach, Robert and J.H. Poore, *From Requirements to Statistical Testing of Embedded Systems* Software Engineering for Automotive Systems - SEAS 2007, ICSE Workshops, Minneapolis, USA

Eckard Bringmann, Andreas Krämer; *Model-based Testing of Automotive Systems* In: ICST, pp.485-493, 2008 International Conference on Software Testing, Verification, and Validation, 2008

Broy M., *Challenges in automotive software engineering*, International conference on Software engineering (ICSE '06),Shanghai, China, 2006

I. K. El-Far and J. A. Whittaker, *Model-Based Software Testing*. Encyclopedia of Software Engineering (edited by J. J. Marciniak). Wiley, 2001

Heimdahl, M.P.E.: *Model-based testing: challenges ahead*, Computer Software and Applications Conference (COMPSAC 2005), 25-28 July 2005, Edinburgh, Scotland, UK, 2005

Jonathan Jacky, Margus Veanes, Colin Campbell, and Wolfram Schulte, *Model-Based Software Testing and Analysis with C#*, ISBN 978-0-521-68761-4, Cambridge University Press 2008

A. Paradkar, *Case Studies on Fault Detection Effectiveness of Model-based Test Generation Techniques*, in ACM SIGSOFT SW Engineering Notes, Proc. of the first int. workshop on Advances in model-based testing A-MOST '05, Vol. 30 Issue 4. ACM Press 2005

S. J. Prowell, *Using Markov Chain Usage Models to Test Complex Systems*, HICSS '05: 38th Annual Hawaii, International Conference on System Sciences, 2005

Mark Utting and Bruno Legeard, *Practical Model-Based Testing: A Tools Approach*, ISBN 978-0-12-372501-1, Morgan-Kaufmann 2007

Hong Zhu et al. (2008). AST '08: *Proceedings of the 3rd International Workshop on Automation of Software Test*. ACM Press. ISBN 978-1-60558-030-2

Model-Based Testing of Reactive Systems Advanced Lecture Series, LNCS 3472, Springer-Verlag, 2005, ISBN 978-3-540-26278-7

Model-based Testing, SoftwareTech July 2009, Vol. 12, No. 2, Software Testing: A Life Cycle Perspective, http://www.goldpractices.com/practices/mbt/

C.6 Functional safety assessment

NOTE Relevant techniques and measures may also be found in B.6.

C.6.1 Decision tables (truth tables)

NOTE This technique/measure is referenced in Tables A.10 and B.7 of IEC 61508-3.

Aim: To provide a clear and coherent specification and analysis of complex logical combinations and relationships.

Description: This method uses two dimensional tables to concisely describe logical relationships between Boolean program variables.

The conciseness and tabular nature of the method makes it appropriate as a means of analysing complex logical combinations expressed in code.

The method is potentially executable if used as a specification.

C.6.2 Software Hazard and Operability Study (CHAZOP, FMEA)

Aim: To determine safety hazards in a proposed or existing system, their possible causes and consequences, and recommend action to minimise the chance of their occurrence.

Description: A team of engineers, with expertise covering the whole system under consideration, participate in a structured examination of a design, through a series of scheduled meetings. They consider both the functional aspects of the design and how the system would operate in practice (including human activity and maintenance). A leader encourages team members to be creative in exposing potential hazards, and drives the procedure by presenting each part of the system in connection with several guide words: "none", "more of", "less of", "part of", "more than" (or "as well as") and "other than". Every applied condition or failure mode is considered for its feasibility, how it could arise, the possible consequences (is there a hazard?), how it could be avoided and if the avoidance technique is worth the expense.

Hazard studies may take place at many stages of project development, but are most effective when performed early enough to influence major design and operability decisions.

The HAZOP technique evolved in the process industry and requires modification for software application. Different derivative methods (or Computer HAZOPs – "CHAZOPs") have been proposed which in general introduce new guide words and/or suggest schemes for systematically covering the system and software architecture.

References:

OF-FMEA: an approach to safety analysis of object-oriented software intensive systems, T. Cichocki, J. Gorski. In Artificial Intelligence and Security in Computing Systems: 9th International Conference, ACS '2002. Ed. J. Soldek. Springer, 2003, ISBN 1402073968, 9781402073960

Software FMEA techniques. P.L.Goddard. In Proc Annual 2000 Reliability and Maintainability Symposium, IEEE, 2000, ISBN: 0-7803-5848-1

Software criticality analysis of COTS/SOUP. P.Bishop, T.Clement, S.Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003

C.6.3 Common cause failure analysis

NOTE 1 This technique/measure is referenced in Table A.10 of IEC 61508-3.

NOTE 2 See also Annex D of IEC 61508-6.

Aim: To determine potential failures in multiple systems or multiple subsystems which would undermine the benefits of redundancy, because of the appearance of the same failures in the multiple parts at the same time.

Description: Systems intended to take care of the safety of a plant often use redundancy in hardware and majority voting. This is to avoid random hardware failures in components or subsystems which would tend to prevent the correct processing of data.

However, some failures can be common to more than one component or subsystem. For example, if a system is installed in one single room, shortcomings in the air-conditioning, might reduce the benefits of redundancy. The same is true for other external effects on the system such as fire, flooding, electromagnetic interference, plane crashes, and earthquakes. The system may also be affected by incidents related to operation and maintenance. It is essential, therefore, that adequate and well- documented procedures are provided for operation and maintenance, and operating and maintenance personnel are extensively trained.

Internal effects are also major contributors to common cause failures. They can stem from design faults in common or identical components and their interfaces, as well as ageing of components. Common cause failure analysis has to search the system for such potential common failures. Methods of common cause failure analysis are: general quality control;

design reviews; verification and testing by an independent team; and analysis of real incidents with feedback of experience from similar systems. The scope of the analysis, however, goes beyond hardware. Even if software diversity is used in different channels of a redundant system, there might be some commonality in the software approaches which could give rise to common cause failure, for example, faults in the common specification.

When common cause failures do not occur exactly at the same time, precautions can be taken by means of comparison methods between the multiple channels which should lead to detection of a failure before this failure is common to all channels. Common cause failure analysis should take this technique into account.

References:

Reliability analysis of hierarchical computer-based systems subject to common-cause failures. L.Xing, L.Meshkat, S.Donohue. Reliability Engineering & System Safety Volume 92, Issue 3, March 2007

C.6.4 Reliability block diagrams

NOTE 1 This technique/measure is referenced in Table A.10 of IEC 61508-3 and is used in Annex B of IEC 61508-6.

NOTE 2 See also B.6.6.7 “Reliability block diagrams”.

Aim: To model, in a diagrammatic form, the set of events that must take place and conditions which must be fulfilled for a successful operation of a system or a task.

Description: The target of the analysis is represented as a success path consisting of blocks, lines and logical junctions. A success path starts from one side of the diagram and continues via the blocks and junctions to the other side of the diagram. A block represents a condition or an event, and the path can pass it if the condition is true or the event has taken place. If the path comes to a junction, it continues if the logic of the junction is fulfilled. If it reaches a vertex, it may continue along all outgoing lines. If there exists at least one success path through the diagram, the target of the analysis is operating correctly.

References:

IEC 61025:2006, *Fault tree analysis (FTA)*

From safety analysis to software requirements. K.M. Hansen, A.P. Ravn, A.P, V Stavridou. IEEE Trans Software Engineering, Volume 24, Issue 7, Jul 1998

IEC 61078:2006, *Analysis techniques for dependability – Reliability block diagram and boolean methods*

Annex D
(informative)

A probabilistic approach to determining software safety integrity for pre-developed software

D.1 General

This annex provides initial guidelines on the use of a probabilistic approach to determining software safety integrity for pre-developed software based on operational experience. This approach is considered particularly appropriate as part of the qualification of operating systems, library modules, compilers and other system software. The annex provides an indication of what is possible, but the techniques should be used only by those who are competent in statistical analysis.

NOTE This annex uses the term confidence level, which is described in IEEE 352. An equivalent term, significance level, is used in IEC 61164.

The techniques could also be used to demonstrate an increase in the safety integrity level of software over time. For example, software built to the requirements of IEC 61508-3 to SIL1 may, after a suitable period of successful operation in a large number of applications, be shown to achieve SIL2.

Table D.1 below shows the number of failure-free demands experienced or hours of failure-free operation needed to qualify for a particular safety integrity level. This table is a summary of the results given in D.2.1 and D.2.3.

Operating experience can be treated mathematically as outlined in D.2 below to supplement or replace statistical testing, and operating experience from several sites may be combined (i.e. by adding the number of treated demands or hours of operation), but only if

– the software version to be used in the E/E/PE safety-related system is identical to the version for which operating experience is being claimed;
– the operational profile of the input space is similar;
– there is an effective system for reporting and documenting failures; and
– the relevant prerequisites (see D.2 below) are satisfied.

Table D.1 – Necessary history for confidence to safety integrity levels

SIL	Low demand mode of operation	Number of treated demands		High demand or continuous mode of operation	Hours of operation in total	
	(Probability of failure to perform its design function on demand)	1-α = 0,99	1-α = 0,95	(Probability of a dangerous failure per hour)	1-α = 0,99	1-α = 0,95
4	$\geq 10^{-5}$ to $< 10^{-4}$	$4{,}6 \times 10^5$	3×10^5	$\geq 10^{-9}$ to $< 10^{-8}$	$4{,}6 \times 10^9$	3×10^9
3	$\geq 10^{-4}$ to $< 10^{-3}$	$4{,}6 \times 10^4$	3×10^4	$\geq 10^{-8}$ to $< 10^{-7}$	$4{,}6 \times 10^8$	3×10^8
2	$\geq 10^{-3}$ to $< 10^{-2}$	$4{,}6 \times 10^3$	3×10^3	$\geq 10^{-7}$ to $< 10^{-6}$	$4{,}6 \times 10^7$	3×10^7
1	$\geq 10^{-2}$ to $< 10^{-1}$	$4{,}6 \times 10^2$	3×10^2	$\geq 10^{-6}$ to $< 10^{-5}$	$4{,}6 \times 10^6$	3×10^6
NOTE 1 1-α represents the confidence level.						
NOTE 2 See D.2.1 and D.2.3 for prerequisites and details of how this table is derived.						

D.2 Statistical testing formulae and examples of their use

D.2.1 Simple statistical test for low demand mode of operation

D.2.1.1 Prerequisites

a) Test data distribution equal to distribution for demands during on-line operation.

b) Test runs are statistically independent from each other, with respect to the cause of a failure.

c) An adequate mechanism exists to detect any failures which may occur.

d) Number of test cases $n > 100$.

e) No failure occurs during the n test cases.

D.2.1.2 Results

Failure probability p (per demand), at the confidence level $1-\alpha$, is given by

$$p \leq 1 - \sqrt[n]{\alpha} \qquad \text{or} \qquad n \geq -\frac{\ln \alpha}{p}$$

D.2.1.3 Example

Table D.2 – Probabilities of failure for low demand mode of operation

$1-\alpha$	P
0,95	$3/n$
0,99	$4{,}6/n$

For a probability of failure on demand of SIL 3 at 95 % confidence the application of the formula gives 30 000 test cases under the conditions of the prerequisites. Table D.1 summarises the results for each safety integrity level.

D.2.2 Testing of an input space (domain) for a low demand mode of operation

D.2.2.1 Prerequisites

The only prerequisite is that the test data is selected to give a random uniform distribution over the input space (domain).

D.2.2.2 Results

The objective is to find the number of tests, n, that are necessary based on the threshold of accuracy, δ, of the inputs for the low demand function (such as a safety shut-down) that is being tested.

Table D.3 – Mean distances of two test points

Dimension of the domain	Mean distance of two test points in direction of an arbitrary axis
1	$\delta = 1/n$
2	$\delta = \sqrt[2]{1/n}$
3	$\delta = \sqrt[3]{1/n}$
k	$\delta = \sqrt[k]{1/n}$
NOTE *k* can be any positive integer. The values 1, 2 and 3 are just examples.	

D.2.2.3 Example

Consider a safety shut-down that is dependent on just two variables, A and B. If it has been verified that the thresholds that partition the input pair of variables A and B are treated correctly to an accuracy of 1 % of A or B's measuring range, the number of uniformly distributed test cases required in the space of A and B is

$$n = 1/\delta^2 = 10^4$$

D.2.3 Simple statistical test for high demand or continuous mode of operation

D.2.3.1 Prerequisites

a) Test data distribution equal to distribution during on-line operation.

b) The relative reduction for the probability of no failure is proportional to the length of the considered time interval and constant otherwise.

c) An adequate mechanism exists to detect any failures which may occur.

d) The test extends over a test time *t*.

e) No failure occurs during *t*.

D.2.3.2 Results

The relationship between the probability of failure λ, the confidence level 1-α and the testing time *t* is

$$\lambda = -\frac{\ln \alpha}{t}$$

The probability of failure is indirectly proportional to the mean operating time between failures:

$$\lambda = \frac{1}{\text{MTBF}}$$

NOTE This standard does not distinguish between the probability of failure per hour and the rate of failures in 1 h. Strictly, the probability of failure, *F*, is related to the failure rate, *f*, by the equation $F = 1\text{-}e^{-ft}$, but the scope of this standard is for failure rates of less than 10^{-5}, and for values this small $F \approx ft$.

D.2.3.3 Example

Table D.4 – Probabilities of failure for high demand or continuous mode of operation

$1-\alpha$	λ
0,95	$3/t$
0,99	$4{,}6/t$

To verify that the mean time between failures is at least 10^8 h with a confidence level of 95 %, a test time of 3×10^8 h is required and the prerequisites must be satisfied. Table D.1 summarises the number of tests required for each safety integrity level.

D.2.4 Complete test

The program is considered as an urn containing a known number *N* of balls. Each ball represents a program property of interest. Balls are drawn at random and replaced after inspection. A complete test is achieved if all the balls are drawn.

D.2.4.1 Prerequisites

a) Test data distribution is such that each of the *N* program properties is tested with equal probability.

b) Test runs are independent from each other.

c) Each occurring failure is detected.

d) Number of test cases $n >> N$.

e) No failure occurs during the *n* test cases.

f) Each test run tests one program property (a program property is what can be tested during one run).

D.2.4.2 Results

The probability *p* to test all program properties is given by

$$p = \sum_{j=0}^{N-1} (-1)^j \binom{N}{j} \left(\frac{N-j}{N}\right)^n \quad \text{or} \quad p = 1 + \sum_{j=1}^{N} (-1)^j \, C_{j,N} \left(\frac{N-j}{N}\right)^n$$

where

$$C_{j,N} = \frac{N(N-1)\ldots(N-j+1)}{j!}$$

For evaluation of this formula usually only the first terms matter since realistic cases are characterised by $n >> N$. The last factor makes all terms for large *j* very small. This is also visible in Table D.5.

D.2.4.3 Example

Consider a program that has been used at several installations for several years. In total, at least $7{,}5 \times 10^6$ runs have been executed. It is estimated that each 100th run fulfils the above prerequisites. So $7{,}5 \times 10^4$ runs made can be taken for statistical evaluation. It is estimated that 4 000 test runs would perform an exhaustive test. The estimates are conservative. According to Table D.5, the probability of not having tested everything equals $2{,}87 \times 10^{-5}$.

For N = 4 000, the values of the first terms depending on *n* are:

Table D.5 – Probability of testing all program properties

n	*P*
5×10^4	$1 - 1{,}49 \times 10^{-2} + 1{,}10 \times 10^{-4} - \ldots$
$7{,}5 \times 10^4$	$1 - 2{,}87 \times 10^{-5} + 4 \times 10^{-10} - \ldots$
1×10^5	$1 - 5{,}54 \times 10^{-8} + 1{,}52 \times 10^{-15} - \ldots$
2×10^5	$1 - 7{,}67 \times 10^{-19} + 2{,}9 \times 10^{-37} - \ldots$

In practice, such estimates should be made so that they are conservative.

D.3 References

Further information on the above techniques can be found in the following documents:

IEC 61164:2004, *Reliability growth – Statistical test and estimation methods*

Verification and Validation of Real-Time Software, Chapter 5. W. J. Quirk (ed.). Springer Verlag, 1985, ISBN 3-540-15102-8

Combining Probabilistic and Deterministic Verification Efforts. W. D. Ehrenberger, SAFECOMP 92, Pergamon Press, ISBN 0-08-041893-7

Ingenieurstatistik. Heinhold/Gaede, Oldenburg, 1972, ISBN 3-486-31743-1

IEEE 352:1987, *IEEE Guide for general principles of reliability analysis of nuclear power generating station safety systems*

Annex E
(informative)

Overview of techniques and measures for design of ASICs

NOTE The overview of techniques and measures contained in this annex and referenced by IEC 61508-2. This annex should not be regarded as either complete or exhaustive.

E.1 Design description in (V)HDL

Aim: Functional description at high level in hardware description language, for example VHDL or Verilog.

Description: Functional description at high abstraction level in hardware description language, for example VHDL or Verilog. The applied hardware description language should allow functional and/or application oriented description and should be abstracted from later implementation details. Dataflows, branches, arithmetical and/or logical operations should be implemented by assignment and operators of the hardware description language, without manual conversion in logical gates of the applied library.

NOTE For simplification “functional description at high abstraction level in hardware description language” will be denoted in the rest of the document as (V)HDL.

Reference:

IEEE VHDL, *Verilog + Standard VHDL Design guide*

E.2 Schematic entry

Aim: Functional description of the circuitry by drawing a circuit plan using gates and/or macros of the vendor library.

Description: Description of the circuit functionality by schematic entry of the circuit plan. The function to be realised should be implemented by instancing (import) the elementary logical circuit elements such as AND, OR, NOT along with macros consisting of complex arithmetical and logical functions, which are then interconnected. Complex circuits should be partitioned considering the functional viewpoints and can be distributed on different drawings, which are hierarchically interconnected. The interconnection signals should be uniquely defined and have explicit signal names over the entire hierarchy. The use of global signals (Labels) should be avoided as far as applicable.

E.3 Structured description

NOTE See also C.2.7 "Structured Programming" and E.12 "Modularization".

Aim: The description of the circuit's functionality should be structured in such a fashion that it is easily readable, i.e. circuit function can be intuitively understood on basis of description without simulation efforts.

Description: Description of the circuit functionality with (V)HDL or by schematic entry. An easily recognisable and modular structure is recommended. Each module should be implemented likewise in the same fashion and should be described in such a way that it is easily readable with clear defined sub functions. A strict distinction between implemented function and interconnection is recommended, i.e. the module, which is implemented by instancing other sub modules, contains explicitly interconnections of the instanced modules and should not contain any circuit logic.

E.4 Proven-in-use tools

Aim: Application of proven-in-use tools to avoid systematic failure by sufficient long-approved practice of the tools in various projects.

Description: Most of the used tools for designing ASICs and FPGAs comprise of sophisticated software, which cannot be considered to operate without any faults with respect to its correct functionality and it is also quite likely that faults might occur due to faulty operation. Therefore only tools with the attribute "proven-in-use" should be preferred for designing ASICs and FPGAs. This implies:

– Application of tools which have been used (in a comparable software version) over a long period of time or high number of users in various projects with equivalent complexity.
– Adequate experience of each ASIC/FPGA designer with the operation of the tool over a long period of time.
– Use of commonly used tools (adequate number of users) so that information regarding known failures with work arounds (version control with "Bug-List") is available. This information should be readily integrated in design flow and helps to avoid systematic failures.
– The consistency check of the internal tool database and the plausibility check avoid faulty output data. Standard tools check the consistency of the internal database, for example the consistency of database between synthesis- and place-and-route-tool, in order to operate with unique data.

NOTE The consistency check is an inherent attribute of the tool under use and the designer has limited influence on it. Therefore, if the possibility of manual consistency check is provided, the designer should use it adequately.

E.5 (V)HDL simulation

NOTE See also E.6 "Functional test on module level".

Aim: Functional verification of circuit described in (V)HDL by means of simulation.

Description: Verification of the function by simulating the entire circuit or each sub module. The (V)HDL simulator detects a sequence of outputs caused by the internal change of the circuit states as the result of applied input stimuli. The verification of the detected output sequence can be carried out either by pretraced sequence of output signals ("Wave form") or by a special environment known as test bench, which is installed during the design process. The chosen simulator should have an attribute "proven-in-use" in order to provide correct results and to mask faulty timing behaviour of the signals (Spikes, tri-state tracing), which might be caused by the simulator itself or faulty modelling.

E.6 Functional test on module level

NOTE See also E.5 "(V)HDL Simulation" and E.13 "Coverage of the verification scenarios".

Aim: Functional verification "Bottom-up".

Description: Verification of the implemented function - for example by simulation - at module level. The module under test will be instanced in a typical virtual test environment known as ”test bench” and stimulated by the test pattern contained in the code. A sufficient high coverage of specified function including all special cases if they exist is at least required. Automatic verification of output sequence by the code of "test bench" should be preferred against manual inspection of output signals.

E.7 Functional test on top level

NOTE See also E.8 "Functional test embedded in system environment”.

Aim: Verification of the ASIC (entire circuit).

Description: The objective of the test is the verification of the entire circuit (ASIC).

E.8 Functional test embedded in system environment

NOTE See also E.7 "Functional test on top level".

Aim: Verification of the specified function embedded in system environment.

Description: This test will verify the entire functionality of the circuit (ASIC) in its system environment, for example with all other components that are located on the circuit boards or elsewhere. A modelling of all relevant components on the circuit board and simulation of ASIC together with the created model to verify the correct functionality inclusive of timing behaviour is recommended. A complete functional test includes also testing of modules that are activated only during presence of failure.

E.9 Restricted use of asynchronous constructs

Aim: Avoidance of typical timing problems during synthesis, avoidance of ambiguity during simulation and synthesis caused by insufficient modelling, design for testability.

Description: Asynchronous constructs such as SET and RESET signals derived over combinatorial logic are susceptible during synthesis and produce circuits with spikes or inverse timing sequence and therefore should be avoided. Also insufficient modelling may not be interpreted properly by the synthesis tool, which causes ambiguous results during simulation. Additionally asynchronous constructs are poorly testable or not at all testable, so that the test coverage of production and on-line test is effectively reduced. The implementation of completely synchronous design with limited number of clock signals is therefore recommended. In systems with multiphase clocks, all the clocks should be derived from one central clock. Clock input of sequential logic should be always supplied exclusively by the clock signal, which does not contain any combinatorial logic. Asynchronous SET and RESET inputs of sequential cells should be always supplied by synchronous signals that do not contain any combinatorial logic. Master SET and RESET should be synchronised using two Flip-flops.

E.10 Synchronisation of primary inputs and control of metastabilities

Aim: Avoidance of ambiguous circuit behaviour as a result of set-up and hold timing violation.

Description: Input signals from external peripherals are generally asynchronous and can change their state arbitrarily. A direct processing of such signals by the synchronous sequential circuit elements of ASIC/FPGA, for example flip-flops leads to set-up and hold time violation resulting in unpredictable timing and functional behaviour of the ASIC/FPGA. Ultimately the metastability of the memory element might occur. Each asynchronous input signal should be therefore synchronised with respect to the synchronous ASIC circuit to avoid the functional ambiguity. Following measures are recommended:

– Input signals should be synchronised with two consecutive memory elements (Flip-flops) or some equivalent circuit in order to achieve a predictable functional behaviour.

– Each asynchronous input signal should be fundamentally synchronised in the above defined manner, i.e. each asynchronous signal is connected with exact one such synchronising circuit. If necessary the output of the synchronising circuit can be used for multiple access.

– The synchronising circuit should be used for stability test of parallel bus signals and to control the data consistency near sampling point

E.11 Design for testability

NOTE 1 See also E.31 "Implementation of test structures".

Aim: Avoidance of not testable or poorly testable structures in order to achieve high test coverage for production test or on-line test.

Description: Design for testability is governed by the avoidance of

– asynchronous constructs
– latches and on-chip tri-state signals
– wired-and / wired-or logic and redundant logic.

The combinatorial depth of the sub circuits plays an important role during the testing. The test pattern required for a complete test increases exponentially with the combinatorial depth of the circuit. Therefore, circuits with high combinatorial depth are only poorly testable with adequate means.

A design for testability orientated approach ensures that the desired test coverage is achieved. As the actual test coverage can be determined at a very late stage in the design process, insufficient consideration of “design for testability” issues might dramatically reduce the achievable test coverage, leading to additional effort.

NOTE 2 The test coverage is usually determined by the percentage of stuck-at faults detected.

E.12 Modularisation

NOTE See also C.2.8 "Information hiding/encapsulation", C.2.9 "Modular approach" and E.3 "Structured description".

Aim: Modular description of the circuit functions.

Description: Distinct partitioning of the total functionality in different modules with limited functions. So the transparency of the modules with the precisely defined interface is established. Every subsystem, at all levels of the design, is clearly defined and is of restricted size (only a few functions). The interfaces between subsystems are kept as simple as possible and the cross-section (i.e. shared data, exchange of information) is minimised. The complexity of individual subsystems is also restricted.

E.13 Coverage of the verification scenarios (test benches)

Aim: Quantitative and qualitative assessment of the applied verification scenarios during the functional test.

Description: The quality of the verification scenarios that is defined during the functional test, i.e. the applied test pattern (stimuli) to verify specified function including all special cases, if they exist, should be qualitatively and/or quantitatively documented. During a quantitative approach the achieved test coverage and the depth of the applied functional tests should be documented. The resulting coverage should meet the levels established for each of the coverage metrics. Any exception will be documented. In the case of a qualitative approach, the number of verified code lines, instructions or paths (“Code coverage”) of the circuit code to be verified should be estimated.

NOTE Exclusive ”Code coverage”-analysis has only a limited relevance, because of high parallelism of the hardware description, and will be justified by exhaustive checks. The code coverage” generally serves to demonstrate the not covered functional code.

E.14 Observation of coding guidelines

Aim: Strict observation of the coding style results in a syntactic and semantic correct circuit code.

Description: Syntactic coding rules help to create an easily readable code and allow a better documentation including version control. Typically, the rules for organising and commenting the instruction blocks or modules can be mentioned here.

Semantic coding rules help avoiding typical implementation problems by avoidance of constructs that lead to faulty synthesis with ambiguous implementation of the circuit function. Typical rules are for example the avoidance of asynchronous constructs or constructs that produce unpredictable timing sequence. In general the use of latches or coupling of data with clock signals lead to such ambiguities.

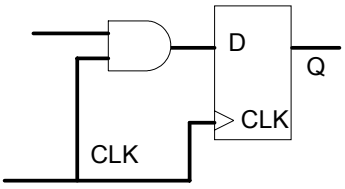
Design guidelines are recommended to avoid systematic design failures during ASIC development process. A coding style in certain aspect limits the design efficiency, offers however in turn the advantage of failure avoidance during ASIC development process. These are in particular:

– avoidance of typical coding infirmity or failure;
– restrictive usage of problematic constructs that produce ambiguous synthesis results;
– design for testability;
– transparent and easy to use code.

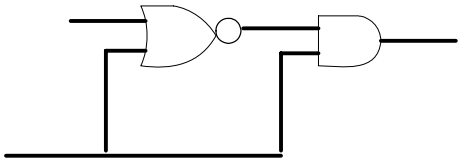
Example of a Coding Style

1. The code should contain as many comments as necessary to understand the function and implementation details. The used conventions have to be defined before the beginning of the design. The compliance of defined conventions should be checked during the design phase.
   1.1. Standard headers include history, cross references to specification, responsibility and design accompanying data such as version number, change requests etc.
   1.2. Easily readable templates: equivalent processes should be described with the same procedure, i.e. usage of predefined templates for recurrent processes (if-then-else, for etc.).
   1.3. Precise and readable naming convention e.g. capital/small letter, pre- and postfix, precise differentiation between port name, internal signals, constants, variables, low active level (xxx_n) etc.
   1.4. Module size restriction and number of ports per module should be limited to increase the readability of code.
   1.5. Structural and defensive code development, e.g. state information should be encapsulated in FSM (Information hiding) to provide the easy alteration of code.
   1.6. Plausibility checks such as range checking etc. should be implemented.
   1.7. Avoidance of following constructs/instructions
      – use of ascending range (x to y) for bus signals;
      – “Disable” instruction in Verilog (corresponds the instruction goto);
      – multidimensional arrays (> 2), records;
      – combination of signed and unsigned data type.
2. Complete synchronous design (clocks derived from central clocks are allowed)
   2.1. Module outputs should be synchronised, it also supports the testability and static timing analysis.
   2.2. Gated clocks should be handled with special precaution.

3. Avoidance of the coupling of data with clock increases the testability, reproducibility between pre- and post-layout data and compliance with RTL (register transfer level) behaviour.
4. Redundant logic is not testable and should be avoided:



Coupling of data and clock signals



Redundant logic

5. Feedback loops in the combinatorial logic should be avoided because they produce unstable design and will not be testable.
6. Full-scan design is recommended.
7. Avoidance of latches increases the testability and reduces the timing constraints during synthesis.
8. Master reset and all asynchronous inputs should be synchronised with two consecutive memory elements (Flip-flops) or an equivalent circuit(s) (metastability).
9. It is recommended to avoid asynchronous set/reset except for master reset.
10. The signals at the module port level should be of the type std_logic or std_logic_vector.

E.15 Application of code checker

Aim: Automatic verification of coding rules ("Coding style") by code checker tool.

Description: The application of code checker helps to a great extent automatically to observe the coding style and generates on-line documentation. However automatic code checker can generally verify the syntax and semantic of the code. The application of such tools should be therefore accompanied by the extension of general coding rules ("tool specific") with the project specific coding rules that the designer has to implement and evaluate separately.

E.16 Defensive programming

See C.2.5.

E.17 Documentation of simulation results

Aim: Documentation of all data needed for a successful simulation in order to verify the specified circuit function.

Description: All the data needed for the functional simulation at module-, chip- or system level should be well documented and archived with the following aims:

– To repeat the simulation at any later phase in turn key fashion.
– To demonstrate the correctness and completeness of all functions specified.

The following database should be archived for this purpose:

– Simulation set-up including complete software of the applied tools, for example simulator, synthesiser with corresponding version and the necessary simulation library.
– Log file of the simulation with full details regarding the time of simulation, applied tools with version and complete report of the work around if it was necessary.
– All relevant simulation results inclusive signal flow, especially in case of manual inspection and documentation of acquired results.

E.18 Code-Inspection

NOTE 1 See also C.5.14 "Formal inspections".

Aim: Review of circuit description.

Description: Review of circuit description should be carried out by

– Checking the coding style.
– Verification of the described functionality against the specification.
– Checking for defensive coding, error and exception handling.

NOTE 2 If the (V)HDL Simulation is not carried out, the completeness of the code inspection and the achieved results should have the equivalent quality that would be achieved by a (V)HDL-simulation.

E.19 Walk-through

NOTE 1 See also C.5.15 "Walk-through".

Aim: Review of the circuit description by walk-through.

Description: A code walk-through consists of a walk-through team selecting a small set of test cases, representative sets of inputs and corresponding expected outputs for the program. The test data is then manually traced through the logic of the program.

NOTE 2 As stand alone measure it should be applied only to the circuits with very low complexity. In the case of the failing (V)HDL-Simulation the completeness of the walk-through and the quality of the achieved results should have the equivalent quality that will be achieved by a (V)HDL-simulation.

Reference:

IEC 61160:2005, *Design review*

E.20 Application of validated soft cores

Aim: Avoidance of failure during the operation of soft cores by application of validated soft cores.

Description: If the vendor validates the soft core, following requirements should be fulfilled:

– The validation of the soft core should be carried out for the operation of the safety related system, having at least an equivalent or higher safety integrity level than the system under plan.
– All the assumptions and confinements, which are necessary for the validation of the soft core, should be complied.
– All the necessary documents for the validation of the soft core should be easily available, see also E.17 "Documentation of simulation results".
– Each vendor specification should be strictly observed and the evidence of the compliance should be documented.

E.21 Validation of the soft core

NOTE See also E.6 "Functional test on module level".

Aim: Avoidance of failure during the operation of the soft core by validation of the soft core during design life cycle.

Description: If the soft core is not explicitly developed for the operation in a safety related system, the generated code should be validated under the same premises that apply for the validation of any source code. This means that all possible test cases should be defined and implemented. The functional verification should be then derived by simulation.

E.22 Simulation of the gate netlist to check timing constraints

Aim: Independent verification of the achieved timing constraint during synthesis.

Description: Simulation of the gate netlist produced by the synthesis including the back-annotation of line delays and gate delays. The stimuli should be derived to stimulate the circuit in such a fashion that it will cover a high percentage of the timing constraints and include all the worst case timing paths. In general, the stimuli needed to perform E.6 "Functional test on module level" or E.7 "Functional test" provide a suitable criteria for the selection of the stimuli, provided sufficient test coverage can be claimed during the functional test. The circuit should be tested under best- and worst-case condition at the maximum specified clock rate.

The timing verification can be carried out by the automatic check of the set-up and hold time of the memory elements (flip-flops) of the target library as well as by the functional verification of the circuit. The functional verification should be primarily performed by observing the outputs of the chip. This can be automated by comparing the output signals of the circuit with an adequate reference model or (V)HDL source code of the circuit. This test is known as a “regression test” and should be preferred against a manual check of the output signals.

NOTE By applying this measure, the timing behaviour of only those paths can be verified which are actually stimulated during the simulation and, therefore, the bespoke measure cannot provide a complete timing analysis of the circuit in general.

E.23 Static analysis of the propagation delay (STA)

Aim: Independent verification of the timing constraints realised during the synthesis.

Description: Static Timing Analysis (STA) analyses all the paths of a netlist (circuit) generated by the synthesis tool considering the back-annotation, i.e. estimated line delays by the synthesis tool, as well as gate delays without performing the actual simulation. Therefore it allows in general a complete analysis of the timing constraint of the entire circuit. The circuit to be tested should be analysed under best- and worst-case condition operating at maximum specified clock rate and accounting for applicable clock jitter and duty cycle skew. The number of non-relevant timing paths can be limited to a certain minimum by adopting a suitable design technique. It is recommended to investigate, analyse and define the used technique that allows easily readable results before beginning with the design.

NOTE It can be assumed that STA covers explicit all the existing timing paths if

a) The timing constraints are properly specified.

b) The circuit under test contains only such timing paths that can be analysed by STA tools, i.e. generally the case with full synchronous circuits.

E.24 Verification of the gate netlist against reference model by simulation

Aim: Functional equivalence check of the synthesised gate netlist.

Description: Simulation of the gate netlist generated by synthesis tool. The applied stimuli for the verification of the circuit by simulation correspond exactly to the stimuli applied during the E.6 "Functional test on module level" and the E.7 "Functional test on top level" for the verification of the function at module level and top level respectively. The functional verification should be primarily performed by observing the outputs of the chip. This can be automated by comparing the output signals of the circuit with an adequate reference model or (V)HDL source code of the circuit. This test is known as a “regression test” and should be preferred to a manual check of the output signals.

NOTE By applying this measure the functional behaviour of only those paths are verified which are actually stimulated during the simulation. The test coverage can, therefore, only be as good as during the original functional test at module- or top-level, respectively. It is possible to complement this measure with a formal equivalence test. In all cases a functional verification of the (V)HDL source code should be carried out with the final netlist generated by the synthesis tool.

E.25 Comparison of the gate netlist with the reference model (formal equivalence test)

Aim: Functional equivalence check which is independent of simulation.

Description: Comparison of the circuit functionality described by the (V)HDL source code with the circuit functionality of the gate netlist generated by synthesis. The tools based on the formal equivalence principle are capable of verifying the functional equivalence of a different representation form of the circuit for example (V)HDL description or netlist description. By applying this measure a functional simulation is not necessary and an independent functional check is feasible. The successful application of this measure can only be guaranteed, if the applied tool is capable of proving complete equivalence and all the discrepancies reported are evaluated either by manual inspection or automatically.

NOTE It is advantageous to combine this measure with E.24 "Verification of the gate netlist against reference model by simulation". In all cases, a functional verification of (V)HDL source code should be carried out with the final netlist generated by the synthesis tool.

E.26 Check of vendor requirements and constraints

Aim: Avoidance of failure during production by checking the vendor requirements.

Description: A careful checking of the vendor requirements and constraints for example minimum and maximum fan-in and fan-out, maximum wire length (line delay), maximum slew rate of the signals, clock skew and so on by the synthesis tool enhances the reliability of the product. Besides the importance of the requirements for the production process, their violation has a great impact on the validity of the applied models that are used for the simulation. So that any violation of the vendor requirements and constraints leads to faulty simulation results producing undesired functionality.

E.27 Documentation of synthesis constraint, results and tools

Aim: Documentation of all defined constraints that are necessary for an optimal synthesis to generate the final gate netlist.

Description: The documentation of all the synthesis constraints and results is indispensable because of the following reasons:

– to reproduce the synthesis at any later phase.
– to generate an independent synthesis results for verification.

Essential documents are:

– Synthesis set-up including the applied tools and synthesis software with the actual version, the applied synthesis library and the defined constraints and scripts.
– Synthesis log file with the time remark, applied tool with version and complete documentation of the synthesis.
– The generated netlist with estimated time delays (standard delay format (SDF) File).

E.28 Application of proven-in-use synthesis tool

Aim: Tool based conversion of (V)HDL description of a circuit in gate netlist.

Description: Tool based mapping of the (V)HDL source code of circuit functionality by connection of the suitable gates and circuit primitives of the target ASIC library. The selected implementation out of a variety of possible implementations that fulfil the desired functionality depends on the most optimal result that is derived by the synthesis constraints such as timing (clock rate) and chip area.

E.29 Application of proven-in-use target library

NOTE See also E.4 "Proven-in-use tools".

Aim: Avoidance of systematic failures caused by a faulty target library.

Description: The synthesis and simulation target library for the development of an ASIC are derived from a common database and, therefore, are not independent. A systematic failure such as:

– ambiguity between real and modelled behaviour of the circuit elements,
– insufficient modelling for example of set-up and hold time,

is one of the typical examples.

Therefore only “proven-in-use” technologies and target libraries should be used for the design of ASICs that perform safety functions. This means:

– The application of target libraries that have been used over a significant long time in projects with comparable complexity and clock rating.
– Availability of the technology and corresponding target library over a sufficient long period, so that enough modelling accuracy of the library can be expected.

E.30 Script based procedures

Aim: Reproducibility of results and automation of the synthesis cycles.

Description: Automatic and script based control of the synthesis cycles including the definition of the applied constraints. Besides a precise documentation of a complete synthesis constraint, it helps to reproduce the netlist after the alteration of the (V)HDL source code under identical conditions.

E.31 Implementation of test structures

Aim: Design of testable ASICs that guarantees the final production test.

Description: Design for testability allows easily testable circuits by implementation of different test structures, for example:

– Scan-path: In a scan technique, either all (full scan design) or part of flip-flops (partial scan design) is connected in a single chain or multiple chains building a chain of shift registers. The scan-path allows an automatic generation of test pattern of the entire logic of a circuit. The tool generating test pattern is called ATPG “Automatic Test Pattern Generator”. The implementation of scan-path improves the testability of a circuit tremendously and allows more than 98 % of test coverage with reasonable effort. It is therefore recommended to implement, if possible, a full scan-path.

– NAND-Tree: In a NAND-tree, all the primary inputs of a circuit are connected in cascading fashion to build a chain. By applying a suitable test pattern (“walking bit”) it is possible to test the switching behaviour (timing and triggering level) of the inputs. NAND-tree offers a straightforward means for the characterisation of primary inputs. Its implementation is recommended, if the switching behaviour of the circuit cannot be tested otherwise.

– Build-in self test (BIST): Self test of the circuit and in particular the self test of the embedded memory can be carried out very efficiently by implementing on-chip test pattern generator. BIST allows an automatic verification of the circuit structure by applying a pseudo-random test pattern and evaluating the signature of the implemented circuit structure. BIST is recommended as additive measure particularly for memory test. The scan-path test can be replaced by a BIST.

– Quiescent current test (IDDQ-test): A static CMOS-circuit consumes a current mainly during switching event. An absolutely defect free circuit consumes therefore negligibly small amount of current (< 1µA, leakage current) as long as the test pattern is held stationary. IDDQ-test is very effective and provides more than 50 % test coverage just after the application of a couple of test patterns. IDDQ-test can be applied on functional test patterns as well as on synthesised test patterns generated by ATPG. This test method has proven to be most helpful in practice and is capable of detecting failure that other tests rarely or even cannot detect. This measure should therefore be applied additive to the regular production tests.

– Boundary-scan: Test architecture implemented for the verification of the interconnection of the components on a printed circuit board according to JTAG standard. Same philosophy can also be applied to verify the interconnection of modules on chip level. Boundary-scan is primarily recommended to improve the testability of the printed circuit board.

E.32 Estimation of test coverage by simulation

Aim: Determination of the achieved test coverage by the implemented test architecture during production test.

Description: Test coverage achieved by the scan-path test, BIST, functional test pattern or any other measures can be determined by fault simulation. During the fault simulation a test pattern is applied to a circuit in which the faults are inserted. A faulty response of the circuit to the applied stimuli corresponds to the faults inserted and thus contributes to the test coverage. The fault simulation allows the detection of stuck-at-faults ”stuck-at-1” and ”stuck-at-0” and the achieved test coverage represents the quality of the test pattern applied. The fault simulation in general can be used very effectively to detect faults associated with logic that is not a part of the scan-path, for example in case of partial scan-paths.

E.33 Estimation of the test coverage by application of ATPG tool

Aim: Determination of the test coverage that can be expected by synthesised test pattern (Scan-path, BIST) during the production test.

Description: Currently, there is variety of procedures, which generate pseudo-random or algorithmic test patterns for a circuit implemented with scan-path. The synthesis tool such as ATPG creates during the synthesis a catalogue of undetected faults. The test coverage can thus be estimated and defines the lower limit of the achieved test coverage with the applied test pattern. It is important to notice that the test coverage is limited to the circuit logic, which

is covered by the scan-path. The modules such as memory, BIST or part of circuits that are not integrated in scan-path are not considered in the estimation of test coverage.

E.34 Justification of proven-in-use for applied hard cores

Aim: Avoidance of systematic failure during the application of hard cores

Description: A hard core is generally regarded as a black box representing the desired functionality and is composed of layout data basis in target technology that provides the desired circuit component. The possible functional failure can be treated in analogy to discrete components like, standard microprocessors, memories etc. The operation of such hard cores without the verification of correct functionality is possible, if for the applied target technology the used core can be considered as proven-in-use component. The rest of the circuit should then be verified intensively.

E.35 Application of validated hard cores

NOTE See also E.6 "Functional test on module level".

Aim: Avoidance of systematic failure during the application of hard cores.

Description: The core validation should be carried out by vendors, because of the complex nature of the core and assumed constraints, during the design phase on the basis of the (V)HDL source code. The validation can be justified only for the configuration and the target technology of the applied component.

E.36 On-line testing of hard cores

NOTE See also E.13 "Coverage of the verification scenarios (test benches)".

Aim: Avoidance of systematic failure during the application of hard cores.

Description: Verification of correct function and implementation of used hard cores by application of on-line tests. In applying this measure an efficient test concept is necessary and the evaluation of the applied concept should be documented.

E.37 Design rule check (DRC)

Aim: Verification of vendor design rules.

Description: Verification of the generated layout with respect to vendor design rules, for example minimum wire lengths, maximum wire lengths and several rules regarding placement of layout structures. A complete and correct run of DRC should be documented in detail.

E.38 Verification of layout versus schematic (LVS)

Aim: Independent verification of the layout.

Description: LVS extracts the circuit functionality from the layout data basis and compares the extracted circuit elements including interconnections with the input netlist. This assures the equivalence of circuit layout with the netlist specifying the circuit functionality. A complete and correct run of LVS should be documented in detail.

E.39 Additional slack (>20 %) for process technologies in use for less than 3 years

Aim: Assurance of the robustness of the implemented circuit functionality even under strong process and parameter fluctuation.

Description: The actual circuit behaviour is defined by number of overlapping physical effects particularly for small structures (for example below 0,5 µm). As a matter of fact, due to the lack of detail knowledge and necessary simplifications an exact model of circuit elements cannot be derived. With decreasing geometrical structures line delays play more and more dominant role. Signal delays along the wire and cross-coupling capacities between the wires grow over proportional. Signal delays are no longer negligible compared to gate delays. Estimated line delays depict increasing risk with decreasing geometrical structures.

Therefore it is recommended to plan an adequate amount of slack (> 20 %) with respect to minimal and maximal timing constraints for circuits designed using processes in use for less than 3 years, in order to guarantee correct operation of the circuit functionality in presence of strongly fluctuating parameters during the production or due to lack of precise modelling.

E.40 Burn-in Test

Aim: Assurance of the robustness of the manufactured chip. Weed out early failures. Bare die chip products do not have to prove their robustness by burn-in but, e.g., by wafer-level stress methods.

Description: The burn-in test should be carried out at the highest tolerable operating temperature (generally 125 °C). The test duration is depending on the aimed SIL-Level or on specific burn-in recommendations for example of the ASIC manufacturer. Burn in can be used to:

– weed out early failures (beginning of bathtube curve with decreasing failure rate);
– prove that early failures are already weeded out during manufacturing and testing (i.e. that devices out of the production line are already in the region of constant failure rate of the bathtube curve).

E.41 Application of proven-in-use device-series

Aim: Assurance of the reliability of the manufactured chips.

Description: The manufacturer of a safety design should have sufficient application experience with the used programmable device technology and the concerning developing tools.

E.42 Proven-in-use production process

Aim: Assurance of the reliability of the manufactured chips.

Description: A proven-in-use production process is characterised by a sufficient series production experience.

E.43 Quality control of the production process

Quality measures and control mechanisms during the device production process ensure a continuous process control. For example optical or electrical control of test structures,

temperature humidity bias-tests or temperature cycle test (see IEC 60068-2-1, IEC 60068-2-2 etc.).

E.44 Manufacturing quality pass of the device

The device quality will be proved by carrying out selected part-stress tests, for example temperature humidity bias-tests or change of temperature tests (see IEC 60068-2-1, IEC 60068-2-2 etc.). The device manufacturer will give proof of it.

E.45 Functional quality pass of the device

All devices will be functionally tested. The device manufacturer will give proof of it.

E.46 Quality standards

The ASIC manufacturer should provide for a sufficient quality management, for example documented within a Quality & Reliability Handbook: ISO 9000 certification or SSQA-, Standard Supplier Quality Assessment

Annex F
(informative)

Definitions of properties of software lifecycle phases

Table F.1 – Software Safety Requirements Specification

(see IEC 61508-3 7.2 and IEC 61508-3 Table C.1)

	Property	Definition
1.1	Completeness with respect to the safety needs to be addressed by software	The Software Safety Requirements Specification addresses all the safety needs and constraints resulting from earlier phases of the safety lifecycle and allocated to the Software. Safety needs and constraints are usually stated in the inputs to the Software Safety Requirements Specification activity. This may include the specification of what the Software must not do or must avoid.
1.2	Correctness with respect to the safety needs to be addressed by software	The Software Safety Requirements Specification providing an appropriate answer to the safety needs and constraints assigned to the Software. The objective is to assure that what is specified will really guarantee safety in all the necessary conditions.
1.3	Freedom from intrinsic specification faults, including freedom from ambiguity	Internal completeness and consistency of the Software Safety Requirements Specification: providing all necessary information for all the functions and situations that can be derived from its statements; expressing no contradicting or inconsistent statements. Contrary to completeness and consistency with respect to safety needs, internal completeness and consistency can be assessed based on the Software Safety Requirements Specification only
1.4	Understandability of safety requirements	The Software Safety Requirements Specification is fully understandable without excessive effort by all those who need to read it, even if they have not been involved earlier in the project, provided that they have the required knowledge. One important objective is to facilitate verification and, possibly, modifications.
1.5	Freedom from adverse interference of non-safety functions with the safety needs to be addressed by software	The Software Safety Requirements Specification avoids requirements that are not necessary to safety of the EUC. The objective is to avoid unnecessary complexity in the design and implementation of the software, so as to reduce the risk of faults and of functions not important to safety interfering with, or jeopardizing, those that are important to safety
1.6	Capability of providing a basis for verification and validation	The Software Safety Requirements Specification gives rise to tests and examinations that generate objective evidence that the software satisfies the Software Safety Requirements Specification.

Table F.2 – Software design and development: software architecture design

(see IEC 61508-3 7.4.3 and IEC 61508-3 Table C.2)

	Property	Definition
2.1	Completeness with respect to Software Safety Requirements Specification	The Software Architecture Design addresses all the safety needs and constraints raised by the Software Safety Requirements Specification.
2.2	Correctness with respect to Software Safety Requirements Specification	The Software Architecture Design provides an appropriate answer to the specified software safety requirements.
2.3	Freedom from intrinsic design faults	The Software Architecture Design and the Design Documentation are free from faults that can be identified independently of any specified software Safety Requirement. Examples: deadlocks, access to unauthorised resources, resource leaks, intrinsic incompleteness (i.e., failure to address all the situations that derive from the design itself).
2.4	Simplicity and understandability. Predictability of behaviour.	The Software Architecture Design allowing a correct and accurate prediction, in all specified situations, of the functioning of the Software. In particular, these situations include erroneous and failure situations. Predictability implies in particular that the functioning does not depend on items that cannot be controlled by designers or users.
2.5	Verifiable and testable design	The Software Architecture Design and the Design Documentation allow and facilitate the production of credible evidence that all the specified software safety requirements are correctly taken into account by the Design and that the Design is free from intrinsic faults. Verifiability may imply derived properties like simplicity, modularity, clarity, testability, provability, etc., depending on the verification techniques used.
2.6	Fault tolerance	The Software Architecture Design gives assurance that the software will have a safe behaviour in the presence of errors (internal errors, errors of operators or of external systems). Defensive design may be active or passive. Active defensive designs may include, features like detection, reporting and containment of errors, graceful degradation and cleaning up of any undesirable side effects prior to the resumption of normal operation. Passive defensive designs include features that guarantee the imperviousness to particular types of errors or particular conditions (avalanches of inputs, particular dates and times) without the software taking any specific action.
2.7	Defence against Common Cause Failure from external events	The Software Architecture Design facilitates the identification of common cause failure modes and effective precautions against failure.

Table F.3 – Software design and development: support tools and programming language

(see IEC 61508-3 7.4.4 and IEC 61508-3 Table C.3)

	Property	Definition
3.1	Support the production of software with the required software properties	Means to provide detection of errors or the elimination of error prone constructs
3.2	Clarity of the operation and functionality of the tool	The provision of comprehensive coverage and feedback on all aspects of operation of the tool
3.3	Correctness and repeatability of output	The consistency and accuracy of the tool output for any given input

Table F.4 – Software design and development: detailed design

(see IEC 61508-3 7.4.5 and IEC 61508-3 7.4.6 and IEC 61508-3 Table C.4)

	Property	Definition
4.1	Completeness with respect to Software Safety Requirements Specification	Methods of detailed software design and production are adopted that ensure that the resulting software addresses all the safety needs and constraints assigned to the Software.
4.2	Correctness with respect to Software Safety Requirements Specification	There exists specific evidence to claim that the safety requirements assigned to the Software have been met by the developed software.
4.3	Freedom from intrinsic design faults	The developed software is free from intrinsic faults. Examples: deadlocks, access to unauthorised resources, resource leaks.
4.4	Simplicity and understandability Predictability of behaviour	The behaviour of the developed software is predictable by objective and convincing testing and analysis.
4.5	Verifiable and testable design	The developed software is verifiable and testable.
4.6	Fault tolerance / Fault detection	Techniques and designs give assurance that the developed software will behave safely in the presence of errors.
4.7	Freedom from common cause failure	Techniques and designs identify common cause failure modes and provide effective precautions against software failure.

Table F.5 – Software design and development: software module testing and integration

(see IEC 61508-3 7.4.7 and IEC 61508-3 7.4.8 and IEC 61508-3 Table C.5)

	Property	Definition
5.1	Completeness of testing and integration with respect to the design specifications	The software testing examines the software behaviour sufficiently thoroughly to ensure that all the requirements of the Software Design Specification have been addressed.
5.2	Correctness of testing and integration with respect to the design specifications (succesful completion)	The module testing task is completed, and there exists specific evidence to claim that the safety requirements have been met.
5.3	Repeatability	Consistent results are produced on repeating the individual assessments carried out as part of the module testing and integration.
5.4	Precisely defined testing configuration	The module testing and integration has been applied to the right version of the elements and the software, with the results claimed, and allows the results to be linked to the specific configuration of the “as-built” software.

Table F.6 – Programmable electronics integration (hardware and software)

(see IEC 61508-3 7.5 and IEC 61508-3 Table C.6)

	Property	Definition
6.1	Completeness of integration with respect to the design specifications	The integration provides the appropriate depth and coverage of the system elements to demonstrate that it can perform the intended functions and does not perform unintended functions under all foreseeable operating conditions and under system failure. This covers the principles used for the verification, the targeted levels of design and aspects of the integration (for example verification of completeness of interaction between modules)
6.2	Correctness of integration with respect to the design specifications (successful completion)	The integration is based on correct assumptions. E.g., correctness of expected results, of the conditions of use considered, representativeness of test environments. The integration task is completed, and there exists specific evidence to claim that the safety requirements have been met.
6.3	Repeatability	Consistent results are produced on repeating the individual assessments carried out as part of the integration.
6.4	Precisely defined integration configuration	The integration gives appropriate assurance that it has been effectively applied as documented, to the right version of the elements and the Software, with the results claimed, and allows the results to be linked to the specific configuration of the “as-built” Software.

Table F.7 – Software aspects of system safety validation

(see IEC 61508-3 7.7 and IEC 61508-3 Table C.7)

	Property	Definition
7.1	Completeness of validation with respect to the Software Design Specification	The software validation addresses all the requirements of the Software Design Specification.
7.2	Correctness of validation with respect to the Software Design Specification (successful completion)	The software validation task is completed, and there exists specific evidence to claim that the safety requirements have been met.
7.3	Repeatability	Consistent results are produced on repeating the individual assessments carried out as part of the software validation.
7.4	Precisely defined validation configuration	The clear and concise definition of the system the requirements the environment

Table F.8 – Software modification

(see IEC 61508-3 7.8 and IEC 61508-3 Table C.8)

	Property	Definition
8.1	Completeness of modification with respect to its requirements	The modification has been properly approved by authorised personnel, with an appropriate understanding of its functional, safety, technical and operational consequences.
8.2	Correctness of modification with respect to its requirements	The modification achieves its specified objectives.
8.3	Freedom from introduction of intrinsic design faults	The modification does not introduce new systematic faults. Examples: division by zero, out of bound indexes or pointers, use of non initialised variables.
8.4	Avoidance of unwanted behaviour	The modification does not introduce any behaviour that, according to constraints stated in the Software Safety Requirements Specification, must be avoided.
8.5	Verifiable and testable design	The software design is such that the effect of the modification is capable of being examined thoroughly.
8.6	Regression testing and verification coverage	The software design is such that effective and thorough regression testing is possible to demonstrate that the software after modification continues to satisfy the Software Safety Requirements Specification.

Table F.9 – Software verification

(see IEC 61508-3 7.9 and IEC 61508-3 Table C.9)

	Property	Definition
9.1	Completeness of verification with respect to the previous phase	The verification is capable of establishing that the software satisfies all relevant requirements of the Software Safety Requirements Specification.
9.2	Correctness of verification with respect to the previous phase (successful completion)	The verification task is completed, and there exists specific evidence to claim that the safety requirements have been met.
9.3	Repeatability	Consistent results are produced on repeating the individual assessments carried out as part of the verification.
9.4	Precisely defined verification configuration;	The verification has been applied to the right version of the elements and the Software, with the results claimed, and allows the results to be linked to the specific configuration of the “as-built” Software.

Table F.10 – Functional safety assessment

(see IEC 61508-3 Clause 8 and IEC 61508-3 Table C.10)

	Property	Definition
10.1	Completeness of functional safety assessment with respect to this standard	The software functional safety assessment produces a clear statement on the extent of compliance found, the judgements made, remedial actions and timescales recommended, the conclusions reached and the recommendations arising for acceptance, qualified acceptance, or rejection and for any time constraints placed on these recommendations.
10.2	Correctness of software functional safety assessment with respect to the Design Specifications (succesful completion)	The software functional safety assessment task is completed, and there exists specific evidence to claim that the safety requirements have been met.
10.3	Traceable closure of all identified issues	There is a clear statement on the extent to which the issues arising during software functional safety assessment have been addressed.
10.4	The ability to modify the software functional safety assessment after change without the need for extensive re-work of the assessment	The software functional safety assessment is capable of being reworked to allow parts of the software functional safety assessment to be re-assessed after software change and for revised conclusions to be achieved, without the need for extensive rework of the complete software functional safety assessment.
10.5	Repeatability	The functional safety assessment is carried out against a consistent, planned and open process on identified individuals and document, which allows scrutiny of the basis for the assessments and the judgement achieved to all those affected by its judgements including system providers, users, maintainers and regulators. The functional safety assessment allows independent competent personnel to repeat the individual assessments carried out as part of the assessment.
10.6	Timeliness	The functional safety assessment is carried out at an appropriate frequency linked to the software safety lifecycle phases and at least prior to determined hazards being present, and it provides timely reporting of deficiencies. The outcome of tests, inspections, analyses etc. are actually available when they are required as input to an assessment decision.
10.7	Precisely defined configuration	The software functional safety assessment allows the results to be linked to the specific configuration of the system which is to be substantiated by the functional safety assessment results.

Annex G
(informative)

Guidance for the development of safety-related object oriented software

All of the recommendations of this standard for software design apply to object oriented software. Because the object-oriented approach presents information differently from procedural or functional approaches, the following list contains those recommendations that need specific consideration:

– understanding class hierarchies, and identification of the software function(s) that will be executed upon the invocation of a given method (including when using an existing class library);
– structure-based testing (IEC 61508-3, Table B.2 and IEC 61508-7, C.5.8)

Tables G.1 and G.2 provide informative guidance for the use of object oriented software to supplement the more general normative guidance provided in IEC 61508-3 Tables A.2 and A.4.

Table G.1 – Object Oriented Software Architecture

	Recommendation	Details	SIL1	SIL2	SIL3	SIL4
G1.1	Traceability of the concept of the application domain to the classes of the architecture.	Note 1	R	HR	HR	HR
G1.2	Use of suitable frames, commonly used combinations of classes and design patterns. NOTE When using existing frames and design patterns, the requirements of pre-developed software apply to these frames and patterns.	Note 2	R	HR	HR	HR

NOTE 1 Traceability from application domain to class architecture is less important.

NOTE 2 EXAMPLE 1: For a part of the intended safety-related project a frame might exist from a non safety-related project that has successfully solved a similar task and that is well known to the project participants. Then use of that frame is recommended.

EXAMPLE 2: It may happen that different algorithms are needed for solving closely connected sub tasks of the safety-related project. The strategy pattern can be chosen for accessing the algorithms.

EXAMPLE 3: Part of the safety-related project may consist of issuing proper warnings to inner and outer stake holders. The observer pattern can be chosen for organizing these warnings. The requirement does not apply for libraries.

NOTE 3 It is usually an abstract basic class that provides access to the derived concrete classes.

Table G.2 – Object Oriented Detailed Design

	Recommendation	SIL1	SIL2	SIL3	SIL4
G2.1	Classes should have only one objective	R	R	HR	HR
G2.2	Inheritance used only if the derived class is a refinement of its basic class	HR	HR	HR	HR
G2.3	Depth of inheritance limited by coding standards	R	R	HR	HR
G2.4	Overriding of operations (methods) under strict control	R	HR	HR	HR
G2.5	Multiple inheritance used only for interface classes	HR	HR	HR	HR
G2.6	Inheritance from unknown classes			NR	NR
G2.7	Verification that the reused object oriented libraries meet the recommendations of this table	HR	HR	HR	HR

NOTE 1 In other words: One class is characterised by having one responsibility, i.e. taking care of closely connected data and the operations on these data.

NOTE 2 Care is required to avoid circular dependencies between objects.

The following terms used above are informally defined here.

Table G.3 – Some Oriented Detailed terms

Term	Informal definition
Basic class	Class that has derived classes. A Basic Class is sometimes called upper class or parent class.
Derived Class	Class (assembly of attributes and operations) that inherits attributes and/or operations from another class (basic class). A derived class is sometimes called subclass or child class.
Frame	Structure of a program, in many cases pre-developed in order to be filled out for the specific application
Overriding	Replacing an operation (method, subroutine) by another operation (method, subroutine) of the same signature and inheritance hierarchy during run-time; property of object-oriented languages or programs; implements polymorphism
Signature of an operation	Name of an operation (subroutine, method), together with its parameters (arguments) and their types, occasionally also their return types. Two signatures are equal if they have the same names, number and types of parameters; in some languages also the return types have to be equal.

Bibliography

[1] IEC 60068-1:1988, *Environmental testing – Part 1: General and guidance*

[2] IEC 60529:1989, *Degrees of protection provided by enclosures (IP Code)*

[3] IEC 60812:2006, *Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*

[4] IEC 60880:2006, *Nuclear power plants – Instrumentation and control systems important to safety – Software aspects for computer-based systems performing category A functions*

[5] IEC 61000-4-1:2006, *Electromagnetic compatibility (EMC) – Part 4-1: Testing and measurement techniques – Overview of IEC 61000-4 series*

[6] IEC 61000-4-5:2005, *Electromagnetic compatibility (EMC) – Part 4-5: Testing and measurement techniques – Surge immunity test*

[7] IEC/TR 61000-5-2:1997, *Electromagnetic compatibility (EMC) – Part 5: Installation and mitigation guidelines – Section 2: Earthing and cabling*

[8] IEC 61025:2006, *Fault tree analysis (FTA)*

[9] IEC 61069-5:1994, *Industrial-process measurement and control – Evaluation of system properties for the purpose of system assessment – Part 5: Assessment of system dependability*

[10] IEC 61078:2006, *Analysis techniques for dependability – Reliability block diagram and boolean methods*

[11] IEC 61131-3:2003, *Programmable controllers – Part 3: Programming languages*

[12] IEC 61160:2005, *Design review*

[13] IEC 61163-1:2006, *Reliability stress screening – Part 1: Repairable assemblies manufactured in lots*

[14] IEC 61164:2004, *Reliability growth – Statistical test and estimation methods*

[15] IEC 61165:2006, *Application of Markov techniques*

[16] IEC 61326-3-1:2008, *Electrical equipment for measurement, control and laboratory use – EMC requirements – Part 3-1: Immunity requirements for safety-related systems and for equipment intended to perform safety-related functions (functional safety) – General industrial applications*

[17] IEC 61326-3-2:2008, *Electrical equipment for measurement, control and laboratory use – EMC requirements – Part 3-2: Immunity requirements for safety-related systems and for equipment intended to perform safety-related functions (functional safety) – Industrial applications with specified electromagnetic environment*

[18] IEC 81346-1:2009, *Industrial systems, installations and equipment and industrial products – Structuring principles and reference designations – Part 1: Basic rules*

[19] IEC 61506:1997, *Industrial-process measurement and control – Documentation of application software*

[20] IEC/TR 61508-0:2005, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 0: Functional safety and IEC 61508*

[21] IEC 61511 (all parts), *Functional safety – Safety instrumented systems for the process industry sector*

[22] IEC 62061:2005, *Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems*

[23] IEC 62308:2006, *Equipment reliability – Reliability assessment methods*

[24] ISO/IEC 1539-1:2004, *Information technology – Programming languages – Fortran – Part 1: Base language*

[25] ISO 5807:1985, *Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts*

[26] ISO/IEC 7185:1990, *Information technology – Programming languages – Pascal*

[27] ISO/IEC 8631:1989, *Information technology – Program constructs and conventions for their representation*

[28] ISO/IEC 8652:1995, *Information technology – Programming languages – Ada*

[29] ISO 8807:1989, *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*

[30] ISO/IEC 9899:1999, *Programming languages – C*

[31] ISO/IEC 10206:1991, *Information technology – Programming languages – Extended Pascal*

[32] ISO/IEC 10514-1:1996, *Information technology – Programming languages – Part 1: Modula-2, Base Language*

[33] ISO/IEC 10514-3:1998, *Information technology – Programming languages – Part 3: Object Oriented Modula-2*

[34] ISO/IEC 13817-1:1996, *Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language*

[35] ISO/IEC 14882:2003, *Programming languages – C++*

[36] ISO/IEC/TR 15942:2000, *Information technology — Programming languages — Guide for the use of the Ada programming language in high integrity systems*

[37] IEC 61800-5-2, *Electromagnetic compatibility (EMC) – Part 5: Installation and mitigation guidelines – Section 2: Earthing and cabling*

[38] IEC 60601 (all parts), *Medical electrical equipment*

[39] IEC 60068-2-1, *Environmental testing – Part 2-1: Tests – Test A: Cold*

[40] IEC 60068-2-2, *Environmental testing – Part 2-2: Tests – Test B: Dry heat*

[41] ISO 9000*, Quality management systems – Fundamentals and vocabulary*

[42] IEC 61508-1:2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1: General requirements*

[43] IEC 61508-2:2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems*

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

[45] IEC 61508-6: 2010, *Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 6: Guidelines on the application of IEC 61508-2 and IEC 61508-3*

Index

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

SÉCURITÉ FONCTIONNELLE DES SYSTÈMES ÉLECTRIQUES/ÉLECTRONIQUES/ÉLECTRONIQUES PROGRAMMABLES RELATIFS À LA SÉCURITÉ –

Partie 7: Présentation de techniques et mesures

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de propriété intellectuelle ou de droits analogues. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de propriété et de ne pas avoir signalé leur existence.

La norme internationale CEI 61508-7 a été établie par le sous-comité 65A: Aspects systèmes, du comité d'études 65 de la CEI: Mesure, commande et automation dans les processus industriels.

Cette deuxième édition annule et remplace la première édition publiée en 2000 dont elle constitue une révision technique.

La présente édition a fait l'objet d'une révision approfondie et intègre de nombreux commentaires reçus lors des différentes phases de révision.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65A/554/FDIS	65A/578/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61508, présentées sous le titre général *Sécurité fonctionnelle des systèmes électriques / électroniques / électroniques programmables relatifs à la sécurité*, peut être consultée sur le site web de la CEI.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

INTRODUCTION

Les systèmes comprenant des composants électriques et/ou électroniques sont utilisés depuis de nombreuses années pour exécuter des fonctions relatives à la sécurité dans la plupart des secteurs d'application. Des systèmes à base d'informatique (dénommés de manière générique systèmes électroniques programmables) sont utilisés dans tous les secteurs d'application pour exécuter des fonctions non relatives à la sécurité, mais aussi de plus en plus souvent relatives à la sécurité. Si l'on veut exploiter efficacement et en toute sécurité la technologie des systèmes informatiques, il est indispensable de fournir à tous les responsables suffisamment d'éléments relatifs à la sécurité pour les guider dans leurs prises de décisions.

La présente Norme internationale présente une approche générique de toutes les activités liées au cycle de vie de sécurité de systèmes électriques et/ou électroniques et/ou électroniques programmables (E/E/PE) qui sont utilisés pour réaliser des fonctions de sécurité. Cette approche unifiée a été adoptée afin de développer une politique technique rationnelle et cohérente concernant tous les systèmes électriques relatifs à la sécurité. Un objectif principal de cette approche est de faciliter le développement de normes internationales de produit et d'application sectorielle basées sur la série CEI 61508.

NOTE 1 Des exemples de normes internationales de produit et d'application sectorielle basées sur la série CEI 61508 sont donnés dans la Bibliographie (voir références [21], [22] et [37]).

Dans la plupart des cas, la sécurité est obtenue par un certain nombre de systèmes fondés sur diverses technologies (par exemple mécanique, hydraulique, pneumatique, électrique, électronique, électronique programmable). En conséquence, toute stratégie de sécurité doit non seulement prendre en compte tous les éléments d'un système individuel (par exemple, les capteurs, les appareils de commande et les actionneurs), mais également prendre en considération tous les systèmes relatifs à la sécurité comme des éléments individuels d'un ensemble complexe. Par conséquent, la présente Norme internationale, bien que traitant des systèmes E/E/PE relatifs à la sécurité, peut aussi fournir un cadre de sécurité susceptible de concerner les systèmes relatifs à la sécurité basés sur d'autres technologies.

Il est admis qu'il existe une grande variété d'applications utilisant des systèmes E/E/PE relatifs à la sécurité dans un grand nombre de secteurs, et couvrant un large éventail de complexité et de potentiel de dangers et de risques. Pour chaque application particulière, les mesures de sécurité requises dépendent de nombreux facteurs propres à l'application. La présente Norme internationale, de par son caractère général, rend désormais possible la prescription de ces mesures dans les futures normes internationales de produit et d'application sectorielle, ainsi que dans les révisions des normes déjà existantes.

La présente Norme internationale

– concerne toutes les phases appropriées du cycle de vie de sécurité global des systèmes E/E/PE et du logiciel (par exemple, depuis le concept initial, en passant par la conception, l'installation, l'exploitation et la maintenance, jusqu'à la mise hors service) lorsque les systèmes E/E/PE permettent d'exécuter des fonctions de sécurité,
– a été élaborée dans le souci de la prise en compte de l'évolution rapide des technologies; le cadre fourni par la présente Norme internationale est suffisamment solide et étendu pour pourvoir aux évolutions futures,
– permet l'élaboration de normes internationales de produit et d'application sectorielle concernant les systèmes E/E/PE relatifs à la sécurité; il convient que l'élaboration de normes internationales de produit et d'application sectorielle dans le cadre de la présente norme, permette d'atteindre un haut niveau de cohérence (par exemple, pour ce qui est des principes sous-jacents, de la terminologie, etc.) à la fois au sein de chaque secteur d'application, et d'un secteur à l'autre. La conséquence en sera une amélioration en termes de sécurité et de gains économiques,
– fournit une méthode de définition d'une spécification des exigences de sécurité nécessaire pour obtenir la sécurité fonctionnelle requise des systèmes E/E/PE relatifs à la sécurité,

– adopte une approche basée sur les risques qui permet de déterminer les exigences en matière d'intégrité de sécurité,

– introduit les niveaux d'intégrité de sécurité pour la spécification du niveau cible d'intégrité de sécurité des fonctions de sécurité devant être réalisées par les systèmes E/E/PE relatifs à la sécurité,

NOTE 2 La norme ne spécifie aucune exigence de niveau d'intégrité de sécurité pour aucune fonction de sécurité, ni comment le niveau d'intégrité de sécurité est déterminé. Elle fournit en revanche un cadre conceptuel basé sur les risques, ainsi que des exemples de méthodes.

– fixe des objectifs chiffrés de défaillance pour les fonctions de sécurité exécutées par les systèmes E/E/PE relatifs à la sécurité, qui sont en rapport avec les niveaux d'intégrité de sécurité,

  – en mode de fonctionnement à faible sollicitation, la limite inférieure est fixée pour une probabilité moyenne de défaillance dangereuse de 10^{-5} en cas de sollicitation,

  – en mode de fonctionnement continu ou à sollicitation élevée, la limite inférieure est fixée à une fréquence moyenne de défaillance dangereuse de 10^{-9} [h^{-1}],

NOTE 3 Un système E/E/PE relatif à la sécurité unique n'implique pas nécessairement une architecture à un seul canal.

NOTE 4 Il peut être possible de concevoir des systèmes relatifs à la sécurité ayant des valeurs plus basses pour l'intégrité de sécurité cible dans le cas de systèmes non complexes. Il est toutefois considéré que ces limites représentent ce qui peut être réalisé à l'heure actuelle pour des systèmes relativement complexes (par exemple des systèmes électroniques programmables relatifs à la sécurité).

– établit des exigences fondées sur l'expérience et le jugement acquis dans le domaine des applications industrielles afin d'éviter des anomalies systématiques ou pour les maintenir sous contrôle. Même si, en général, la probabilité d'occurrence des défaillances systématiques ne peut être quantifiée, la norme permet cependant pour une fonction de sécurité spécifique, de déclarer que l'objectif chiffré de défaillance associé à cette fonction de sécurité peut être réputé atteint si toutes les exigences de la norme sont remplies,

– introduit une capabilité systématique s'appliquant à un élément du fait qu'il permet d'assurer que l'intégrité de sécurité systématique satisfait aux exigences du niveau d'intégrité de sécurité spécifié,

– adopte une large gamme de principes, techniques et mesures pour la réalisation de la sécurité fonctionnelle des systèmes E/E/PE relatifs à la sécurité, mais n'utilise pas de manière explicite le concept de sécurité intrinsèque. Les principes de " sécurité intrinsèque " peuvent toutefois être applicables, l'adoption de ces concepts étant par ailleurs acceptable sous réserve de la satisfaction aux exigences des articles concernés de la norme.

SÉCURITÉ FONCTIONNELLE DES SYSTÈMES ÉLECTRIQUES/ÉLECTRONIQUES/ÉLECTRONIQUES PROGRAMMABLES RELATIFS À LA SÉCURITÉ –

Partie 7: Présentation de techniques et mesures

1 Domaine d'application

1.1 La présente partie de la CEI 61508 contient une présentation de différentes techniques et mesures de sécurité pertinentes pour la CEI 61508-2 et la CEI 61508-3.

Il convient que les références citées soient considérées comme des références fondamentales des méthodes et outils, ou comme des exemples; elles peuvent ne pas représenter la technologie de pointe.

1.2 Les CEI 61508-1, CEI 61508-2, CEI 61508-3 et CEI 61508-4 sont des publications fondamentales de sécurité, bien que ce statut ne soit pas applicable dans le contexte des systèmes E/E/PE de faible complexité relatifs à la sécurité (voir 3.4.3 de la CEI 61508-4). En tant que publications fondamentales de sécurité, ces normes sont destinées à être utilisées par les comités d'études pour la préparation des normes conformément aux principes contenus dans le Guide CEI 104 et le Guide ISO/CEI 51. Les CEI 61508-1, CEI 61508-2, CEI 61508-3 et CEI 61508-4 sont également destinées à être utilisées comme publications autonomes. La fonction de sécurité horizontale de la présente norme internationale ne s'applique pas aux appareils médicaux conformes à la série CEI 60601.

1.3 Une des responsabilités incombant à un comité d'études consiste, dans toute la mesure du possible, à utiliser les publications fondamentales de sécurité pour la préparation de ses publications. Dans ce contexte, les exigences, les méthodes ou les conditions d'essai de cette publication fondamentale de sécurité ne s'appliquent que si elles sont indiquées spécifiquement ou incluses dans les publications préparées par ces comités d'études.

1.4 La Figure 1 illustre la structure générale des parties 1 à 7 de la CEI 61508 et montre le rôle que la CEI 61508-7 joue dans la réalisation de la sécurité fonctionnelle pour les systèmes E/E/PE relatifs à la sécurité.

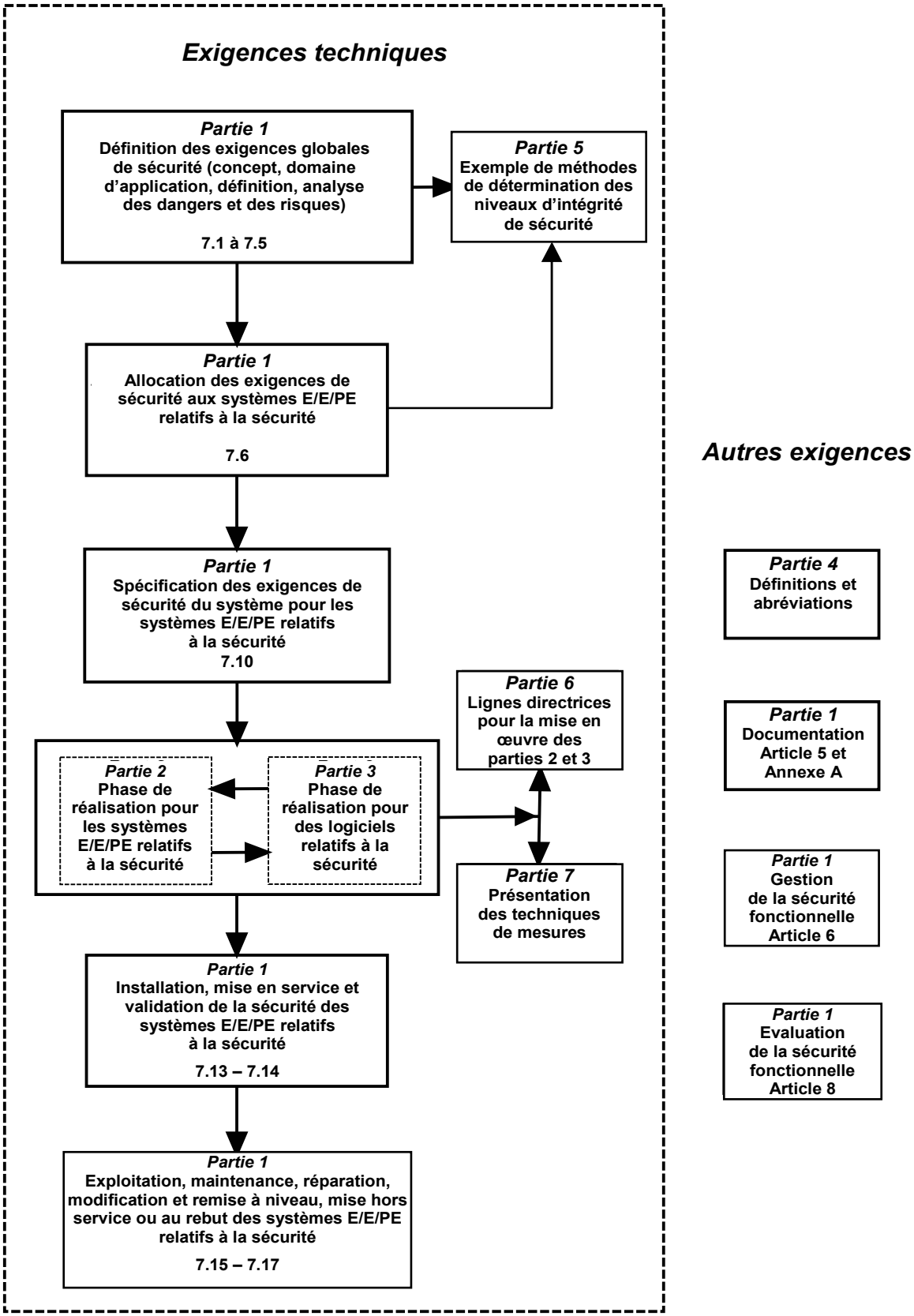


Figure 1 – Structure générale de la série CEI 61508

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 61508-4:2010, *Sécurité fonctionnelle des systèmes électriques/ électroniques /électroniques programmables relatifs à la sécurité – Partie 4: Définitions et abréviations*

3 Définitions et abréviations

Pour les besoins du présent document, les définitions et abréviations données dans la CEI 61508-4 s'appliquent.

Annexe A
(informative)

Présentation de techniques et mesures pour les systèmes E/E/PE relatifs à la sécurité – maîtrise des défaillances aléatoires du matériel (voir CEI 61508-2)

A.1 Electriques

Objectif général: Maîtriser les défaillances des composants électromécaniques.

A.1.1 Détection des défaillances par surveillance en ligne

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.2, A.3, A.7 et A.13 à A.18 de la CEI 61508-2.

But: Détecter les défaillances en surveillant le comportement du système E/E/PE relatif à la sécurité en réponse à l'exploitation normale (en ligne) de l'équipement commandé (EUC).

Description: Dans certaines conditions, les défaillances peuvent être détectées grâce aux informations concernant (par exemple) le comportement temporel de l'EUC. Par exemple, si un commutateur faisant partie du système E/E/PE relatif à la sécurité est normalement commandé par l'EUC et s'il ne change pas d'état à l'instant voulu, une défaillance est détectée. Il n'est généralement pas possible de localiser la défaillance.

A.1.2 Surveillance des contacts de relais

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.2 et A.14 de la CEI 61508-2.

But: Détecter les défaillances (de soudure par exemple) des contacts de relais.

Description: Les relais à contact forcé (ou à contact dirigé positivement) sont conçus de façon que leurs contacts soient rigidement maintenus ensemble. En supposant qu'il y ait deux ensembles de contacts inverseurs, *a* et *b*, si le contact normalement ouvert, *a*, se soude, le contact normalement fermé, *b*, ne peut pas se fermer lorsque la bobine du relais n'est plus commandée. Par conséquent, la surveillance de la fermeture du contact normalement fermé *b* lorsque la bobine du relais n'est plus commandée, peut servir à prouver que le contact normalement ouvert *a* s'est ouvert. Le défaut de fermeture du contact *b* normalement fermé indique une défaillance du contact *a*; il convient alors que le circuit de surveillance garantisse un arrêt en sécurité ou que l'arrêt en sécurité soit maintenu pour toute machine commandée par le contact *a*.

Références:

Zusammenstellung und Bewertung elektromechanischer Sicherheitsschaltungen für Verriegelungseinrichtungen. F. Kreutzkampf, W. Hertel, Sicherheitstechnisches Informations- und Arbeitsblatt 330212, BIA-Handbuch. 17. Lfg. X/91, Erich Schmidt Verlag, Bielefeld
www.BGIA-HANDBUCHdigital.de/330212

A.1.3 Comparateur

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.2, A.3 et A.4 de la CEI 61508-2.

But: Détecter, le plus tôt possible, les défaillances (non simultanées) dans une unité de traitement autonome ou dans le comparateur.

Description: Les signaux d'unités de traitement autonomes sont comparés de façon cyclique ou continue par un comparateur matériel. Le comparateur peut lui-même être soumis à l'essai

par un appareil externe, ou utiliser une technologie d'auto-surveillance automatique. Les différences détectées dans le comportement des processeurs génèrent un message de défaillance.

A.1.4 Voteur majoritaire

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.2, A.3 et A.4 de la CEI 61508-2.

But: Détecter et masquer les défaillances dans au moins l'un des trois canaux matériels.

Description: Un dispositif de vote majoritaire (2 sur 3, 3 sur 3, ou *m* sur *n*) est utilisé pour détecter et masquer les défaillances. Le voteur peut lui-même être soumis à l'essai par un appareil externe ou utiliser une technologie d'auto-surveillance automatique.

Références:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.1.5 Principe du courant de repos

NOTE Cette technique/mesure est mentionnée dans le Tableau A.16 de la CEI 61508-2.

But: Assurer la fonction de sécurité lorsque l'alimentation est coupée ou perdue.

Description: La fonction de sécurité est assurée si les contacts sont ouverts et le courant ne passe pas. Par exemple, si les freins sont utilisés pour arrêter un mouvement dangereux d'un moteur, les freins sont desserrés en fermant les contacts dans le système relatif à la sécurité et serrés en ouvrant ces mêmes contacts dans le système relatif à la sécurité.

Référence:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.2 Electroniques

Objectif général: Maîtriser les défaillances dans les composants à semi-conducteurs.

A.2.1 Essais par un matériel redondant

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.3, A.15, A.16 et A.18 de la CEI 61508-2.

But: Détecter les défaillances à l'aide de matériel redondant, c'est-à-dire en utilisant du matériel supplémentaire non nécessaire pour l'exécution des fonctions du processus.

Description: Le matériel redondant peut être utilisé pour soumettre à essai à une fréquence adéquate les fonctions de sécurité spécifiées. Cette approche est normalement nécessaire pour réaliser A.1.1 ou A.2.2.

A.2.2 Principes dynamiques

NOTE Cette technique/mesure est mentionnée dans le Tableau A.3 de la CEI 61508-2.

But: Détecter les défaillances statiques par un traitement dynamique des signaux.

Description: Un changement forcé de signaux normalement statiques (générés à l'intérieur ou à l'extérieur) aide à la détection des défaillances statiques des composants. Cette technique est souvent associée aux composants électromécaniques.

Référence:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld, 1993. http://www.bgia-handbuchdigital.de/330220

A.2.3 Port d'accès d'essai normalisé et architecture d'essai du type «registre à décalage périphérique (scan path)"

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.3, A.15 et A.18 de la CEI 61508-2.

But: Surveiller et observer ce qui se passe à chaque broche d'un circuit intégré.

Description: L'essai du type registre à décalage périphérique (scan path) est une technique de conception de circuits intégrés qui accroît la possibilité de soumettre à essai le circuit intégré en résolvant le problème d'accès aux points d'essai situés à l'intérieur de celui-ci. Dans un circuit intégré typique avec un registre à décalage périphérique, composé d'un noyau logique et de registres buffers d'entrée/sortie, un registre à décalage est placé entre le noyau logique et les registres buffers d'entrée/sortie adjacents à chaque broche du circuit intégré. Chaque registre à décalage est situé dans une cellule du registre à décalage périphérique. La cellule du registre à décalage périphérique peut surveiller et observer ce qui se passe à chaque broche d'entrée et de sortie d'un circuit intégré via le port d'accès pour les essais normalisés. La surveillance interne du noyau logique du circuit intégré est effectuée en isolant le noyau logique des stimuli reçus des composants alentours et en effectuant ensuite un essai automatique interne. Ces essais peuvent être utilisés pour détecter les défaillances dans le circuit intégré.

Référence:

IEEE 1149-1:2001, *IEEE standard test access port and boundary-scan architecture, IEEE Computer Society*, 2001, ISBN: 0-7381-2944-5

A.2.4 (Vacant)

A.2.5 Redondance surveillée

NOTE Cette technique/mesure est mentionnée dans le Tableau A.3 de la CEI 61508-2.

But: Détecter les défaillances, en fournissant plusieurs unités fonctionnelles, en surveillant le comportement de chacune de celles-ci pour en détecter les défaillances et en initiant une transition vers un état de sécurité si un écart de comportement est détecté.

Description: La fonction de sécurité est assurée par au moins deux canaux matériels. Les sorties de ces canaux sont surveillées et un état de sécurité est instauré si une anomalie est détectée (c'est-à-dire si les signaux de sortie de tous les canaux ne sont pas les mêmes).

Références:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld, 1993. http://www.bgia-handbuchdigital.de/330220

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

A.2.6 Composants électriques/électroniques avec contrôle automatique

NOTE Cette technique/mesure est mentionnée dans le Tableau A.3 de la CEI 61508-2.

But: Détecter les anomalies par des contrôles périodiques des fonctions de sécurité.

Description: Le matériel est soumis à l'essai avant de lancer le processus et est soumis à l'essai de façon répétée à des intervalles réguliers. L'EUC ne continue à fonctionner que si chaque essai est réussi.

Références:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch, Erich-Schmidt Verlag, Bielefeld, 1993. http://www.bgia-handbuchdigital.de/330220

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

A.2.7 Surveillance du signal analogique

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.3 et A.13 de la CEI 61508-2.

But: Améliorer la fiabilité des signaux mesurés.

Description: Toutes les fois qu'il y a un choix à effectuer, les signaux analogiques sont utilisés de préférence aux états numériques tout ou rien. Par exemple, la disjonction ou les états de sécurité sont représentés par des niveaux de signaux analogiques, généralement avec une surveillance de tolérance du niveau du signal. La technique offre la continuité de la surveillance et un meilleur niveau de confiance dans le transmetteur, réduisant ainsi la fréquence de l'essai périodique nécessaire de la fonction de capteur du transmetteur. Les interfaces externes, par exemple les lignes d'impulsions, nécessitent également un essai.

A.2.8 Déclassement

NOTE Cette technique/mesure est mentionnée en 7.4.2.13 de la CEI 61508-2.

But: Augmenter la fiabilité des composants matériels.

Description: Les composants matériels sont utilisés à des niveaux dont la conception du système garantit qu'ils sont bien au-dessous des caractéristiques maximales des spécifications. Le déclassement est la pratique qui garantit que, dans toutes les conditions de fonctionnement normales, les composants sont utilisés bien en dessous de leurs niveaux de contrainte maximale.

A.3 Unités de traitement

Objectif général: Reconnaître les défaillances qui sont à l'origine des résultats incorrects dans les unités de traitement.

A.3.1 Autotest logiciel – nombre limité de trames (un canal)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.4 de la CEI 61508-2.

But: Détecter le plus tôt possible les défaillances de l'unité de traitement.

Description: Le matériel est construit avec les techniques standards qui ne prennent en compte aucune exigence spéciale relative à la sécurité. La détection de défaillances est entièrement effectuée par des fonctions de logiciel supplémentaires qui effectuent des autotests en utilisant au moins deux configurations (trames) de données complémentaires (par exemple 55hex et AAhex).

A.3.2 Autotest logiciel – walking bit (un canal)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.4 de la CEI 61508-2.

But: Détecter le plus tôt possible les défaillances de la mémoire physique (par exemple les registres) et du décodeur d'instructions de l'unité de traitement.

Description: La détection de défaillances est entièrement réalisée par des fonctions de logiciel supplémentaires qui effectuent des autotests en utilisant une configuration de données (par exemple "walking bit pattern") qui soumet à essai le moyen de stockage physique (registres de données et d'adresse) et le décodeur d'instructions. Toutefois, la couverture du diagnostic n'est que de 90 %.

A.3.3 Autotest pris en charge par le matériel (un canal)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.4 de la CEI 61508-2.

But: Détecter le plus tôt possible les défaillances de l'unité de traitement avec un matériel spécial qui augmente la vitesse et élargit le champ d'application de la détection de défaillances.

Description: Des installations de matériel spécial supplémentaire prennent en charge les fonctions d'autotest pour la détection de défaillances. Cela pourrait être par exemple une unité de matériel qui surveille de façon cyclique la sortie d'une certaine configuration de bit selon le principe du chien de garde.

A.3.4 Traitement codé (un canal)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.4 de la CEI 61508-2.

But: Détecter le plus tôt possible les défaillances de l'unité de traitement.

Description: Les unités de traitement peuvent être conçues avec des techniques de circuits spéciales reconnaissant ou corrigeant les défaillances. A ce jour, ces techniques ne sont appliquées qu'à des circuits relativement simples et ne sont pas très répandues; il convient toutefois de ne pas exclure des développements futurs.

Références:

Le processeur codé: un nouveau concept appliqué à la sécurité des systèmes de transports. Gabriel, Martin, Wartski, Revue Générale des chemins de fer, No. 6, juin 1990

Vital Coded Microprocessor Principles and Application for Various Transit Systems. P. Forin, IFAC Control Computers Communications in Transportation, 79-84, 1989

A.3.5 Comparaison réciproque par logiciel

NOTE Cette technique/mesure est mentionnée dans le Tableau A.4 de la CEI 61508-2.

But: Détecter le plus tôt possible les défaillances de l'unité de traitement par comparaison de logiciels dynamiques.

Description: Deux unités de traitement se transmettent réciproquement les données (y compris les résultats, les résultats intermédiaires et les données d'essai). Une comparaison des données est effectuée à l'aide d'un logiciel dans chaque unité et les différences détectées génèrent un message de défaillance.

A.4 Plages de mémoire invariable

Objectif général: La détection des modifications des informations dans la mémoire invariable.

A.4.1 Redondance multi-bits à sauvegarde de mot (par exemple, surveillance de la ROM avec un code de Hamming modifié)

NOTE 1 Cette technique/mesure est mentionnée dans le Tableau A.5 de la CEI 61508-2.

NOTE 2 Voir également A.5.6 « Surveillance de la RAM avec un code de Hamming modifié ou détection de la défaillance des données par des codes de détection d'erreur (EDC) » et C.3.2 « Codes de détection et correction d'erreur ».

But: Détecter toutes les défaillances d'un seul bit, toutes les défaillances de deux bits, quelques défaillances de trois bits et quelques défaillances de tous les bits dans un mot de 16 bits.

Description: Chaque mot de mémoire est allongé par plusieurs bits redondants pour produire un code de Hamming modifié avec une distance de Hamming d'au moins 4. Chaque fois qu'un mot est lu, la vérification des bits redondants peut déterminer si une altération a eu lieu ou non. S'il y a une différence, un message de défaillance est produit. La procédure peut aussi servir à détecter les défaillances d'adressage en calculant les bits redondants pour la concaténation du mot porteur d'information et son adresse.

Références:

Prüfbare und korrigierbare Codes. W. W. Peterson, München, Oldenburg, 1967

Error detecting and error correcting codes. R. W. Hamming, The Bell System Technical Journal 29 (2), 147-160, 1950

A.4.2 Somme de contrôle modifiée

NOTE Cette technique/mesure est mentionnée dans le Tableau A.5 de la CEI 61508-2.

But: Détecter toutes les défaillances des bits impairs, soit environ 50 % de toutes les défaillances de bits possibles.

Description: Une somme de contrôle est créée par un algorithme adéquat qui utilise tous les mots dans un bloc de mémoire. La somme de contrôle peut être stockée comme un mot supplémentaire dans la ROM ou un mot supplémentaire peut être ajouté au bloc de mémoire pour être sûr que l'algorithme de la somme de contrôle produit une valeur prédéterminée. Dans un essai de mémoire ultérieur, une somme de contrôle est recréée avec le même algorithme et le résultat est comparé avec la valeur stockée ou définie. S'il y a une différence, un message de défaillance est produit.

A.4.3 Signature d'un seul mot (8 bits)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.5 de la CEI 61508-2.

But: Détecter toutes les défaillances d'un seul bit et toutes les défaillances multi-bits dans un mot, ainsi qu'environ 99,6 % de toutes les défaillances de bits possibles.

Description: Le contenu d'un bloc de mémoire est comprimé (soit par voie matérielle, soit par voie logicielle) à l'aide d'un algorithme de contrôle cyclique par redondance (CRC) en un seul mot mémoire. Un algorithme CRC typique traite tout le contenu du bloc comme un flux de données en série par octet ou par bit sur lequel une division polynomiale continue est effectuée à l'aide d'un générateur polynomial. Le reste de la division représente le contenu comprimé de la mémoire – il s'agit de la «signature» de la mémoire – et est stocké. La signature est recalculée lors d'essais ultérieurs et comparée avec celle déjà stockée. Un message de défaillance est produit s'il y a une différence.

A.4.4 Signature d'un mot double (16 bits)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.5 de la CEI 61508-2.

But: Détecter toutes les défaillances d'un seul bit et toutes les défaillances multi-bits dans un mot, ainsi qu'environ 99,998 % de toutes les défaillances de bits possibles.

Description: Cette procédure calcule une signature à l'aide d'un algorithme de contrôle cyclique par redondance (CRC), mais la valeur obtenue a une taille d'au moins deux mots. La signature allongée est stockée, recalculée et comparée comme pour le cas d'un seul mot. Un message de défaillance est produit s'il y a une différence entre les signatures stockées et recalculées.

A.4.5 Réplication de bloc (par exemple, double ROM avec comparaison par matériel ou logiciel)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.5 de la CEI 61508-2.

But: Détecter toutes les défaillances de bits.

Description: L'espace adresse est dupliqué en deux mémoires. La première mémoire est normalement activée. La deuxième mémoire contient les mêmes informations et est accessible en parallèle avec la première Les sorties sont comparées et un message de défaillance est produit si une différence est détectée. Pour détecter certains types d'erreurs binaires, les données doivent être stockées inversées dans l'une des deux mémoires et inversées à nouveau lors de la lecture.

A.5 Plages de mémoire variable

Objectif général: Détecter les défaillances lors de l'adressage, de l'écriture, du stockage et de la lecture.

NOTE Les erreurs intermittentes sont énumérées dans le Tableau A.1 de la CEI 61508-2 comme anomalies à détecter en cours de fonctionnement ou à analyser dans la déduction de la proportion de défaillances en sécurité. Les sources d'erreurs intermittentes sont les suivantes: (1) particules alpha d'une désintégration de paquets, (2) neutrons, (3) bruit EMI externe et (4) diaphonie interne. Le bruit EMI externe est couvert par d'autres exigences de la présente norme internationale.

Il convient que des mesures d'intégrité de sécurité appliquées pendant le temps d'exécution maîtrisent l'effet des particules alpha et des neutrons. Les mesures d'intégrité de sécurité efficaces pour les erreurs récurrentes peuvent ne pas se révéler efficaces pour les erreurs intermittentes, par exemple les essais RAM, tels que « walk-path », « galpat », etc. ne sont pas efficaces, tandis que les techniques de surveillance telles que la parité et le code ECC avec lecture récurrente des cellules de mémoire le sont.

Une erreur intermittente se produit lorsqu'un événement de rayonnement génère une perturbation de charge suffisante pour inverser ou basculer l'état de données d'une cellule de mémoire à semi-conducteur à faible énergie, d'un registre, d'un verrou ou d'un circuit bistable. L'erreur est qualifiée d'intermittente dans la mesure où le circuit proprement dit ne fait pas l'objet d'un endommagement permanent par le rayonnement. Les erreurs intermittentes sont classées en perturbations à un seul bit (SBU)[1] ou en perturbations isolées (SEU)[2] et en perturbations multi-bits (MBU)[3].

Si le circuit perturbé est un élément de stockage tel qu'une cellule de mémoire ou un circuit bistable, l'état est stocké jusqu'à l'opération d'écriture (prévue) suivante. Les nouvelles données feront l'objet d'un stockage approprié. Dans un circuit combiné, l'effet qui se produit est plutôt un régime transitoire, étant donné qu'un flux d'énergie continu s'écoule du composant qui commande ce nœud. L'effet produit pourrait également être un régime transitoire sur les fils de connexion et les lignes de communication. Cependant, en raison de la plus grande capacité, l'effet produit par les particules alpha et les neutrons est considéré négligeable.

Les erreurs intermittentes peuvent concerner tout type de mémoire variable, c'est-à-dire DRAM, SRAM, banques de registre en µP, mémoire cache, pipelines, registres de configuration de dispositifs tels que ADC, DMA, MMU, contrôleur d'interruption, temporisateurs complexes. La sensibilité aux particules alpha et neutrons est fonction à la fois de la tension et de la géométrie du noyau. Des géométries de plus petites dimensions à une tension de noyau de 2,5 V et plus particulièrement en dessous de 1,8 V nécessiteraient une évaluation plus approfondie et l'adoption de mesures de protection plus efficaces.

---

1 SBU = *Single Bit Upset*.

2 SEU = *Single Event Upset*.

3 MBU = *Multi Bits Upset*.

Le taux d'erreurs intermittentes a été indiqué (voir a) et i) ci-dessous) comme se situant dans une plage comprise entre 700 Fit/MBit et 1 200 Fit/MBit pour les mémoires (emboîtées). Il s'agit d'une valeur de référence à comparer avec les données provenant du procédé silicium utilisé pour la mise en œuvre du dispositif. Il y a peu, les SBU étaient considérées comme les plus touchées, mais les tendances les plus récentes (voir a) ci-dessous) font état d'un pourcentage croissant des MBU dans les taux d'erreurs intermittentes globaux (SER) pour les technologies qui utilisent des fréquences inférieures ou égales à 65 nm.

Les ouvrages et sources documentaires suivants fournissent des détails concernant les erreurs intermittentes:

a) Altitude SEE Test European Platform (ASTEP) and First Results in CMOS 130 nm SRAM. J-L. Autran, P. Roche, C. Sudre et al. Nuclear Science, IEEE Transactions on Volume 54, Issue 4, Aug. 2007, pp.1002 - 1009

b) Radiation-Induced Soft Errors in Advanced Semiconductor Technologies, Robert C Baumann, Fellow, IEEE, IEEE transactions on device and materials reliability, vol 5, no. 3, september 2005

c) Soft errors' impact on system reliability, Ritesh Mastipuram and Edwin C Wee, Cypress Semiconductor, 2004

d) Trends And Challenges In VLSI Circuit Reliability, C. Costantinescu, Intel, 2003, IEEE Computer Society

e) Basic mechanisms and modeling of single-event upset in digital microelectronics, P. E. Dodd and L. W. Massengill, IEEE Trans. Nucl. Sci., vol. 50, no. 3, pp . 583–602, Jun. 2003

f) Destructive single-event effects in semiconductor devices and ICs, F. W. Sexton, IEEE Trans. Nucl. Sci., vol. 50, no. 3, pp . 603–621, Jun. 2003

g) Coming Challenges in Microarchitecture and Architecture, Ronen, Mendelson, Proceedings of the IEEE, Volume 89, Issue 3, Mar 2001, pp. 325 – 340

h) Scaling and Technology Issues for Soft Error Rates, A Johnston, 4th Annual Research Conference on Reliability Stanford University, October 2000

i) International Technology Roadmap for Semiconductors (ITRS), plusieurs documents

A.5.1 Essai RAM «échiquier» ou «défilement»

NOTE Cette technique/mesure est mentionnée dans le Tableau A.6 de la CEI 61508-2.

But: Détecter les défaillances de bit à prédominance statique.

Description: Une structure de type échiquier de 0 s et de 1 s est inscrite dans les cellules d'une mémoire binaire. Les cellules sont alors contrôlées par paires pour assurer que les contenus sont identiques et corrects. L'adresse de la première cellule d'une paire de ce type est variable et l'adresse de la deuxième cellule de cette paire est formée par l'inversion bit par bit de la première adresse. Dans un premier temps, la plage d'adresses de la mémoire est déroulée vers les adresses supérieures à partir de l'adresse variable et, dans un deuxième temps, elle est déroulée vers les adresses inférieures. Les deux déroulements sont ensuite répétés avec une distribution préalable inversée. Un message de défaillance est produit s'il y a une différence.

Dans le «défilement» d'un essai RAM, les cellules d'une mémoire binaire sont initialisées par un train binaire uniforme. Dans un premier déroulement, les cellules sont contrôlées vers le haut: le contenu de chaque cellule est contrôlé et les contenus sont ensuite inversés. La base, qui est créée lors du premier déroulement, est traitée dans un deuxième déroulement vers le bas et de la même façon. Les deux premiers déroulements sont répétés avec une distribution préalable inversée dans un troisième ou quatrième déroulement. Un message de défaillance est produit s'il y a une différence.

A.5.2 Essai RAM «walkpath »

NOTE Cette technique/mesure est mentionnée dans le Tableau A.6 de la CEI 61508-2.

But: Détecter les défaillances binaires statiques et dynamiques et les interactions entre cellules mémoire.

Description: La plage de mémoire à soumettre à essai est initialisée par un train binaire uniforme. La première cellule est ensuite inversée et la zone mémoire restante est contrôlée pour assurer que son contenu est correct. Après cela, la première cellule est inversée à nouveau pour retrouver sa valeur initiale et toute la procédure est répétée pour la cellule suivante. Un deuxième déroulement du «modèle du bit errant» est effectué avec une

distribution préalable inverse du contenu. Un message de défaillance est produit s'il y a une différence.

A.5.3 Essai RAM "galpat" ou "galpat transparent"

NOTE Cette technique/mesure est mentionnée dans le Tableau A.6 de la CEI 61508-2.

But: Détecter les défaillances binaires statiques et une grande partie des couplages dynamiques.

Description: Dans l'essai RAM «galpat», la plage de mémoire choisie est d'abord initialisée uniformément (c'est-à-dire tout en 0 s ou tout en 1 s). La première cellule mémoire à soumettre à essai est ensuite inversée et toutes les cellules restantes sont contrôlées pour assurer que leur contenu est correct. Après chaque accès en lecture à l'une des cellules restantes, la cellule inversée est aussi contrôlée. Cette procédure est répétée pour chaque cellule dans la plage de mémoire choisie. Un deuxième déroulement est effectué avec l'initialisation opposée. Toute différence génère un message de défaillance.

L'essai «galpat transparent » est une variante de la procédure ci-dessus: au lieu d'initialiser toutes les cellules dans la plage de mémoire choisie, les contenus existants demeurent inchangés et les signatures sont utilisées pour comparer les contenus des ensembles de cellules. La première cellule à soumettre à essai dans la plage choisie est sélectionnée et la signature S1 de toutes les cellules restantes dans la plage est calculée et enregistrée. La cellule à soumettre à essai est ensuite inversée et la signature S2 de toutes les cellules restantes est recalculée. (Après chaque accès en lecture à l'une des cellules restantes, la cellule inversée est aussi contrôlée). S2 est comparée avec S1 et toute différence génère un message de défaillance. La cellule soumise à essai est à nouveau inversée pour rétablir le contenu de départ et la signature S3 de toutes les cellules restantes est recalculée et comparée avec S1. Toute différence génère un message de défaillance. Toutes les cellules mémoire dans la plage choisie sont contrôlées de la même façon.

A.5.4 Essai RAM «Abraham»

NOTE Cette technique/mesure est mentionnée dans le Tableau A.6 de la CEI 61508-2.

But: Détecter toutes les défaillances de collage et de couplage entre les cellules mémoire.

Description: La proportion d'anomalies détectées dépasse celle de l'essai RAM «galpat». Le nombre d'opérations requis pour effectuer l'essai de toute la mémoire est d'environ 30 *n*, *n* étant le nombre de cellules dans la mémoire. L'utilisation de l'essai peut être rendue transparente pendant le cycle d'exploitation en découpant la mémoire et en soumettant à essai chaque découpage dans des segments temporels différents.

Référence:

Efficient Algorithms for Testing Semiconductor Random-Access Memories. R. Nair, S. M. Thatte, J. A. Abraham, IEEE Trans. Comput. C-27 (6), 572-576, 1978

A.5.5 Redondance à un bit (par exemple, surveillance de la RAM avec un bit de parité)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.6 de la CEI 61508-2.

But: Détecter 50 % de toutes les défaillances binaires possibles dans la plage de mémoire soumise à essai.

Description: Chaque mot de la mémoire est allongé d'un bit (le bit de parité) qui complète chaque mot pour avoir un nombre pair ou impair de niveaux logiques 1 s. La parité du mot porteur d'information est contrôlée chaque fois qu'il est lu. Si un mauvais nombre de 1 s est trouvé, un message de défaillance est produit. Il convient que le choix d'une parité paire ou

impaire soit tel que le plus défavorable des mots zéro (que des 0) et un (que des 1) en cas de défaillance, ne soit pas un code valide. La parité peut aussi servir à détecter des défaillances d'adressage lorsque la parité est calculée pour la concaténation du mot porteur d'information et de son adresse.

A.5.6 Surveillance de la RAM avec un code de Hamming modifié, ou détection de la défaillance des données par des codes de détection d'erreurs (EDC)

NOTE 1 Cette technique/mesure est mentionnée dans le Tableau A.6 de la CEI 61508-2.

NOTE 2 Voir également A.4.1 « Redondance multi-bits à sauvegarde de mot (par exemple surveillance de la RAM avec un code de Hamming modifié) » et C.3.2 « codes de détection et correction d'erreur ».

But: Détecter toutes les défaillances binaires impaires, toutes les défaillances de deux bits, quelques défaillances de trois bits et multi-bits.

Description: Chaque mot de mémoire est allongé par plusieurs bits redondants pour produire un code de Hamming modifié avec une distance de Hamming d'au moins 4. Chaque fois qu'un mot est lu, la vérification des bits redondants peut permettre de déterminer si une altération a eu lieu ou non. Si une différence est détectée, un message de défaillance est produit. La procédure peut aussi servir à détecter une défaillance d'adressage lorsque les bits redondants sont calculés pour la concaténation du mot porteur d'information et de son adresse.

Références:

Prüfbare und korrigierbare Codes. W. W. Peterson, München, Oldenburg, 1967

Error detecting and error correcting codes. R. W. Hamming, The Bell System Technical Journal 29 (2), 147-160, 1950

A.5.7 Double RAM avec comparaison matérielle ou logicielle et essai de lecture/écriture

NOTE Cette technique/mesure est mentionnée dans le Tableau A.6 de la CEI 61508-2.

But: Détecter toutes les défaillances de bits.

Description: L'espace adresse est dupliqué en deux mémoires. La première mémoire est activée normalement. La deuxième mémoire contient les mêmes informations et est accessible en parallèle avec la première. Les sorties sont comparées et un message de défaillance est produit si une différence est détectée. Pour détecter certains types d'erreurs binaires, les données doivent être stockées inversées dans l'une des deux mémoires et inversées à nouveau lors de la lecture.

A.6 Unités E/S et interfaces (communication externe)

Objectif général: Détecter les défaillances des unités d'entrée et de sortie (numérique, analogique, en série ou parallèle) et prévenir l'envoi de sorties inadmissibles au processus.

A.6.1 Trame d'essai

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.7, A.13 et A.14 de la CEI 61508-2.

But: Détecter les défaillances statiques (défaillances dues à un blocage) et les interférences.

Description: Il s'agit d'un essai cyclique indépendant du flux de données des unités d'entrée et de sortie. Il utilise une trame d'essai définie pour comparer les observations avec les valeurs prévues correspondantes. Le contenu, la réception et l'évaluation de la trame d'essai

doivent tous être indépendants les uns des autres. Il convient que l'EUC ne soit pas influencé de façon inadmissible par la trame d'essai.

A.6.2 Protection par code

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.7, A.15, A.16 et A.18 de la CEI 61508-2.

But: Détecter les défaillances systématiques et les défaillances aléatoires du matériel dans le flux de données d'entrée/de sortie.

Description: Cette procédure protège les informations d'entrée et de sortie contre les défaillances systématiques et les défaillances aléatoires du matériel. La protection par code permet une détection des défaillances fonction du flux de données dans les unités d'entrée et de sortie, basée sur la redondance des informations et/ou la redondance temporelle. Typiquement, l'information redondante se superpose aux données d'entrée et/ou de sortie. Il en résulte alors un moyen de surveillance du fonctionnement correct des circuits d'entrée ou de sortie. De nombreuses techniques sont possibles, par exemple un signal à fréquence porteuse peut être superposé au signal de sortie d'un capteur. L'unité de traitement logique peut alors vérifier la présence de la fréquence porteuse ou un code binaire redondant peut être ajouté sur un canal de sortie afin de permettre la surveillance de la validité du passage d'un signal entre l'unité logique et l'actionneur final.

A.6.3 Sortie parallèle multicanal

NOTE Cette technique/mesure est mentionnée dans le Tableau A.7 de la CEI 61508-2.

But: Détecter les défaillances aléatoires du matériel (défaillances dues à un blocage), les défaillances dues à des influences externes, les défaillances de synchronisation, les défaillances d'adressage, les défaillances dues à une dérive et les défaillances transitoires.

Description: Sortie parallèle à plusieurs canaux dépendant du flux de données, avec des sorties indépendantes, qui permet de détecter les défaillances aléatoires du matériel. La détection des défaillances est effectuée par des comparateurs externes. S'il y a une défaillance, l'EUC est arrêté directement. Cette mesure n'est efficace que si le flux de données change pendant l'intervalle d'essai de diagnostic.

A.6.4 Sorties surveillées

NOTE Cette technique/mesure est mentionnée dans le Tableau A.7 de la CEI 61508-2.

But: Détecter les défaillances individuelles, les défaillances dues à des influences externes, les défaillances de synchronisation, les défaillances d'adressage, les défaillances dues à une dérive (pour les signaux analogiques) et les défaillances transitoires.

Description: Comparaison dépendante du flux de données des sorties, avec des entrées indépendantes, qui permet de garantir la conformité avec une plage de tolérances définie (temps, valeur). Une défaillance détectée ne peut pas toujours être mise en relation avec la sortie défectueuse. Cette mesure n'est efficace que si le flux de données change pendant l'intervalle d'essai de diagnostic.

A.6.5 Comparaison/vote majoritaire sur les entrées

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.7 et A.13 de la CEI 61508-2.

But: Détecter les défaillances individuelles, les défaillances dues à des influences externes, les défaillances de synchronisation, les défaillances d'adressage, les défaillances dues à une dérive (pour les signaux analogiques) et les défaillances transitoires.

Description: Comparaison dépendant du flux de données d'entrées indépendantes qui permet de garantir la conformité avec une plage de tolérances définie (temps, valeur). Il y

aura des redondances de 1 parmi 2, de 2 parmi 3 ou plus. Cette mesure n'est efficace que si le flux de données change pendant l'intervalle d'essai de diagnostic.

A.7 Chemins de données (communication interne)

Objectif général: Détecter les défaillances dues à un défaut dans le transfert d'informations.

A.7.1 Redondance matérielle sur un bit

NOTE Cette technique/mesure est mentionnée dans le Tableau A.8 de la CEI 61508-2.

But: Détecter toutes les défaillances binaires impaires, c'est-à-dire 50 % de toutes les défaillances binaires possibles dans le train de données.

Description: Le bus est complété par une ligne (bit) et cette ligne supplémentaire (bit) est utilisée pour détecter les défaillances par un contrôle de parité.

A.7.2 Redondance matérielle sur plusieurs bits

NOTE Cette technique/mesure est mentionnée dans le Tableau A.8 de la CEI 61508-2.

But: Détecter les défaillances pendant la communication sur le bus et dans les liaisons de transmission en série.

Description: Le bus est complété par deux ou plusieurs lignes (bits) et ces lignes supplémentaires (bits) sont utilisées pour détecter les défaillances avec les techniques de code de Hamming.

A.7.3 Redondance matérielle complète

NOTE Cette technique/mesure est mentionnée dans le Tableau A.8 de la CEI 61508-2.

But: Détecter les défaillances pendant la communication en comparant les signaux sur deux bus.

Description: Le bus est doublé et les lignes supplémentaires (bits) sont utilisées pour détecter les défaillances.

A.7.4 Inspection utilisant des trames d'essai

NOTE Cette technique/mesure est mentionnée dans le Tableau A.8 de la CEI 61508-2.

But: Détecter les défaillances statiques (défaillances dues à un blocage) et les interférences.

Description: Il s'agit d'un essai cyclique, indépendant du flux de données, des chemins de données. Il utilise une trame d'essai définie pour comparer les observations avec les valeurs prévues correspondantes.

Le contenu, la réception et l'évaluation de la trame d'essai doivent tous être indépendants les uns des autres. Il convient que l'EUC ne soit pas influencé de façon inadmissible par la trame d'essai.

A.7.5 Redondance de transmission

NOTE Cette technique/mesure est mentionnée dans le Tableau A.8 de la CEI 61508-2.

But: Détecter les défaillances transitoires dans la communication sur le bus.

Description: Les informations sont transférées plusieurs fois en séquence. La répétition n'est efficace que contre les défaillances transitoires.

A.7.6 Redondance d'informations

NOTE Cette technique/mesure est mentionnée dans le Tableau A.8 de la CEI 61508-2.

But: Détecter les défaillances dans la communication sur le bus.

Description: Les données sont transmises en paquets, accompagnées d'une somme de contrôle calculée pour chacun des paquets. Le récepteur calcule à son tour la somme de contrôle correspondant aux données reçues et compare les résultats à la somme de contrôle reçue.

A.8 Alimentation

Objectif général: Détecter ou tolérer les défaillances dues à un défaut de l'alimentation.

A.8.1 Protection contre les surtensions avec arrêt de sécurité

NOTE Cette technique/mesure est mentionnée dans le Tableau A.9 de la CEI 61508-2.

But: Protéger le système relatif à la sécurité contre les surtensions.

Description: Une surtension est détectée assez tôt pour que toutes les sorties puissent être mises en position de sécurité par le sous-programme de mise hors tension ou qu'il y ait un basculement sur une deuxième unité d'alimentation.

Référence:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.8.2 Surveillance de la tension (secondaire)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.9 de la CEI 61508-2.

But: Surveiller les tensions secondaires et mettre sur la position de sécurité si la tension ne se situe pas dans la plage spécifiée.

Description: La tension secondaire est surveillée et une mise hors tension a lieu, ou il y a un basculement sur une deuxième unité d'alimentation si la tension ne se situe pas dans la plage spécifiée.

A.8.3 Mise hors tension avec arrêt de sécurité

NOTE Cette technique/mesure est mentionnée dans le Tableau A.9 de la CEI 61508-2.

But: Couper l'alimentation tout en enregistrant toutes les informations critiques relatives à la sécurité.

Description: La surtension ou la sous-tension est détectée assez tôt pour que l'état interne puisse être sauvegardé dans une mémoire non volatile (si nécessaire), et pour que toutes les sorties puissent être mises en position de sécurité par le sous-programme de mise hors tension, ou pour que toutes les sorties puissent être mises en position de sécurité par le sous-programme de mise hors tension, ou qu'il y ait un basculement sur une deuxième unité d'alimentation.

A.9 Surveillance temporelle et logique de la séquence du programme

NOTE Ce groupe de techniques et mesures est mentionné dans les Tableaux A.15, A.16 et A.18 de la CEI 61508-2.

Objectif général: Détecter une séquence de programme défectueuse. Une séquence de programme est défectueuse si les éléments individuels d'un programme (par exemple modules logiciels, sous-programmes ou commandes) sont traités dans une mauvaise séquence ou période de temps, ou si l'horloge du processeur présente une anomalie.

A.9.1 « Chien de garde » avec base de temps séparée sans fenêtre temporelle

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.10 et A.11 de la CEI 61508-2.

But: Surveiller le comportement et la vraisemblance de la séquence du programme.

Description: Des éléments de séquencement externes avec une base de temps séparée (par exemple des chiens de garde de séquencement) sont périodiquement déclenchés pour surveiller le comportement de l'ordinateur et la vraisemblance de la séquence du programme. Il est important que les points de déclenchement soient correctement placés dans le programme. Le chien de garde n'est pas déclenché à une période fixe, mais un intervalle maximal est spécifié.

A.9.2 « Chien de garde » avec base de temps séparée et fenêtre temporelle

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.10 et A.11 de la CEI 61508-2.

But: Surveiller le comportement et la vraisemblance de la séquence du programme.

Description: Des éléments de séquencement externes avec une base de temps séparée (par exemple des chiens de garde de séquencement) sont périodiquement déclenchés pour surveiller le comportement de l'ordinateur et la vraisemblance de la séquence du programme. Il est important que les points de déclenchement soient correctement placés dans le programme. Une limite inférieure et une limite supérieure sont fixées pour le chien de garde de séquencement. Si la séquence de programme met plus ou moins de temps que prévu, une action d'urgence est entreprise.

A.9.3 Surveillance logique de la séquence du programme

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.10 et A.11 de la CEI 61508-2.

But: Surveiller la bonne séquence des sections individuelles du programme.

Description: La séquence correcte des sections individuelles du programme est surveillée à l'aide d'un logiciel (procédure de comptage, procédure adaptée) ou à l'aide d'éléments de surveillance externes. Il est important que les points de surveillance soient placés correctement dans le programme.

A.9.4 Combinaison de surveillance temporelle et logique des séquences du programme

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.10 et A.11 de la CEI 61508-2.

But: Surveiller le comportement et la bonne séquence des sections individuelles du programme.

Description: Un moyen temporel (par exemple un chien de garde de séquencement) surveillant la séquence du programme n'est redéclenché que si la séquence des sections du programme est également exécutée correctement.

A.9.5 Surveillance temporelle avec contrôle en ligne

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.10 et A.11 de la CEI 61508-2.

But: Détecter les anomalies de la surveillance temporelle.

Description: La surveillance temporelle est contrôlée lors de la mise en marche et celle-ci n'est possible que si la surveillance temporelle fonctionne correctement. Par exemple, une sonde de température pourrait être contrôlée avec une résistance chauffante lors de la mise en marche.

A.10 Ventilation et chauffage

NOTE Ce groupe de techniques et de mesures est mentionné dans les Tableaux A.16 et A.18 de la CEI 61508-2.

Objectif général: Maîtriser les défaillances de la ventilation ou du chauffage, et/ou surveillance de l'aération ou du chauffage s'il s'agit de sécurité.

A.10.1 Capteur de température

But: Détecter une température trop élevée ou trop basse avant que le système ne commence à fonctionner hors spécification.

Description: Un capteur de température surveille la température aux points les plus critiques du système E/E/PE relatif à la sécurité. Avant que la température ne sorte de la plage spécifiée, une action d'urgence est entreprise.

A.10.2 Surveillance des ventilateurs

But: Détecter un mauvais fonctionnement des ventilateurs.

Description: Le bon fonctionnement des ventilateurs est surveillé. Si un ventilateur ne fonctionne pas correctement, une action d'entretien (ou en dernier recours, d'urgence) est entreprise.

A.10.3 Actionnement de l'arrêt de sécurité par l'intermédiaire d'un fusible thermique

But: Arrêter le système relatif à la sécurité avant que le système ne fonctionne en dehors de sa spécification thermique.

Description: Un fusible thermique est utilisé pour arrêter le système relatif à la sécurité. Pour un PES, l'arrêt est produit par un programme de mise hors tension qui enregistre toutes les informations nécessaires pour une action d'urgence.

A.10.4 Message échelonné des capteurs thermiques et de l'alarme conditionnelle

But: Indiquer que le système relatif à la sécurité fonctionne en dehors de sa spécification thermique.

Description: La température est surveillée et une alarme se déclenche si la température est en dehors de la plage spécifiée.

A.10.5 Connexion du refroidissement par air forcé et indication d'état

But: Prévenir la surchauffe par un refroidissement par air forcé.

Description: La température est surveillée et le refroidissement par air forcé est enclenché si celle-ci est plus élevée qu'une valeur limite spécifiée. L'utilisateur est informé de l'état.

A.11 Communication et mémoire de masse

Objectif général: Maîtriser les défaillances pendant la communication avec des sources externes et la mémoire de masse.

A.11.1 Séparation entre les lignes d'alimentation électrique et les lignes de données

NOTE Cette technique/mesure est mentionnée dans le Tableau A.16 de la CEI 61508-2.

But: Minimiser les interférences induites sur les lignes de données par les intensités élevées.

Description: Les lignes d'alimentation électrique sont séparées des lignes acheminant les données. Le champ électrique qui pourrait causer des pointes de tension sur les lignes de données diminue avec la distance.

A.11.2 Séparation spatiale des lignes multiples

NOTE Cette technique/mesure est mentionnée dans le Tableau A.16 de la CEI 61508-2.

But: Minimiser les interférences induites sur les lignes multiples par les intensités élevées.

Description: Les lignes transportant des signaux dupliqués sont séparées les unes des autres. Le champ électrique qui pourrait causer des pointes de tension sur les lignes multiples diminue avec la distance. Cette mesure réduit aussi les défaillances de cause commune.

A.11.3 Augmentation de l'immunité aux interférences

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.16 et A.18 de la CEI 61508-2.

But: Minimiser les interférences électromagnétiques sur le système relatif à la sécurité.

Description: Des techniques de conception telles que le blindage et le filtrage sont utilisées pour accroître l'immunité du système relatif à la sécurité aux perturbations électromagnétiques qui peuvent être rayonnées ou conduites par les lignes d'alimentation ou de signalisation par rayonnement ou conduction ou encore résulter de décharges électrostatiques.

NOTE Voir [16] et [17] pour les exigences d'immunité applicables aux systèmes relatifs à la sécurité et aux équipements destinés à exécuter des fonctions relatives à la sécurité (sécurité fonctionnelle) dans des applications industrielles.

Références:

CEI/TR 61000-5-2:1997, *Compatibilité électromagnétique (CEM) – Partie 5: Guides d'installation et d'atténuation – Section 2: Mise à la terre et câblage*

Principles and Techniques of Electromagnetic Compatibility, Second Edition, C. Christopoulos, CRC Press, 2007, ISBN-10: 0849370353, ISBN-13: 978-0849370359

Noise Reduction Techniques in Electronic Systems. H. W. Ott, John Wiley Interscience, 2nd Edition, 1988

EMC for Product Designers. Tim Williams, Newnes, 2007, ISBN 0750681705

Grounding and Shielding Techniques in Instrumentation, 3rd edition, R. Morrison . Wiley-Interscience, New York, 1986, ISBN-10: 0471838055, ISBN-13: 978-0471838050

A.11.4 Transmission de signaux complémentaires

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.7 et A.16 de la CEI 61508-2.

But: Détecter les mêmes tensions induites dans les lignes de transmission de signaux multiples.

Description: Toutes les informations dupliquées sont transmises par des signaux complémentaires (par exemple en logique positive ou négative). Les défaillances de cause commune (dues par exemple à une émission électromagnétique) peuvent être détectées par un comparateur complémentaire.

Référence:

Elektronik in der Sicherheitstechnik. H. Jürs, D. Reinert, Sicherheitstechnisches Informations- und Arbeitsblatt 330220, BIA-Handbuch. 20. Lfg. V/93, Erich Schmidt Verlag, Bielefeld. http://www.bgia-handbuchdigital.de/330220

A.12 Capteurs

Objectif général: Maîtriser les défaillances des capteurs du système relatif à la sécurité.

A.12.1 Capteur de référence

NOTE Cette technique/mesure est mentionnée dans le Tableau A.13 de la CEI 61508-2.

But: Détecter le mauvais fonctionnement d'un capteur.

Description: Un capteur de référence indépendant est utilisé pour surveiller le fonctionnement d'un capteur de processus. Tous les signaux d'entrée sont contrôlés à des intervalles adéquats par le capteur de référence pour détecter les défaillances du capteur de processus.

Référence:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

A.12.2 Commutateur à action directe

NOTE Cette technique/mesure est mentionnée dans le Tableau A.13 de la CEI 61508-2.

But: Ouvrir un contact par une connexion mécanique directe entre la came et le contact de commutation.

Description: Un commutateur à action directe ouvre ses contacts normalement fermés par une connexion mécanique directe entre la came et le contact de commutation. Cela garantit que, chaque fois que la came de commutation est en position de fonctionnement, les contacts de commutation doivent être ouverts.

Référence:

Verriegelung beweglicher Schutzeinrichtungen. F. Kreutzkampf, K. Becker, Sicherheitstechnisches Informations- und Arbeitsblatt 330210, BIA-Handbuch. 1. Lfg. IX/85, Erich Schmidt Verlag, Bielefeld

A.13 Eléments finaux (actionneurs)

Objectif général: Maîtriser les défaillances des éléments finaux du système relatif à la sécurité.

A.13.1 Surveillance

NOTE Cette technique/mesure est mentionnée dans le Tableau A.14 de la CEI 61508-2.

But: Détecter le mauvais fonctionnement d'un actionneur.

Description: Le fonctionnement de l'actionneur est surveillé (par exemple par les contacts à action directe d'un relais, voir surveillance des contacts de relais en A.1.2). La redondance induite par cette surveillance peut servir à déclencher une action d'urgence.

Références:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

Zusammenstellung und Bewertung elektromechanischer Sicherheitsschaltungen für Verriegelungseinrichtungen. F. Kreutzkampf, W. Hertel, Sicherheitstechnisches Informations- und Arbeitsblatt 330212, BIA-Handbuch. 17. Lfg. X/91, Erich Schmidt Verlag, Bielefeld

A.13.2 Surveillance croisée de plusieurs actionneurs

NOTE Cette technique/mesure est mentionnée dans le Tableau A.14 de la CEI 61508-2.

But: Détecter les anomalies des actionneurs en comparant les résultats.

Description: Chaque actionneur multiple est surveillé par un canal matériel différent. S'il y a un écart, une action d'urgence est entreprise.

A.14 Mesures contre l'environnement physique

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.16 et A.18 de la CEI 61508-2.

But: Prévenir les influences de l'environnement physique (eau, poussière, produits corrosifs) à l'origine de défaillances.

Description: L'enveloppe du matériel est conçue pour résister à l'environnement prévisible.

Référence:

CEI 60529:1989, *Degrés de protection procurés par les enveloppes (Code IP)*

Annexe B
(informative)

Présentation de techniques et mesures pour les systèmes E/E/PE relatifs à la sécurité – prévention des défaillances systématiques (voir la CEI 61508-2 et la CEI 61508-3)

NOTE Beaucoup de techniques citées dans la présente annexe sont applicables au logiciel mais n'ont pas été reproduites dans l'Annexe C.

B.1 Mesures et techniques générales

B.1.1 Gestion de projet

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.1 à B.6 de la CEI 61508-2.

But: Eviter les défaillances par l'adoption d'un modèle organisationnel ainsi que de règles et de mesures pour le développement et l'essai des systèmes relatifs à la sécurité.

Description: Les mesures les plus adaptées et les plus importantes sont:

– la création d'un modèle organisationnel, en particulier pour l'assurance de la qualité qui est exposée dans un manuel sur l'assurance de la qualité; et
– la mise en place de règles et de mesures pour la création et la validation des systèmes relatifs à la sécurité dans les recommandations de projets croisés et les recommandations spécifiques à un projet.

Un certain nombre de principes de base importants sont exposés dans ce qui suit:

– définition d'une organisation de conception:
  – tâches et responsabilités des unités organisationnelles;
  – autorité des services d'assurance de la qualité;
  – indépendance de l'assurance de la qualité (inspection interne) par rapport au développement;
– définition d'un plan séquentiel (modèles d'activité):
  – détermination de toutes les activités pertinentes dans l'exécution du projet, y compris les inspections internes et leur planification;
  – actualisation du projet;
– définition d'une séquence normalisée pour une inspection interne:
  – planification, exécution et contrôle de l'inspection (théorie de l'inspection);
  – création de mécanismes pour les sous-produits;
  – mise en sécurité des inspections répétitives;
– gestion de configuration:
  – administration et contrôle des versions;
  – détection des effets des modifications;
  – inspections de cohérence après les modifications;
– introduction d'une évaluation quantitative des mesures de l'assurance de la qualité:
  – identification des exigences;
  – statistiques des défaillances;
– introduction de méthodes universelles assistées par ordinateur, comprenant les outils et la formation du personnel.

Références:

ISO 9001:2008, *Systèmes de management de la qualité – Exigences*

ISO/CEI 15504 (toutes les parties), *Technologies de l'information – Évaluation des procédés*

CMMI: Guidelines for Process Integration and Product Improvement, 2nd Edition. M. B. Chrissis, M. Konrad, S. Shrum, Addison-Wesley Professional, 2006, ISBN-10: 0-3212-7967-0, ISBN-13: 978-0-3212-7967-5

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

B.1.2 Documentation

NOTE 1 Cette technique/mesure est mentionnée dans les Tableaux B.1 à B.6 de la CEI 61508-2.

NOTE 2 Voir aussi l'Article 5 et l'Annexe A de la CEI 61508-1.

But: Eviter les défaillances et faciliter l'évaluation de la sécurité du système en documentant chaque étape du développement.

Description: L'aptitude à l'exploitation et la sécurité en exploitation, ainsi que le soin apporté au développement par toutes les parties intervenant doivent être démontrés lors de l'évaluation. Une importance particulière est donnée à la documentation afin de pouvoir montrer le soin apporté au développement et pour garantir la vérification des preuves de la sécurité à tout moment.

Des mesures usuelles importantes sont l'introduction de recommandations et d'une assistance par ordinateur, c'est-à-dire

- des recommandations qui:
  - spécifient un plan de groupement;
  - demandent une liste de contrôle pour les contenus; et
  - déterminent la forme du document;
- l'administration de la documentation à l'aide d'une bibliothèque de projet organisée et assistée par ordinateur.

Les mesures individuelles sont:

- la séparation dans la documentation:
  - des exigences;
  - du système (documentation pour l'utilisateur); et
  - du développement (y compris l'inspection interne);
- le groupement de la documentation sur le développement conformément au cycle de vie de sécurité;
- la définition de modules de documentation normalisés à partir desquels les documents peuvent être compilés;
- l'identification claire des parties constitutives de la documentation;
- la mise à jour formalisée des révisions;
- le choix de moyens de description clairs et intelligibles:
  - notation formelle pour les déterminations;

- langage naturel pour les introductions, les justifications et la représentation des intentions;
- représentations graphiques pour les exemples;
- définition sémantique des éléments graphiques; et
- répertoires de mots spécialisés.

Références:

CEI 61506:1997, *Mesure et commande dans les processus industriels – Documentation des logiciels d'application*

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.1.3 Séparation des systèmes E/E/PE relatifs à la sécurité et des systèmes non relatifs à la sécurité

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.1 et B.6 de la CEI 61508-2.

But: Eviter que la partie du système non relative à la sécurité ait une influence néfaste sur la partie relative à la sécurité.

Description: Dans la spécification, il convient de décider si une séparation des systèmes relatifs à la sécurité des systèmes non relatifs à la sécurité est possible. Il convient de rédiger des spécifications claires pour l'interface entre ces deux parties. Une séparation claire réduit l'effort d'essai des systèmes relatifs à la sécurité.

Référence:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.1.4 Diversité du matériel

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.15, A.16 et A.18 de la CEI 61508-2.

But: Détecter les défaillances systématiques pendant le fonctionnement de l'EUC à l'aide de composants diversifiés ayant des taux de défaillance et des types de défaillances différents.

Description: Des types différents de composants sont utilisés pour les différents canaux d'un système relatif à la sécurité. Cela réduit la probabilité des défaillances de cause commune (par exemple surtension, perturbation électromagnétique) et augmente la probabilité de détection de ces défaillances.

L'existence de moyens différents pour effectuer une fonction requise, par exemple des principes physiques différents, offre d'autres façons de résoudre le même problème. Il y a plusieurs types de diversité. La diversité fonctionnelle utilise des approches différentes pour obtenir le même résultat.

Référence:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.2 Spécification des exigences de conception relatives aux systèmes E/E/PE

Objectif général: Produire une spécification qui soit, autant que possible, complète, sans erreur, sans contradiction et simple à vérifier.

B.2.1 Spécification structurée

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.1 et B.6 de la CEI 61508-2.

But: Réduire la complexité en créant une structure hiérarchique des exigences partielles. Eviter les défaillances d'interface entre les exigences.

Description: Cette technique structure la spécification fonctionnelle en exigences partielles de façon qu'il y ait des relations visibles les plus simples possible entre ces exigences. Cette analyse est successivement affinée jusqu'à ce que des petites exigences partielles claires puissent être distinguées. Le résultat de la mise au point finale est une structure hiérarchique d'exigences partielles qui offre un cadre pour la spécification des exigences complètes. Cette méthode met l'accent sur les interfaces des exigences partielles et est particulièrement efficace pour éviter les défaillances d'interfaces.

Références:

ESA PSS 05-02, *Guide to the user requirements definition phase*, Issue 1, Revision 1, ESA Board for Software Standardisation and Control (BSSC), ESA, Paris, March 1995, ftp://ftp.estec.esa.nl/pub/wm/wme/bssc/PSS0502.pdf

Structured Analysis and System Specification. T. De Marco, Yourdon Press, Englewood Cliffs, 1979, ISBN-10: 0138543801, ISBN-13: 978-0138543808

B.2.2 Méthodes formelles

NOTE 1 Voir en C.2.4 les détails sur les méthodes formelles spécifiques.

NOTE 2 Cette technique/mesure est mentionnée dans les Tableaux B.1, B.2 et B.6 de la CEI 61508-2.

But: Les méthodes formelles transfèrent les principes de raisonnement mathématique dans la spécification et la mise en œuvre de systèmes techniques ce qui augmente par conséquent l'exhaustivité, la cohérence ou l'exactitude d'une spécification ou mise en œuvre.

Description: Les méthodes formelles offrent un moyen de développer une description d'un système à une certaine étape de sa spécification et/ou mise en œuvre. Ces descriptions formelles sont des modèles mathématiques de la fonction et/ou de la structure système.

Il est par conséquent possible de réaliser une description de système non ambiguë (par exemple tout état d'un automate est décrit par son état initial, ses entrées et les équations de transition correspondantes) permettant de mieux comprendre le système sous-jacent.

Le choix d'une méthode formelle appropriée est difficile et nécessite de totalement comprendre le système, son processus de développement et la plage de modèles mathématiques qu'il est possible d'utiliser (voir les notes ci-dessous).

NOTE 3 Le théorème pertinent du modèle (propriétés) assure pour le système une plus grande confiance que la simulation, c'est-à-dire en observant les actions sélectionnées du système.

NOTE 4 Les inconvénients des méthodes formelles peuvent comprendre les éléments suivants:

– niveau fixe d'abstraction;
– prise en compte limitée de toutes les fonctionnalités pertinentes à une étape donnée;
– difficulté que rencontrent les ingénieurs chargés de la mise en œuvre à comprendre le modèle;
– efforts soutenus pour développer, analyser et maintenir le modèle tout au long du cycle de vie du système;
– disponibilité d'outils de support efficaces pour construire et analyser le modèle;

– disponibilité d'un personnel capable de développer et d'analyser le modèle.

NOTE 5 Le centre d'intérêt de l'ensemble des méthodes formelles a principalement porté sur la modélisation de la fonction cible du système en négligeant souvent la robustesse d'un système aux anomalies. Par conséquent, des méthodes formelles respectives, y compris la robustesse du système, doivent être sélectionnées.

Référence:

Formal Specification: Techniques and Applications. N.Nissanke, Springer-Verlag Telos, 1999, ISBN-10: 1852330023

B.2.3 Méthodes semi-formelles

NOTE 1 La présente liste est complétée par d'autres techniques semi-formelles relatives aux logiciels dans le Tableau B.7 de la CEI 61508-3, comme suit:

– des diagrammes de bloc logiques/fonctionnels: décrits dans la CEI 61131-3;
– des diagrammes de séquences: décrits dans la CEI 61131-3;
– des diagrammes de flux de données: voir C.2.2;
– des automates finis/diagrammes de transition d'état: voir B.2.3.2;
– des réseaux de Pétri temporels: voir B.2.3.3;
– des modèles de données entité-relation-attribut: voir B.2.4.4;
– des diagrammes de séquences de messages: voir C.2.14;
– des tables de décision/de vérité: voir C.6.1.

But: Enoncer des parties d'une spécification de manière non ambiguë et cohérente pour que certaines erreurs et omissions, ainsi qu'un comportement inadapté, puissent être détectés.

NOTE 2 Cette technique/mesure est mentionnée dans les Tableaux B.1, B.2 et B.6 de la CEI 61508-2, ainsi que dans les Tableaux A.1, A.2, A.4, B.7, C.1, C.2, C.4 et C.17 de la CEI 61508-3.

B.2.3.1 Généralités

But: Prouver que la conception est conforme à sa spécification.

Description: Les méthodes semi-formelles offrent un moyen de développer une description d'un système à une étape de son développement, c'est-à-dire la spécification, la conception ou le codage. La description peut, dans certains cas, être analysée par une machine ou animée afin de visualiser différents aspects du comportement du système. L'animation peut renforcer la confiance en ce que le système satisfait à l'exigence réelle aussi bien qu'à l'exigence spécifiée.

Deux méthodes semi-formelles sont décrites dans les paragraphes suivants.

B.2.3.2 Automates finis/diagrammes de transition d'état

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5, B.7, C.15 et C.17 de la CEI 61508-3.

But: Modéliser, vérifier, spécifier ou mettre en place la structure de commande d'un système.

Description: De nombreux systèmes peuvent être décrits en termes d'états, d'entrées et d'actions. Ainsi, dans l'état S1, à réception d'une entrée I, un système peut exécuter l'action A et passer à l'état S2. En décrivant les actions d'un système pour chaque entrée dans chaque état, il est possible de décrire entièrement un système. Le modèle de système résultant est appelé automate fini. Il est souvent représenté sous forme de «diagramme de transition d'état» montrant comment le système se déplace d'un état à un autre ou sous forme de matrice dont les dimensions sont l'état et l'entrée. Les cellules de la matrice contiennent par ailleurs l'action et le nouvel état qui est le résultat de la réception de l'entrée dans l'état donné.

Lorsqu'un système est compliqué ou a une structure naturelle, cela peut être reflété par un automate fini à couches. Un diagramme d'état (Statechart) est un type de diagramme de

transition d'état qui autorise des états emboîtés (l'état de l'objet se divise en au moins deux sous-états qui peuvent évoluer en parallèle pour éventuellement se reformer en un seul état à un point donné); ceci complète la puissance d'expression de la notation de transition d'état mais peut rendre inutilement plus complexe un système relatif à la sécurité. Les diagrammes d'état disposent d'une spécification (mathématique) formelle. Les diagrammes de transition d'état peuvent s'appliquer à l'ensemble du système ou à certains de leurs objets ou éléments.

Une spécification ou conception exprimée en automate fini peut subir des essais:

– de complétude (le système ou l'objet doit avoir une action et un nouvel état pour chaque entrée dans chaque état);
– de cohérence (un seul changement d'état est possible pour chaque paire état/entrée); et
– d'accessibilité (s'il est possible ou non de passer d'un état à l'autre par toute séquence d'entrées); et
– d'absence de boucles sans fin ou d'états de fin de course; etc.

Ces propriétés sont importantes pour les systèmes critiques. Les outils à l'appui de ces vérifications sont développés facilement, divers modèles basés sur des automates finis (langages formels, réseaux de Pétri, graphiques de Markov, etc.) pouvant par ailleurs être utilisés. Il existe aussi des algorithmes qui permettent la génération automatique des cas d'essai pour vérifier la réalisation d'un automate fini ou pour animer un modèle d'automate fini. Les diagrammes de transition d'état et les diagrammes d'état sont largement pris en charge par des outils permettant de les tracer et de les vérifier, et qui génèrent un code de mise en œuvre de l'automate fini décrit.

Ces algorithmes peuvent également être utilisés pour les calculs de probabilité de défaillance, voir B.6 et C.6.

Référence:

Introduction to Automata Theory, Languages, and Computation (3ème Edition). J. Hopcroft, R. Motwani, J. Ullman, Addison-Wesley Longman Publishing Co, 2006, ISBN:0321462254

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.2.3.3 Réseaux de Pétri temporels

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5, B.7, C.15 et C.17 de la CEI 61508-3.

But: Modéliser les aspects pertinents du comportement du système, évaluer et éventuellement améliorer les exigences de sécurité et d'exploitation par l'analyse et la reconception.

Description: Les réseaux de Pétri représentent un cas particulier d'automates finis. Ils appartiennent à une catégorie de modèles théoriques graphiques qui conviennent pour représenter les informations et le flux de commande dans les systèmes qui ont des traitements simultanés et ont un comportement asynchrone.

Un réseau de Pétri est un réseau de places et de transitions. Les places peuvent être «marquées» ou « non-marquées ». Une transition est «permise» lorsque toutes les places d'entrée au réseau sont marquées. Lorsqu'elle est permise, elle peut (mais n'est pas obligée de) «se déclencher». Si elle se déclenche, les places d'entrée à la transition deviennent non marquées et, à la place, chaque place de sortie de la transition est marquée.

Les dangers potentiels peuvent être représentés comme des états particuliers (marquages) dans le modèle. Le modèle du réseau de Pétri peut être élargi pour permettre des fonctions temporelles du système. Bien que les réseaux de Pétri «classiques» se concentrent sur les

aspects de flux de commande, plusieurs extensions ont été proposées pour incorporer le flux de données dans le modèle.

Ces réseaux constituent également un appui très efficace à l'exécution de la simulation de Monte Carlo afin d'effectuer les calculs de probabilité de défaillance, voir B.6.6.8.

Références:

Timed Petri Nets: Theory and Application. Jiacun Wang, Springer, 1998, ISBN 0792382706

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.2.4 Outils de spécification assistée par ordinateur

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.1 et B.6 de la CEI et 61508-2, ainsi que dans les Tableaux A.1, A.2, C.1 et C.2 de la CEI 61508-3.

B.2.4.1 Généralités

But: Utiliser des techniques de spécification formelles pour faciliter la détection automatique de l'ambiguïté et de la complétude.

Description: Cette technique donne une spécification sous forme d'une base de données qui peut être inspectée automatiquement pour évaluer la cohérence et la complétude. L'outil de spécification peut animer divers aspects du système spécifié pour l'utilisateur. En général, cette technique facilite non seulement la création de la spécification mais aussi la conception et d'autres phases du cycle de vie du projet. Les outils de spécification peuvent être classés selon les paragraphes suivants.

B.2.4.2 Outils orientés vers aucune méthode spécifique

But: Aider l'utilisateur à écrire une spécification valable en fournissant des raccourcis et des liens entre les parties en relation.

Description: L'outil de spécification effectue une partie du travail habituel de l'utilisateur et facilite la gestion du projet. Il n'applique aucune méthodologie de spécification particulière. L'indépendance relative par rapport à la méthode offre aux utilisateurs une grande liberté mais leur apporte également peu de soutien spécialisé nécessaire lors de la création des spécifications. Cela rend la familiarisation avec le système plus difficile.

B.2.4.3 Procédure orientée vers le modèle avec une analyse hiérarchique

But: Eviter toute incomplétude, ambiguïté et contradiction de la spécification, par exemple, aider l'utilisateur à écrire une spécification valable en assurant la cohérence entre la description des actions et des données à différents niveaux d'abstraction.

Description: Cette méthode donne une représentation fonctionnelle du système souhaité (analyse structurée) à des niveaux d'abstraction différents (degré de précision). Il existe un très grand nombre de modèles de ce type: les automates finis constituent une classe des modèles largement utilisés pour décrire l'évolution des systèmes discrets/numériques. Les équations différentielles sont similaires en termes d'esprit et de but aux systèmes continus/analogiques. L'analyse à des niveaux différents influence à la fois les actions et les données. L'évaluation de l'ambiguïté et de la complétude est possible entre les niveaux hiérarchiques ainsi qu'entre deux unités fonctionnelles (modules) sur le même niveau (par exemple, tout état d'un modèle de système est décrit par son état initial, les entrées et les équations de transition de l'automate).

NOTE Les problèmes liés aux descriptions fondées sur le modèle peuvent être les suivants: le niveau d'abstraction, la prise en compte limitée de toutes les fonctionnalités pertinentes à une étape donnée, la difficulté

que rencontrent les intervenants à comprendre le modèle (de la lecture de la syntaxe à sa compréhension), les efforts soutenus nécessaires pour développer, analyser et maintenir un modèle tout au long du cycle de vie d'un système, la disponibilité d'outils de support efficaces pour construire et analyser le modèle (le développement de ce type d'outils représente de toute évidence une activité soutenue) et la disponibilité d'un personnel capable de développer et d'analyser les modèles.

Référence:

System requirements analysis. Jeffrey O. Grady, Academic Press, 2006, ISBN 012088514X, 9780120885145

B.2.4.4 Modèles de données entité-relation-attribut

But: Aider l'utilisateur à écrire une spécification valable en se concentrant sur les entités du système et les relations entre ces entités.

Description: Le système souhaité est décrit comme un ensemble d'objets et de leurs relations. L'outil permet de déterminer quelles relations peuvent être interprétées par le système. En général, les relations permettent une description de la structure hiérarchique des objets, du flux de données, des relations entre les données et des données qui sont soumises à certains procédés de fabrication. La procédure classique a été élargie aux applications de commande de procédés. Les capacités d'inspection et l'assistance à l'utilisateur dépendent de la variété des relations illustrées. D'autre part, un grand nombre de possibilités de représentation rend l'application de cette technique compliquée.

Référence:

Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle. Karl Eugene Wiegers, Microsoft Press, 2003, ISBN 0735618798, 9780735618794

B.2.4.5 Interrogation et réponse

But: Aider l'utilisateur à écrire une spécification valable en identifiant les relations entre les rapports entre les réponses aux différents stimuli.

Description: Les relations entre les objets du système sont spécifiées dans une notation qui rend compte des « interrogations » et des « réponses ». Un langage simple et facilement élargi est utilisé; il contient des éléments de langage qui représentent des objets, des relations, des caractéristiques et des structures.

B.2.5 Listes de contrôle

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.1, B.2 et B.6 de la CEI 61508-2, ainsi que dans les Tableaux A.10, B.8, C.10 et C.18 de la CEI 61508-3.

But: Attirer l'attention et gérer l'évaluation critique de tous les aspects importants du système par une phase du cycle de vie de sécurité assurant une couverture complète sans établir les exigences exactes.

Description: Ensemble de questions auxquelles la personne effectuant la liste de contrôle doit répondre. Nombre de ces questions sont d'ordre général et la personne chargée de l'évaluation doit les interpréter de la façon qui semble la plus appropriée au système particulier évalué. Les listes de contrôle peuvent être utilisées pour toutes les phases du cycle de vie de sécurité global des systèmes E/E/PE et des logiciels, et sont particulièrement utiles comme outil destiné à faciliter l'évaluation de la sécurité fonctionnelle.

Pour être adaptées à une grande diversité des systèmes à valider, la plupart des listes de contrôle comportent des questions applicables à de nombreux types de système. Par conséquent, la liste de contrôle utilisée peut très bien contenir des questions qui sont inadaptées au système considéré et qu'il convient d'ignorer. Il se peut aussi qu'il faille, pour

un système particulier, ajouter à la liste de contrôle standard des questions spécifiques relatives au système traité.

Dans tous les cas, il convient d'indiquer clairement que l'utilisation des listes de contrôle dépend essentiellement de la compétence et du jugement de l'ingénieur choisissant et appliquant la liste de contrôle. Par conséquent, il convient que les décisions prises par l'ingénieur et concernant la ou les listes de contrôle choisies, ainsi que toutes les questions supplémentaires ou superflues, soient entièrement documentées et justifiées. L'objectif est d'assurer que l'application des listes de contrôle peut être examinée et que les mêmes résultats seront obtenus, sauf si des critères différents sont utilisés.

La préparation d'une liste de contrôle nécessite d'être aussi concise que possible. Lorsqu'une justification longue est nécessaire, il convient que celle-ci soit faite par référence à la documentation supplémentaire. Il convient d'utiliser des critères tels que «réussite, échec, non concluant», ou des types de réponses restrictives similaires, pour documenter les résultats de chaque question. Cette précision simplifie beaucoup la procédure pour obtenir une conclusion générale des résultats de l'évaluation de la liste de contrôle.

Références:

CEI 60880:2006, *Centrales nucléaires de puissance – Instrumentation et contrôle-commande importants pour la sûreté – Aspects logiciels des systèmes programmés réalisant des fonctions de catégorie A*

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Software Quality Assurance: From Theory to Implementation. Daniel Galin, Pearson Education, 2004, ISBN 0201709457, 9780201709452

CEI 61346 (toutes les parties), *Systèmes industriels, installations et appareils, et produits industriels – Principes de structuration et désignations de référence*

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824

B.2.6 Inspection de la spécification

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.1 et B.6 de la CEI 61508-2.

But: Eviter l'incomplétude et la contradiction dans la spécification.

Description: L'inspection est une technique générale dans laquelle diverses qualités d'un document de spécification sont évaluées par une équipe indépendante. L'équipe d'inspection remet les questions au créateur qui doit y répondre de façon satisfaisante. Il convient que l'évaluation soit (si possible) effectuée par une équipe qui n'était pas impliquée dans la création de la spécification. Le degré d'indépendance requis est déterminé par les niveaux d'intégrité de sécurité exigés du système. Il convient que les inspecteurs indépendants puissent reconstruire la fonction opérationnelle du système de façon indiscutable sans se référer à d'autres spécifications. Ils doivent également vérifier que tous les aspects de sécurité et techniques pertinents des mesures opérationnelles et organisationnelles sont couverts. Cette procédure s'est avérée très efficace dans la pratique.

Références:

CEI 61160:2005, *Revue de conception*

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Software Quality Assurance: From Theory to Implementation. D. Galin, Pearson Education, 2004, ISBN 0201709457, 9780201709452

B.3 Conception et développement des systèmes E/E/PE

Objectif général: Produire une conception stable du système relatif à la sécurité, conforme à la spécification.

B.3.1 Respect des lignes directrices et des normes

NOTE Cette technique/mesure est mentionnée dans le Tableau B.2 de la CEI 61508-2.

But: Respecter les normes d'application sectorielle (non précisées dans la présente norme).

Description: Il convient que les lignes directrices soient suivies lors de la conception du système relatif à la sécurité. Il convient que ces lignes directrices génèrent d'abord des systèmes relatifs à la sécurité pratiquement sans défaillances et ensuite facilitent la validation de sécurité qui s'ensuit. Elles peuvent être universelles, spécifiques à un projet ou spécifiques à une seule phase.

Références:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.3.2 Conception structurée

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2 et B.6 de la CEI 61508-2.

But: Réduire la complexité en créant une structure hiérarchique d'exigences partielles. Eviter les défaillances d'interface entre les exigences. Simplifier la vérification.

Description: Lors de la conception du matériel, il convient d'utiliser des critères ou des méthodes spécifiques. Par exemple, les éléments suivants peuvent être exigés:

– une conception de circuit hiérarchiquement structurée;
– une utilisation de composants de circuit fabriqués et soumis à essai.

De même, lors de la conception du logiciel, l'utilisation d'organigrammes structurés permet de créer une structure non ambiguë de modules logiciels. Cette structure présente la manière suivant laquelle les différents modules sont en relation, les données précises qui sont échangées entre les modules, et les commandes exactes qui existent entre les modules.

Références:

CEI 61346 (toutes les parties), *Systèmes industriels, installations et appareils, et produits industriels – Principes de structuration et désignations de référence*

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Software Design. D. Budgen, Pearson Education, 2003, ISBN 0201722194, 9780201722192

An Overview of JSD, J. R. Cameron, IEEE Trans SE-12 No. 2, February 1986

Structured Development for Real-Time Systems (3 Volumes). P. T. Yourdon, P. T. Yourdon Press, 1985

Structured Development for Real-Time Systems (3 Volumes). P. T. Ward, S. J. Mellor, Yourdon Press, 1985

Applications and Extensions of SADT. D. T. Ross, Computer, 25-34, April 1985

Essential Systems Analysis. St. M. McMenamin, F. Palmer, Yourdon Inc, 1984

Structured Analysis (SA): A language for communicating ideas. D.T. Ross, IEEE Trans. Software Eng, Vol. SE-3, (1), 16-34

B.3.3 Utilisation d'éléments ayant fait leurs preuves

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2 et B.6 de la CEI 61508-2.

But: Réduire le risque d'anomalies nouvelles et non détectées en utilisant des éléments ayant des caractéristiques spécifiques.

Description: Le choix d'éléments éprouvés est effectué par le fabricant, pour ce qui est de la sécurité, en fonction de la fiabilité de ces éléments (par exemple l'utilisation d'unités physiques soumises à essai en exploitation pour satisfaire aux exigences élevées de sécurité ou le stockage de programmes relatifs à la sécurité dans des mémoires sûres uniquement). La sécurité des mémoires peut se rapporter à un accès non autorisé, à des influences environnementales (compatibilité électromagnétique, radiation, etc.), ainsi qu'à la réaction des éléments en cas de défaillance.

Références:

CEI 61163-1:2006, *Déverminage sous contraintes – Partie 1: Assemblages réparables fabriqués en lots*

B.3.4 Modularisation

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2 et B.6 de la CEI 61508-2.

But: Réduire la complexité et éviter les défaillances liées à l'interfaçage entre les sous-systèmes.

Description: Chaque sous-système, à tous les niveaux de la conception, est clairement défini et est de taille limitée (quelques fonctions uniquement). Les interfaces entre les sous-systèmes sont maintenues aussi simples que possible et les interactions (c'est-à-dire les données communes, l'échange d'informations) sont réduites au minimum. La complexité des sous-systèmes individuels est aussi limitée.

Références:

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Software Reliability – Principles and Practices. G. J. Myers, Wiley-Interscience, New York, 1976, ISBN-10: 0471627658, ISBN-13: 978-0471627654

B.3.5 Outils de conception assistée par ordinateur

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2 et B.6 de la CEI 61508-2 et dans les Tableaux A.4 et C.4 de la CEI 61508-3.

But: Effectuer la procédure de conception de manière plus systématique. Inclure des éléments de construction automatiques appropriés qui sont déjà disponibles et éprouvés.

Description: Il convient que les outils de conception assistée par ordinateur (CAO) soient utilisés pour la conception du matériel et des logiciels lorsqu'ils existent et qu'ils soient justifiés par la complexité du système. Il convient que la précision de ces outils soit démontrée par des essais spécifiques, par un rapport détaillé sur une utilisation satisfaisante ou par une vérification indépendante de leurs résultats pour le système relatif à la sécurité particulier en cours de conception.

Il convient de choisir les outils de support pour leur degré d'intégration. Dans ce contexte, les outils sont intégrés s'ils fonctionnent ensemble de sorte que les sorties d'un outil présentent un contenu et un format appropriés à l'entrée automatique d'un autre outil, réduisant ainsi au minimum la possibilité d'introduire des erreurs humaines lors du remaniement des résultats intermédiaires.

Références:

Overview of Technology Computer-Aided Design Tools and Applications in Technology Development, Manufacturing and Design. W. Fichtner, Journal of Computational and Theoretical Nanoscience, Volume 5, Number 6, June 2008, pp. 1089-1105(17)

The Electromagnetic Data Exchange: Much more than a Common Data Format. P.E. Frandsen et al. In *Proceeding of the 2nd European Conference on Antennas and Propagation.* The Institution of Engineering and Technology (IET), 2007, ISBN 978-0-86341-842-6

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

B.3.6 Simulation

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2, B.5 et B.6 de la CEI 61508-2.

But: Effectuer une inspection complète et systématique d'un circuit électrique/électronique, ainsi que des performances fonctionnelles et du dimensionnement correct des composants.

Description: La fonction du circuit du système relatif à la sécurité est simulée sur ordinateur par un modèle de comportement de logiciel. Les composants individuels du circuit ont chacun leur propre comportement simulé et la réponse du circuit dans lequel ils sont assemblés est étudiée en observant les données marginales aux limites de chaque composant.

B.3.7 Inspection (revues et analyse)

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2 et B.6 de la CEI 61508-2.

But: Révéler les écarts entre la spécification et la réalisation.

Description: Les fonctions spécifiées du système relatif à la sécurité sont étudiées et évaluées pour assurer que le système relatif à la sécurité est conforme aux exigences de la spécification. Tout point de doute concernant la réalisation et l'utilisation du produit est documenté afin qu'il puisse être résolu. Contrairement à un sondage, l'auteur est passif et l'inspecteur est actif pendant la procédure d'inspection.

Références:

CEI 61160:2005, *Revue de conception*

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

ANSI/IEE 1028:1997, *IEEE Standard for software reviews*

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.3.8 Sondage

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2 et B.6 de la CEI 61508-2.

But: Révéler les écarts entre la spécification et la réalisation.

Description: Les fonctions spécifiées du projet du système relatif à la sécurité sont étudiées et évaluées pour assurer que le système relatif à la sécurité satisfait aux exigences exposées dans la spécification. Les doutes et points faibles éventuels concernant la réalisation et l'utilisation du produit sont documentés pour qu'ils puissent être résolus. Contrairement à une inspection, l'auteur est actif et l'inspecteur est passif lors du sondage.

Références:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

ANSI/IEEE 1028:1997, *IEEE Standard for software reviews*

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

Methodisches Testen von Programmen. G. J. Myers, Oldenbourg Verlag, München, Wien, 1987

B.4 Procédures d'exploitation et de maintenance des systèmes E/E/PE

Objectif général: Développer des procédures qui aident à éviter les défaillances au cours de l'exploitation et de la maintenance du système relatif à la sécurité.

B.4.1 Instructions d'exploitation et de maintenance

NOTE Cette technique/mesure est mentionnée dans le Tableau B.4 de la CEI 61508-2.

But: Eviter des erreurs pendant l'exploitation et la maintenance du système relatif à la sécurité.

Description: Les instructions pour l'utilisateur contiennent les informations essentielles sur la façon d'utiliser et de maintenir le système relatif à la sécurité. Dans des cas particuliers, ces instructions comporteront également des exemples sur la façon d'installer le système relatif à la sécurité en général. Toutes les instructions doivent être facilement compréhensibles. Il convient que des figures et des schémas soient utilisés pour décrire les procédures et les interdépendances complexes.

Référence:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.4.2 Facilité d'utilisation

NOTE Cette technique/mesure est mentionnée dans le Tableau B.4 de la CEI 61508-2.

But: Réduire la complexité pendant l'exploitation du système relatif à la sécurité.

Description: La bonne exploitation du système relatif à la sécurité peut dépendre, dans une certaine mesure, de l'intervention humaine. Prenant en compte la conception du système qui convient et la conception du lieu de travail, la personne qui développe le système relatif à la sécurité doit assurer que

- le besoin d'intervention de l'homme est réduit au strict minimum;
- l'intervention nécessaire est aussi simple que possible;
- le risque de dommage potentiel résultant d'erreurs de l'utilisateur est réduit au minimum;
- les moyens d'intervention et les indications sont conçus en fonction d'exigences ergonomiques;
- les moyens mis à la disposition de l'opérateur sont simples, bien repérés et d'utilisation intuitive;
- l'opérateur n'est pas surchargé, même dans les situations extrêmes;
- la formation sur les procédures et les moyens d'intervention est adaptée au niveau de connaissance et de motivation de l'utilisateur qui reçoit la formation.

B.4.3 Facilité de maintenance

NOTE Cette technique/mesure est mentionnée dans le Tableau B.4 de la CEI 61508-2.

But: Simplifier les procédures de maintenance du système relatif à la sécurité et concevoir les moyens nécessaires pour un diagnostic et une réparation efficaces.

Description: La maintenance préventive et la réparation sont souvent effectuées dans des conditions difficiles et sous la pression des délais. Par conséquent, il convient que la personne qui développe le système relatif à la sécurité s'assure que

- les mesures de maintenance relatives à la sécurité sont nécessaires aussi rarement que possible, l'idéal serait qu'elles ne soient pas du tout nécessaires;
- des outils de diagnostic suffisants, sensibles et faciles à manipuler sont inclus pour les réparations inévitables; il convient que les outils comprennent toutes les interfaces nécessaires;
- si des outils de diagnostic séparés doivent être développés ou obtenus, il convient alors qu'ils soient disponibles le moment venu.

B.4.4 Possibilités limitées d'exploitation

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.4 et B.6 de la CEI 61508-2.

But: Réduire les possibilités d'exploitation pour l'utilisateur normal.

Description: Cette approche réduit les possibilités d'exploitation en

– limitant l'exploitation dans des modes d'exploitation spéciaux, par exemple par des interrupteurs à clé;
– limitant le nombre d'éléments en exploitation;
– limitant le nombre de modes d'exploitation généralement possibles.

Référence:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.4.5 Exploitation uniquement par des opérateurs qualifiés

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.4 et B.6 de la CEI 61508-2.

But: Eviter les défaillances en exploitation dues à une mauvaise utilisation.

Description: L'opérateur du système relatif à la sécurité est formé à un niveau correspondant au niveau d'intégrité de complexité et de sécurité du système relatif à la sécurité. La formation comprend l'apprentissage des bases du procédé de production et la connaissance de la relation entre le système relatif à la sécurité et l'EUC.

Référence:

Guidelines for Safe Automation of Chemical Processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.4.6 Protection contre les erreurs humaines

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.4 et B.6 de la CEI 61508-2.

But: Protéger le système contre tous les types d'erreurs de l'opérateur.

Description: Les entrées erronées (valeur, temps, etc.) sont détectées par des contrôles de vraisemblance ou par surveillance de l'EUC. Pour intégrer ces moyens dans la conception, il est nécessaire d'établir très tôt quelles entrées sont possibles et quelles entrées sont autorisées.

B.4.7 (Non utilisé)

B.4.8 Protection contre les modifications

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.17 et A.18 de la CEI 61508-2.

But: Protéger le système relatif à la sécurité contre les modifications du matériel par des moyens techniques.

Description: Les modifications ou manipulations sont automatiquement détectées, par exemple par des contrôles de vraisemblance des signaux des capteurs, une détection par des procédés techniques et par des essais de démarrage automatique. Si une modification est détectée, alors une action d'urgence est entreprise.

B.4.9 Accusé de réception des entrées

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.17 et A.18 de la CEI 61508-2.

But: Une erreur pendant l'exploitation est détectée par l'opérateur lui-même avant d'activer l'EUC.

Description: Une entrée dans l'EUC par le système relatif à la sécurité est renvoyée à l'opérateur avant d'être envoyée à l'EUC de façon que l'opérateur puisse détecter et corriger une erreur. Tout comme les actions personnelles non provoquées et anormales, il convient que la conception du système prenne en compte les limites de vitesse supérieure et inférieure et le sens de la réaction humaine. Cela éviterait, par exemple, que l'opérateur appuie sur les touches plus vite que prévu et que le système prenne deux frappes pour une seule ou qu'une même touche soit frappée deux fois parce que le système (affichage) aura été trop lent à réagir à la première frappe. Il convient que la frappe de la même touche ne soit pas valide plus d'une fois à la suite pour la saisie de données critiques; le fait d'appuyer sur les touches «entrée» ou «oui» un nombre de fois illimité ne doit pas conduire à une action dangereuse du système.

Il convient que des procédures de temporisation soient prévues avec des questions à choix multiples (oui/non, etc.) pour les fois où l'opérateur hésite et laisse le système en attente.

La possibilité de réinitialiser un PES relatif à la sécurité rend le système vulnérable, à moins que le logiciel/matériel ne soient conçus en ayant à l'esprit de telles possibilités.

B.5 Intégration des systèmes E/E/PE

Objectif général: Eviter les défaillances pendant la phase d'intégration et révéler toutes les défaillances qui ont lieu pendant cette phase et pendant la phase précédente.

B.5.1 Essais fonctionnels

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.3 et B.5 de la CEI 61508-2, ainsi que dans les Tableaux A.5, A.6, A.7, C.5, C.6 et C.7 de la CEI 61508-3.

But: Révéler les défaillances pendant les phases de spécification et de conception. Eviter les défaillances lors de la réalisation et de l'intégration du logiciel et du matériel.

Description: Pendant les essais fonctionnels, des revues sont effectuées afin de constater si les caractéristiques spécifiées du système ont été respectées. Le système reçoit les paramètres qui caractérisent correctement le fonctionnement normalement attendu. Les sorties sont observées et leur réaction est comparée avec celle indiquée dans la spécification. Les écarts par rapport à la spécification et les indications d'une spécification incomplète sont documentés.

Les essais fonctionnels des composants électroniques conçus pour une architecture à plusieurs canaux impliquent habituellement que les composants fabriqués soient soumis à essai avec des composants similaires préalablement validés. En outre, il est recommandé de soumettre à essai les composants fabriqués conjointement avec d'autres composants similaires du même lot afin d'identifier les anomalies de mode commun qui autrement seraient restées masquées.

La puissance de travail du système doit également être suffisante, voir recommandation en C.5.20.

Références:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Practical Software Testing: A Process-oriented Approach. I. Burnstein, Springer, 2003, ISBN 0387951318, 9780387951317

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.5.2 Essai « boîte noire »

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.3, B.5 et B.6 de la CEI 61508-2, ainsi que dans les Tableaux A.5, A.6, A.7, C.5, C.6 et C.7 de la CEI 61508-3.

But: Contrôler le comportement dynamique dans des conditions réelles de fonctionnement. Révéler les défaillances pour que la spécification fonctionnelle soit satisfaite et évaluer son utilité et sa robustesse.

Description: Les fonctions d'un système ou d'un programme sont exécutées dans un environnement spécifié avec des données d'essai spécifiées déduites systématiquement de la spécification conformément aux critères établis. Cela met en évidence le comportement du système et permet une comparaison avec la spécification. Aucune connaissance de la structure interne du système n'est utilisée pour diriger l'essai. Le but est de déterminer si l'unité fonctionnelle effectue correctement toutes les fonctions exigées par la spécification. La technique consistant à former des classes d'équivalence constitue un exemple des critères pour définir des données d'essai «boîte noire». L'ensemble des données d'entrée est subdivisé en plages de valeurs d'entrée spécifiques (classes d'équivalence) à l'aide de la spécification. Les cas d'essai sont alors formés à partir

- des données comprises dans les plages permises;
- des données en dehors des plages permises;
- des données aux limites des plages établies;
- des valeurs extrêmes;
- et des combinaisons des classes ci-dessus.

D'autres critères peuvent être efficaces pour choisir les cas d'essai dans les différentes activités d'essai (essai du module, essai d'intégration et essai du système). Par exemple, pour l'essai du système dans le cadre d'une validation, on se fie au critère «conditions d'exploitation extrêmes».

Références:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844

Essentials of Software Engineering. Frank F. Tsui, Orlando Karam. Jones & Bartlett, 2006, ISBN 076373537X, 9780763735371

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Systematic Software Testing. Rick D. Craig, Stefan P. Jaskiel. Artech House, 2002. ISBN 1580535089, 9781580535083

B.5.3 Essais statistiques

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.3, B.5 et B.6 de la CEI 61508-2.

But: Vérifier le comportement dynamique du système relatif à la sécurité et évaluer son utilité et sa robustesse.

Description: Cette approche soumet à essai un système ou un programme avec des données d'entrée choisies en fonction de la distribution statistique attendue des données d'exploitation réelles, c'est–à-dire le profil opérationnel.

Références:

A discussion of statistical testing on a safety-related application. S Kuball, J H R May, Proc. IMechE Vol. 221 Part O: J. Risk and Reliability, Institution of Mechanical Engineers, 2007

Practical Reliability Engineering. P. O'Connor, D. Newton, R. Bromley, John Wiley and Sons, 2002, ISBN 0470844639, 9780470844632

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

B.5.4 Expérience pratique

NOTE 1 Cette technique/mesure est mentionnée dans les Tableaux B.3, B.5 et B.6 de la CEI 61508-2.

NOTE 2 Voir aussi en C.2.10 une mesure similaire et à l'Annexe D une approche statistique, tous deux dans le contexte du logiciel.

But: Utiliser l'expérience pratique de différentes applications comme l'une des mesures permettant d'éviter les anomalies, soit pendant l'intégration de systèmes E/E/PE, et/ou pendant la validation de sécurité desdits systèmes.

Description: Utilisation de composants ou de sous-systèmes dont l'utilisation montre qu'ils n'ont pas d'anomalies, ou alors uniquement des anomalies sans importance, lors d'un fonctionnement pratiquement sans remplacement pendant une période suffisante et dans beaucoup d'applications différentes. La personne qui développe le système doit faire attention aux fonctions qui ont en fait été validées par l'expérience pratique, en particulier pour les composants complexes avec une multitude de fonctions possibles (par exemple le système d'exploitation, les circuits intégrés). Par exemple, considérons les sous-programmes d'autotest pour la détection d'anomalies: s'il n'y a pas de panne du matériel pendant l'exploitation, ces sous-programmes ne peuvent pas être considérés comme validés par l'expérience dans la mesure où ils n'ont jamais effectué leur fonction de détection d'anomalies.

Pour pouvoir appliquer l'expérience pratique, les exigences suivantes doivent être remplies:

– spécification inchangée;
– 10 systèmes dans des applications différentes;
– 10^5 heures de fonctionnement et au moins un an d'historique en service.

NOTE 3 Les normes sectorielles peuvent spécifier des nombres différents.

L'expérience pratique est démontrée par la documentation du fournisseur et/ou de la société d'exploitation. Cette documentation doit contenir au moins

– la désignation exacte du système et de son composant, y compris la vérification de la version du matériel;
– les utilisateurs et le temps d'application;
– le nombre d'heures de fonctionnement;
– les procédures pour le choix des systèmes et applications obtenus pour l'argumentation;
– les procédures pour la détection et l'enregistrement des anomalies ainsi que pour leur correction.

Références:

CEI 60300-3-2:2004, *Gestion de la sûreté de fonctionnement – Partie 3-2: Guide d'application – Recueil de données de sûreté de fonctionnement dans des conditions d'exploitation*

Guidelines for safe automation of chemical processes. CCPS, AIChE, New York, 1993, ISBN-10: 0-8169-0554-1, ISBN-13: 978-0-8169-0554-6

B.6 Validation de la sécurité des systèmes E/E/PE

Objectif général: Prouver que le système E/E/PE relatif à la sécurité est conforme aux spécifications relatives aux exigences de sécurité et de conception des systèmes E/E/PE.

B.6.1 Essais fonctionnels dans des conditions environnementales

NOTE Cette technique/mesure est mentionnée dans le Tableau B.5 de la CEI 61508-2.

But: Evaluer si le système relatif à la sécurité est protégé contre les influences environnementales habituelles.

Description: Le système est soumis à diverses conditions environnementales (conformément aux normes des séries CEI 60068 ou CEI 61000 par exemple) pendant lesquelles la fiabilité des fonctions de sécurité (ainsi que leur compatibilité avec les normes mentionnées) est évaluée.

Références:

CEI 60068-1:1988, *Essais d'environnement – Partie 1: Généralités et guide*
Amendement 1(1992)

CEI 61000-4-1:2006, *Compatibilité électromagnétique (CEM) – Partie 4-1: Techniques d'essai et de mesure – Vue d'ensemble de la série CEI 61000-4*

Dependability of Critical Computer Systems 3. P. G. Bishop et al., Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.2 Essai d'immunité aux interférences/ et aux ondes de choc

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5 et B.6 de la CEI 61508-2.

But: Evaluer la capacité du système relatif à la sécurité à supporter les ondes de choc.

Description: Le système est chargé avec un programme d'application typique et toutes les lignes périphériques (toutes les interfaces numériques, analogiques et série, ainsi que les connexions et l'alimentation du bus, etc.) sont soumises à des signaux parasites normalisés. Afin d'obtenir un état quantitatif, il est recommandé d'approcher prudemment la limite de l'onde de choc. La catégorie de parasites choisie n'est pas respectée si une défaillance apparaît.

Références:

CEI 61000-4-5:2005, *Compatibilité électromagnétique (CEM) – Partie 4-5: Techniques d'essai et de mesure – Essai d'immunité aux ondes de choc*

C37.90.1-2002, *IEEE Standard for Surge Withstand Capability (SWC) Tests for Relays and Relay Systems Associated with Electric Power Apparatus*

B.6.3 (Non utilisé)

B.6.4 Analyse statique

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5 et B.6 de la CEI 61508-2, ainsi que dans les Tableaux A.9, B.8, C.9 et C.18 de la CEI 61508-3.

But: Eviter les anomalies systématiques qui peuvent causer des pannes du système soumis à essai, soit au début soit après des années d'exploitation.

Description: Cette approche systématique et éventuellement assistée par ordinateur inspecte les caractéristiques statiques spécifiques du système prototype pour assurer la complétude, la cohérence et l'absence d'ambiguïté par rapport à l'exigence en question (par exemple recommandations de construction, spécifications du système et fiche technique de l'appareil). Une analyse statique est reproductible. Elle est appliquée à un prototype qui a atteint une étape bien définie de réalisation. Des exemples d'analyse statique pour le matériel et les logiciels sont

- l'analyse de cohérence du flux de données (telle que l'essai consistant en la vérification qu'un objet de donnée est interprété partout comme ayant la même valeur);
- l'analyse du flux de commande (telle que la détermination du chemin d'accès ou de code inaccessible);
- l'analyse de l'interface (telle que la recherche du transfert de variables entre différents modules logiciels);
- l'analyse du flux de données pour détecter les séquences suspectes de création, désignation et suppression de variables;
- la vérification par essai du respect des recommandations spécifiques (par exemple distances dans l'air et lignes de fuite, distance d'assemblage, disposition de l'unité physique, unités physiques mécaniquement sensibles, utilisation exclusive des unités physiques qui ont été mises en place).

Références:

Static Analysis and Software Assurance. D. Wagner, Lecture Notes in Computer Science, Volume 2126/2001, Springer, 2001, ISBN 978-3-540-42314-0

An Industrial Perspective on Static Analysis. B A Wichmann, A A Canning, D L Clutterbuck, L A Winsborrow, N J Ward and D W R Marsh. Software Engineering Journal., 69 – 75, March 1995

Dependability of Critical Computer Systems 3. P. G. Bishop et al.. Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.5 Analyse dynamique et essais

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5 et B.6 de la CEI 61508-2, ainsi que dans les Tableaux A.5, A.9, B.2, C.5, C.9 et C.12 de la CEI 61508-3.

But: Détecter les défaillances de la spécification en inspectant le comportement dynamique d'un prototype à une étape avancée de sa réalisation.

Description: L'analyse dynamique d'un système relatif à la sécurité est effectuée en soumettant un prototype presque opérationnel du système relatif à la sécurité à des données d'entrée typiques de l'environnement d'exploitation prévu. L'analyse est satisfaisante si le comportement observé du système relatif à la sécurité est conforme au comportement requis. Toute défaillance du système relatif à la sécurité doit être corrigée et la nouvelle version opérationnelle doit alors faire l'objet d'une nouvelle analyse.

Références:

The Concept of Dynamic Analysis. T. Ball, ESEC/FSE '99, Lecture Notes in Computer Science, Springer, 1999, ISBN 978-3-540-66538-0

Dependability of Critical Computer Systems 3. P. G. Bishop et al.. Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.6 Analyse de défaillance

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5 et B.6 de la CEI 61508-2.

B.6.6.1 Analyse des modes de défaillance et de leurs effets (AMDE)

But: Analyser une conception du système en étudiant systématiquement toutes les sources possibles de défaillances des composants de ce système et en déterminant les effets de ces défaillances sur le comportement et la sécurité du système.

Description: L'analyse est généralement effectuée lors d'une réunion d'ingénieurs. Les composants d'un système sont analysés l'un après l'autre pour établir l'ensemble des modes de défaillance du composant, leurs causes et leurs effets (au niveau local et au niveau du système général), ainsi que des procédures et recommandations pour la détection. Si les recommandations sont prises en charge, elles sont documentées comme des actions de correction effectivement prises.

Références:

CEI 60812:2006, *Techniques d'analyse de la fiabilité du système – Procédure d'analyse des modes de défaillance et de leurs effets (AMDE)*

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824

Reliability Technology. A. E. Green. A. J. Bourne, Wiley-Interscience, 1972, ISBN 0471324809

B.6.6.2 Diagrammes cause-conséquence

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.3, B.4, C.13 et C.14 de la CEI 61508-3.

But: Analyser et modéliser, sous forme de diagrammes compacts, la séquence des événements qui peut se développer dans un système comme conséquence d'une association d'événements initiaux.

Description: Cette technique peut être considérée comme une association d'analyses par arbre de panne et par arbre d'événement. Elle part d'un événement (d'origine) critique, le diagramme cause-conséquence étant tracé vers l'aval en utilisant les portes OUI/NON décrivant la réussite et l'échec de certaines opérations. Ceci permet de former des séquences d'événements qui génèrent un accident ou une situation maîtrisée. Des diagrammes de cause (c'est-à-dire des arbres de panne) sont établis pour chaque défaillance. L'analyse qui part d'une situation accidentelle et examine les événements de tête génère un arbre de panne général, cette situation accidentelle constituant l'événement de tête. Vers l'aval, les conséquences possibles d'un événement sont définies. Le graphique peut comporter des symboles graphiques qui décrivent les conditions de propagation le long des différentes branches partant du sommet. Des temporisations peuvent aussi être incluses. Ces conditions peuvent aussi être décrites avec des arbres de panne. Les lignes de propagation peuvent être associées à des symboles logiques pour rendre le diagramme plus compact. Un ensemble de symboles standard est défini pour une utilisation dans les diagrammes cause-conséquence. Les diagrammes peuvent être utilisés pour générer les arbres de panne et calculer la probabilité d'occurrence de certaines conséquences critiques. Ils peuvent également être utilisés pour produire des arbres d'événements.

Références:

CEI 62502, *Techniques d'analyse de la sûreté de fonctionnement – Analyse par arbre d'évènement (AAE)*[4]

The Cause Consequence Diagram Method as a Basis for Quantitative Accident Analysis. B. S. Nielsen, Danish Atomic Energy Commission, Riso-M-1374, 1971

B.6.6.3 Analyse par arbre d'événement

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.4 et C.14 de la CEI 61508-3.

But: Modéliser, sous forme graphique, la séquence des événements qui peut se développer dans un système après un événement initiateur et indiquer ainsi comment des conséquences graves peuvent se produire. Un arbre d'événement est difficile à construire à partir de rien; l'utilisation d'un diagramme cause-conséquence est par conséquent utile.

Description: Au sommet du diagramme apparaissent les conditions de la séquence qui sont pertinentes pour la progression des événements suivant l'événement initiateur. Partant sous l'événement initiateur, qui est la cible de l'analyse, une ligne est tracée vers la première condition de la séquence. Là, le diagramme se divise en deux branches, «oui» et «non», décrivant comment les événements futurs dépendent de la condition. Pour chacune de ces branches, on continue vers la condition suivante de la même façon. Toutes les conditions, toutefois, ne conviennent pas à toutes les branches. On continue jusqu'à la fin de la séquence et chaque branche de l'arbre ainsi construit représente une conséquence possible. Sous réserve de l'indépendance des conditions des différentes séquences, l'arbre des événements peut servir à calculer la probabilité des différentes conséquences sur la base de la probabilité et du nombre de conditions dans la séquence. Dans la mesure où les conditions sont rarement pleinement indépendantes, ce type de calcul doit être considéré avec la plus grande attention et effectué par des analystes qualifiés.

Références:

CEI 62502, *Techniques d'analyse de la sûreté de fonctionnement – Analyse par arbre d'évènement (AAE)*[5]

Risk Assessment and Risk Management for the Chemical Process Industry. H.R. Greenberg, J.J. Cramer, John Wiley and Sons, 1991, ISBN 0471288829, 9780471288824

B.6.6.4 Analyse des modes de défaillance, de leurs effets et de leur criticité (AMDEC)

NOTE L'analyse des défaillances est mentionnée dans les Tableaux A.10, B.4, C.10 et C.14 de la CEI 61508-3.

But: Etablir un ordre de criticité des composants qui pourraient être à l'origine d'une blessure, d'un préjudice ou d'une dégradation du système par le biais de défaillances uniques, afin de déterminer quels composants peuvent nécessiter une attention particulière et des mesures de surveillance nécessaires pendant la conception ou l'exploitation.

Description: Cette méthode est comparable à la AMDE, mais comporte toutefois une ou plusieurs colonnes d'indication de la criticité, qui peut être classée de plusieurs façons. La méthode la plus laborieuse est décrite par la Society for Automotive Engineers (SAE) dans la norme ARP 926. Dans cette procédure, le degré de criticité de tout composant est indiqué par le nombre de défaillances d'un type particulier attendu au cours d'un cycle d'un million d'utilisations se produisant dans un mode critique. Le degré de criticité est fonction de neuf paramètres, dont la plupart doivent être mesurés. Une méthode très simple pour déterminer la

---

4 A l'étude.

5 A l'étude.

criticité consiste à multiplier la probabilité de la défaillance du composant par celle du préjudice qui pourrait être causé; cette méthode est similaire à l'évaluation simple du facteur de risque.

Références:

CEI 60812:2006, *Techniques d'analyse de la fiabilité du système – Procédure d'analyse des modes de défaillance et de leurs effets (AMDE)*

Software criticality analysis of COTS/SOUP. P.Bishop, T.Clement, S.Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003

Software FMEA techniques. P.L.Goddard. In Proc Annual 2000 Reliability and Maintainability Symposium, IEEE, 2000, ISBN: 0-7803-5848-1

B.6.6.5 Analyse par arbre de panne (AAP)

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.4 et C.14 de la CEI 61508-3.

But: Faciliter l'analyse des événements ou des associations d'événements qui auront une conséquence grave ou dangereuse et calculer la probabilité d'occurrence de l'événement de tête.

Description: Partant d'un événement qui serait la cause immédiate d'une conséquence grave ou dangereuse (« l'événement de tête »), l'analyse est effectuée afin d'identifier les causes de cet événement. Cette analyse est effectuée en plusieurs étapes par l'utilisation d'opérateurs logiques (et, ou, etc.). Les causes intermédiaires sont analysées de la même façon et ainsi de suite jusqu'aux événements initiaux où s'arrête l'analyse.

La méthode est graphique et un ensemble de symboles normalisés est utilisé pour tracer l'arbre de panne. A la fin de l'analyse, l'arbre de panne représente la fonction logique qui relie les événements initiaux (généralement les défaillances de composants) à l'événement de tête (défaillance du système global). La technique est principalement destinée à l'analyse des systèmes matériels, mais on a aussi tenté d'appliquer cette approche à l'analyse des défaillances de logiciels. Cette technique peut être utilisée qualitativement pour l'analyse des défaillances (identification des scénarios de défaillance: ensembles de coupes d'ordre minimal ou implicants premiers), semi-quantitativement (par classement des scénarios selon leurs probabilités) et quantitativement pour les calculs de probabilité d'occurrence de l'événement de tête (voir C.6).

Références:

CEI 61025:2006, *Analyse par arbre de panne (AAP)*

From safety analysis to software requirements. K.M. Hansen, A.P. Ravn, A.P, V Stavridou. IEEE Trans Software Engineering, Volume 24, Issue 7, Jul 1998

B.6.6.6 Modèles de Markov

NOTE Se reporter à B.1 de la CEI 61508-6 pour l'utilisation de cette technique avec les diagrammes de fiabilité, dans le contexte de l'analyse de l'intégrité de sécurité du matériel.

But: Modéliser le comportement du système par un graphique de transition d'états et évaluer les paramètres probabilistes (par exemple non fiabilité, non disponibilité, *MTTF*, *MUT*, *MDT*) d'un système.

Description: Il s'agit d'un automate fini (voir B.2.3.2) représenté par un graphe orienté. Les nœuds (cercles) représentent les états et les arêtes (flèches) entre les nœuds représentent les transitions (défaillance, réparation, etc.) observées entre les états. Les arêtes sont pondérées par les taux de défaillance ou de réparation correspondants. La propriété

fondamentale de processus de Markov homogènes réside dans le fait que le futur dépend uniquement du présent: un changement d'état, *N,* vers l'état suivant, *N*+1, est indépendant de l'état précédent *N*-1. Cela implique que toutes les lois probabilistes des modèles sont exponentielles.

Les défaillances, états et taux peuvent être détaillés de manière à obtenir une description précise du système comme, par exemple, les défaillances détectées ou non détectées, la manifestation d'une défaillance plus importante, etc. Les intervalles d'essai périodique peuvent également être modélisés de manière appropriée en utilisant les processus de Markov appelés processus multi-phases où les probabilités des états à la fin d'une phase (par exemple juste avant un essai périodique) peuvent être utilisées pour calculer les conditions initiales pour la phase suivante (par exemple les probabilités des différents états après la réalisation d'un essai périodique).

La technique de Markov est bien adaptée à la modélisation de systèmes multiples dont le niveau de redondance varie en fonction du temps à cause des défaillances des composants et des réparations. D'autres méthodes classiques comme, par exemple, les méthodes AMDE (analyse des modes de défaillance et de leurs effets) et AAP (analyse par arbre de panne), ne peuvent être facilement adaptées à la modélisation des effets des défaillances pendant le cycle de vie complet du système étant donné qu'aucune formule combinatoire simple n'est disponible pour le calcul des probabilités correspondantes.

Dans les cas les plus simples, les formules qui décrivent les probabilités du système sont disponibles dans la documentation ou peuvent être calculées manuellement, certaines méthodes de simplification (c'est-à-dire de réduction du nombre d'états) existant déjà par ailleurs pour le traitement de cas plus complexes.

Néanmoins, d'un point de vue mathématique, un diagramme de Markov homogène ne constitue qu'un ensemble simple et courant d'équations différentielles linéaires avec des coefficients constants. Cela a fait depuis longtemps l'objet d'une analyse, des algorithmes puissants ayant par ailleurs été développés pour gérer ces calculs. Par conséquent, lorsque la taille du modèle augmente, il se révèle très efficace d'utiliser ces algorithmes appliqués dans différents progiciels.

Il doit être noté que la taille du diagramme augmente de manière exponentielle en fonction du nombre de composants: c'est ce qu'on appelle l'explosion combinatoire. Cette technique peut par conséquent être utilisée sans calculs approchés uniquement pour les systèmes de petites dimensions.

Lorsque des lois non exponentielles doivent être utilisées – semi-modèles de Markov – il convient alors d'utiliser la simulation de Monte Carlo (voir B.6.6.8).

Références:

CEI 61165:2006, *Application des techniques de Markov*

The Theory of Stochastic Processes. R. E. Cox and H. D. Miller, Methuen and Co. Ltd., London, UK, 1963

Finite MARKOV Chains. J. G. Kemeny and J. L. Snell. D. Van Nostrand Company Inc, Princeton, 1959

The Theory and Practice of Reliable System Design. D. P. Siewiorek and R. S. Swarz, Digital Press, 1982

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.6.7 Diagrammes de blocs de fiabilité (BdF)

NOTE 1 Cette technique/mesure est utilisée dans l'Annexe B de la CEI 61508-6.

NOTE 2 Voir également C.6.4 « diagrammes de blocs de fiabilité ».

But: Modéliser, sous forme de diagramme, l'ensemble des événements qui doivent se produire et les conditions qui doivent être satisfaites pour obtenir le fonctionnement correct d'un système ou d'une tâche. Il s'agit plus d'une méthode de représentation que d'une méthode d'analyse.

Description: Le but de l'analyse est représenté par un cheminement réussi constitué de cases, de lignes et de jonctions logiques. Un cheminement réussi débute d'un côté du diagramme et continue à travers les cases et les jonctions vers l'autre côté du diagramme. Une case représente une condition ou un événement et le cheminement peut traverser la case si la condition est vraie ou si l'événement s'est produit. Si le cheminement arrive à une jonction, il continue si la logique de la jonction est satisfaite. S'il atteint un sommet, le cheminement peut continuer le long de toutes les lignes sortantes. Si au moins un cheminement réussi traverse le diagramme, l'analyse est satisfaisante.

Un diagramme de fiabilité est une représentation structurelle du système modélisé. Il constitue un type de circuit électrique: lorsque le courant circule entre l'entrée et la sortie, cela signifie que le système modélisé fonctionne correctement; lorsque le circuit est coupé, cela signifie que le système modélisé fait l'objet d'une défaillance. Cela génère le concept d'ensembles de coupes d'ordre minimal qui représentent les combinaisons de défaillances (c'est-à-dire les places de « coupe » du diagramme de fiabilité), conduisant à la défaillance du système modélisé.

D'un point de vue mathématique, un diagramme de fiabilité est similaire à un arbre de panne. Il représente la fonction logique qui relie les états de chaque composant (défaillant ou en état de fonctionnement) à l'état du système global (défaillant ou en état de fonctionnement). Par conséquent, les calculs sont similaires à ceux décrits pour les arbres de panne.

Références:

CEI 61078:2006, *Techniques d'analyse pour la sûreté de fonctionnement – Bloc-diagramme de fiabilité et méthodes booléennes*

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.6.8 Simulation de Monte Carlo

NOTE Cette technique/mesure est mentionnée dans le Tableau B.4 de la CEI 61508-3 et utilisée dans l'Annexe B de la CEI 61508-6.

But: Simuler des phénomènes du monde réel en générant des nombres aléatoires lorsque la pratique ne permet pas d'utiliser des méthodes analytiques.

Description: Les simulations de Monte Carlo sont utilisées pour résoudre deux types de problèmes:

– les problèmes probabilistes, où des nombres aléatoires sont utilisés pour générer des phénomènes stochastiques; et

– les problèmes déterministes, qui sont traduits mathématiquement en un problème probabiliste équivalent (par exemple calculs intégraux).

Le principe de la simulation de Monte Carlo consiste à utiliser des nombres aléatoires pour animer un modèle de comportement fonctionnel et dysfonctionnel du système étudié. Ces modèles de comportement sont fournis par les modèles de transition d'états (diagramme de Markov, réseaux de Pétri, langages formels, etc.). La simulation de Monte Carlo est utilisée

afin de produire un échantillon statistique important à partir duquel les résultats statistiques sont obtenus.

Lors de l'utilisation de simulations de Monte Carlo, il faut assurer que les valeurs d'erreurs systématiques, de tolérances ou de bruit sont raisonnables. L'intervalle de confiance, qui peut être obtenu aisément à partir des simulations, doit permettre de gérer ces valeurs. Contrairement aux méthodes analytiques, la simulation de Monte Carlo est une méthode d'approximation autonome. Les événements ne surviennent pas simplement sans qu'il ne soit nécessaire de les identifier de manière à simplifier le modèle.

Un principe général des simulations de Monte Carlo consiste à réénoncer et reformuler le problème de manière que les résultats obtenus soient aussi précis que possible plutôt que d'aborder le problème tel qu'il était initialement énoncé.

Dans le contexte de la présente norme, la simulation de Monte Carlo peut être utilisée pour les calculs SIL et afin de prendre en considération les incertitudes liées aux données de fiabilité. Les ordinateurs actuels permettent d'effectuer les calculs SIL4 facilement.

Références:

Monte Carlo Methods. J. M. Hammersley, D. C. Handscomb, Chapman & Hall, 1979

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.6.9 Modèles d'arbres de panne

NOTE 1 Se reporter à la CEI 61508-6 pour l'utilisation de cette technique dans le contexte de l'analyse de l'intégrité de sécurité du matériel.

NOTE 2 Les arbres de panne ont déjà été décrits ci-dessus comme moyens de validation de la sécurité (voir B.6.6.5). Ils sont également largement utilisés pour l'analyse des défaillances et les calculs probabilistes.

But: Construire, par une approche graphique du haut vers le bas (effet-cause), la fonction logique qui relie les événements initiaux (modes de défaillance) à l'événement de tête (événement indésirable).

Description: Il s'agit à la fois d'une méthode d'analyse qui permet à l'analyste de développer le modèle étape par étape et d'un modèle mathématique utilisé dans des calculs probabilistes. Ce modèle permet d'effectuer:

– une analyse qualitative par l'identification et le tri des scénarios de défaillance (ensembles de coupes d'ordre minimal ou implicants premiers),

– une analyse semi-quantitative par le classement des scénarios selon leurs probabilités d'occurrence,

– une analyse quantitative par le calcul de la probabilité d'occurrence de l'événement de tête.

A l'instar du diagramme de fiabilité, un arbre de panne représente la fonction logique (booléenne) qui relie les états de chaque composant (défaillant ou en état de fonctionnement) à l'état du système global (défaillant ou en état de fonctionnement). Par conséquent, lorsque les composants sont indépendants, les calculs probabilistes peuvent être effectués en appliquant simplement les propriétés fondamentales des probabilités appliquées à la fonction logique. Ces calculs ne sont pas si faciles à effectuer dans la mesure où il s'agit d'un modèle statique qui fonctionne fondamentalement avec des probabilités véritables (c'est-à-dire constantes). Les probabilités dépendant du temps doivent être traitées avec attention. Par exemple, la valeur PFD_{avg} de systèmes de sécurité comprenant des composants faisant l'objet d'un essai périodique ne peut être calculée directement, ce calcul se révélant même plus difficile pour la valeur *PFH* des systèmes de sécurité fonctionnant en mode continu. Par conséquent, il convient que seuls les ingénieurs en fiabilité ayant une bonne connaissance

des règles mathématiques sous-jacentes effectuent les calculs de non disponibilité/*PFD* et de non fiabilité/*PFH* à l'aide de cette méthode.

Les calculs peuvent être effectués manuellement pour des arbres de panne très simples, de nombreux algorithmes ayant toutefois été développés et mis en œuvre pour gérer des équations logiques complexes au cours des 50 dernières années. L'état de la technique actuel utilise des diagrammes de décision binaire (BDD) qui constituent une technique de codage compact de l'équation logique dans la mémoire d'un ordinateur. Cette technique est, actuellement, la seule méthode capable d'effectuer les calculs probabilistes sans valeurs approchées applicables aux systèmes industriels. Elle est également suffisamment rapide pour permettre à la simulation de Monte Carlo de traiter les incertitudes.

Référence:

CEI 61025:2006, *Analyse par arbre de panne (AAP)*

B.6.6.10 Modèles de réseaux de Pétri stochastiques généralisés

NOTE 1 Se reporter à la CEI 61508-6 pour l'utilisation de cette technique dans le contexte de l'analyse de l'intégrité de sécurité du matériel.

NOTE 2 Les réseaux de Pétri ont déjà été mentionnés comme méthode semi-formelle (voir B.2.3.3). Ils peuvent également être utilisés avec efficacité dans le contexte de l'intégrité de sécurité du matériel.

But: Construire le graphe d'un modèle fonctionnel et dysfonctionnel dont le comportement est le plus proche possible de celui du système modélisé réel afin de fournir une aide efficace pour la simulation de Monte Carlo.

Description: Il s'agit d'un automate fini asynchrone tel que décrit en B.2.3.3, à l'exception du fait que la propriété appropriée recherchée lors d'une validation semi-formelle n'existe plus lors de la modélisation du comportement dysfonctionnel d'un système de sécurité. Les places ainsi désignées (illustrées par des cercles) représentent les états potentiels et les transitions (illustrées par des rectangles) représentent les événements susceptibles de se produire. Outre le marquage des places (voir B.2.3.3), des messages ou des prédicats peuvent être utilisés pour valider (activer) les transitions, le délai qui s'écoule entre la validation d'une transition et son déclenchement pouvant par ailleurs être déterministe ou stochastique. C'est la raison pour laquelle ces réseaux de Pétri sont appelés réseaux de Pétri «stochastiques généralisés».

Les réseaux de Pétri constituent des modèles de comportement souples qui se révèlent très efficaces comme aide à la simulation de Monte Carlo (voir B.6.6.8). A l'exception de la précision de la simulation de Monte Carlo proprement dite qui, dans tous les cas, est toujours connue, toutes les limites applicables aux autres méthodes (dépendances, explosion combinatoire, lois non exponentielles, etc.) sont compensées. Les ordinateurs actuels permettent de surmonter cette difficulté même pour les évaluations SIL4.

Références:

CEI 62551, *Techniques d'analyse pour la sûreté de fonctionnement – Modélisation par réseaux de Pétri (CD1)*[6]

Sécurisation des architectures informatiques. Jean-Louis Boulanger, Hermès – Lavoisier, 2009, ISBN: 978-2-7462-1991-5

B.6.7 Analyse des cas les plus défavorables

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5 et B.6 de la CEI 61508-2.

---

6 A l'étude.

But: Eviter les défaillances systématiques provenant d'associations défavorables des conditions environnementales et des tolérances des composants.

Description: L'aptitude à l'exploitation du système et le dimensionnement des composants sont étudiés ou calculés sur une base théorique. Les conditions environnementales sont modifiées, leurs valeurs marginales les plus élevées admissibles leur étant par ailleurs attribuées. Les réactions les plus essentielles du système sont inspectées et comparées à la spécification.

B.6.8 Essai fonctionnel étendu

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5 et B.6 de la CEI 61508-2.

But: Révéler les défaillances au cours des phases de spécification, de conception et de développement. Contrôler le comportement du système relatif à la sécurité en cas d'entrées rares ou non spécifiées.

Description: L'essai fonctionnel étendu étudie le comportement fonctionnel du système relatif à la sécurité en réponse aux conditions d'entrée qui sont supposées se produire que rarement (par exemple défaillance grave) ou qui ne relèvent pas de la spécification du système relatif à la sécurité (par exemple mauvaise utilisation). Pour les conditions rares, le comportement observé du système relatif à la sécurité est comparé à la spécification. Lorsque la réaction du système relatif à la sécurité n'est pas spécifiée, il convient de vérifier que la sécurité de l'usine est préservée par la réaction observée.

Références:

Software Testing and Quality Assurance. K. Naik, P. Tripathy, Wiley Interscience, 2008, Print ISBN: 9780471789116 Online ISBN: 9780470382844

The Art of Software Testing, Second Edition. G. Myers et al., Wiley & Sons, New York, 2004, ISBN 0471469122, 9780471469124

Dependability of Critical Computer Systems 3. P. G. Bishop et al.. Elsevier Applied Science, 1990, ISBN 1-85166-544-7

B.6.9 Essai du cas le plus défavorable

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5 et B.6 de la CEI 61508-2.

But: Soumettre à essai les cas spécifiés lors de l'analyse du cas le plus défavorable.

Description: L'aptitude à l'exploitation du système et le dimensionnement des composants sont soumis à essai dans les conditions du cas le plus défavorable. Les conditions environnementales sont modifiées, leurs valeurs marginales les plus élevées admissibles leur étant par ailleurs attribuées. Les réactions les plus essentielles du système sont inspectées et comparées à la spécification.

B.6.10 Essai d'insertion d'anomalie

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5 et B.6 de la CEI 61508-2.

But: Introduire ou simuler des anomalies dans le matériel du système et documenter sa réaction.

Description: Il s'agit d'une méthode qualitative d'évaluation de la sûreté de fonctionnement. Il est préférable d'utiliser des schémas de blocs fonctionnels, ainsi que des schémas de câblages et électriques détaillés pour décrire l'emplacement et le type d'anomalie et la façon dont elle est introduite; par exemple: l'alimentation de différents modules peut être coupée; l'alimentation, le bus ou les lignes d'adresses peuvent être court-circuités ou en circuit ouvert;

les composants ou leurs ports peuvent être mis en circuit ouvert ou en court-circuit; les relais peuvent ne pas se fermer ou ne pas s'ouvrir ou le faire au mauvais moment, etc. Les défaillances du système qui en résultent sont classées, comme dans les Tableaux 1 et 2 de la CEI 60812, par exemple. En principe, des anomalies uniques, en régime établi, sont introduites. Toutefois, si une anomalie n'est pas identifiée par les essais de diagnostic intégrés ou n'est pas évidente, elle peut être laissée dans le système et l'effet d'une deuxième anomalie peut être étudié. Le nombre d'anomalies peut facilement atteindre des centaines.

Le travail est effectué par une équipe multidisciplinaire et il convient que le fournisseur du système soit présent et consulté. Il convient de calculer ou d'estimer la durée moyenne de bon fonctionnement pour les anomalies qui ont de graves conséquences. Si le temps calculé est court, il convient d'apporter des modifications.

Références:

CEI 60812:2006, *Techniques d'analyse de la fiabilité du système – Procédure d'analyse des modes de défaillance et de leurs effets (AMDE)*

CEI 61069-5:1994, *Mesure et commande dans les processus industriels – Appréciation des propriétés d'un système en vue de son évaluation – Partie 5: Evaluation de la sûreté de fonctionnement d'un système*

Annexe C
(informative)

Présentation de techniques et mesures pour l'obtention de l'intégrité de sécurité du logiciel (voir CEI 61508-3)

C.1 Généralités

Il convient de ne pas considérer la présentation des techniques contenues dans la présente annexe comme complète ou exhaustive.

C.2 Exigences et conception détaillée

NOTE Des techniques et mesures appropriées sont décrites en B.2.

C.2.1 Méthodes diagrammatiques structurées

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.2 et A.4 de la CEI 61508-3.

C.2.1.1 Généralités

But: Le but principal des méthodes structurées est de promouvoir la qualité de développement du logiciel en portant l'attention sur les premières parties du cycle de vie. Pour ce faire, les méthodes font appel à des procédures et notations (assistées par ordinateur) à la fois précises et intuitives, pour déterminer et documenter des exigences et des caractéristiques d'implémentation dans un ordre logique et de manière structurée.

Description: Il existe toute une gamme de méthodes structurées. Certaines sont prévues pour les fonctions traditionnelles de traitement des données et des transactions alors que d'autres sont plus orientées vers la commande de procédé et les applications temps réel (qui tendent à être plus critiques pour la sécurité). Le langage de modélisation unifié (Unified Modeling Language (UML)) (voir C.3.12) comprend de nombreux exemples de notations structurées.

Les méthodes structurées sont essentiellement des «outils de réflexion» permettant l'approche et le partage systématiques d'un problème ou d'un système. Elles comportent les caractéristiques principales suivantes:

– un ordre de réflexion logique, avec partage d'un problème important en parties gérables;
– une analyse et une documentation du système total, comprenant l'environnement aussi bien que le système requis;
– la décomposition des données et de la fonction du système requis;
– des listes de contrôle, c'est-à-dire les listes de type d'éléments nécessitant une analyse;
– une faible servitude intellectuelle – méthodes simples, intuitives et pragmatiques;
– l'accord d'une attention toute particulière, bien souvent, au développement d'un modèle schématique du système prévu, et d'un soutien à la méthode globale grâce à l'outil CASE.

Les notations d'appui pour l'analyse et la documentation de problèmes et d'entités du système (par exemple les traitements et les flux de données) tendent à être précises, les notations qui permettent d'exprimer les fonctions de traitement effectuées par ces entités tendant toutefois à être plus informelles. Cependant, certaines méthodes utilisent effectivement en partie des notations (mathématiques) formelles (par exemple, des expressions régulières ou des automates finis). Une précision accrue non seulement réduit l'importance des erreurs de compréhension, mais permet également le traitement automatique.

Un autre avantage des notations structurées est leur visibilité qui permet la vérification intuitive d'une spécification ou d'une conception par l'utilisateur, en fonction de ses connaissances approfondies mais non formalisées.

Cette présentation décrit plusieurs méthodes structurées de manière plus détaillée.

Références:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Software Design. D. Budgen, Pearson Education, 2003, ISBN 0201722194, 9780201722192

C.2.1.2 Controlled Requirements Expression (CORE)

But: S'assurer que toutes les exigences sont déterminées et exprimées.

Description: Cette méthode a pour but de combler l'écart entre le client/utilisateur final et l'analyste. Elle n'est pas mathématiquement rigoureuse mais facilite le processus de communication: la méthode CORE permet l'expression des exigences plutôt que la spécification. L'approche est structurée et l'expression passe par différents niveaux d'affinement. La méthode CORE permet une vue élargie du problème, en tenant compte de la connaissance de l'environnement dans lequel le système sera utilisé ainsi que des points de vue respectifs des différents types d'utilisateur. Cette méthode comporte les lignes directrices et la tactique à utiliser pour reconnaître les écarts par rapport à la «conception générale». Les écarts peuvent être corrigés ou identifiés explicitement et documentés. Ainsi, les spécifications peuvent ne pas être complètes, mais les problèmes non résolus et les zones à haut risque sont détectés et doivent être pris en compte dans la suite de la conception.

Références:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794

C.2.1.3 Jackson System Development (JSD)

But: Méthode de développement couvrant le développement des systèmes logiciels à partir des exigences jusqu'au programme, en portant une attention toute particulière aux systèmes temps réel.

Description: La méthode JSD est une procédure de développement en plusieurs étapes dans laquelle le développeur modélise le comportement du monde réel devant servir de base aux fonctions du système, détermine les fonctions requises et les introduit dans le modèle, puis transforme la spécification résultante en une spécification réalisable dans l'environnement cible. Elle couvre donc les phases traditionnelles de spécification, de conception et de développement tout en différant des méthodes traditionnelles par le fait qu'elle n'est pas de conception descendante.

De plus, cette méthode porte une grande attention à l'étape initiale de découverte des entités dans l'environnement réel du système en cours de réalisation, ainsi qu'à la modélisation des entités et à ce qui peut leur arriver. Une fois cette analyse du «monde réel» effectuée et un modèle créé, les fonctions requises du système sont analysées afin de déterminer comment elles peuvent être intégrées dans ce modèle réel. Les descriptions structurées de tous les processus du modèle sont ajoutées au modèle de système résultant, l'ensemble étant alors transformé en programmes qui fonctionneront dans l'environnement logiciel et matériel prévu.

Références:

Systems Analysis and Design. D. Yeates, A. Wakefield. Pearson Education, 2003, ISBN 0273655361, 9780273655367

An Overview of JSD. J. R. Cameron. IEEE Transactions on Software Engineering, SE-12,No. 2, February 1986

C.2.1.4 Yourdon temps réel

But: Spécification et conception de systèmes temps réel.

Description: Le schéma de développement servant de base à cette technique suppose une évolution en trois étapes d'un système en cours de développement. La première étape comprend la construction d'un «modèle essentiel» décrivant le comportement requis par le système. La deuxième étape comprend la construction d'un modèle d'implémentation décrivant les structures et les mécanismes qui, lorsqu'ils sont implémentés, concrétisent le comportement requis. La troisième étape comprend la réalisation réelle du système en matériel et logiciel. Les trois étapes correspondent approximativement aux phases traditionnelles de spécification, de conception et de développement, mais portent une attention plus grande au fait qu'à chaque étape, le développeur est engagé dans une activité de modélisation.

Le modèle essentiel est constitué de deux parties:

– le modèle lié à l'environnement, contenant une description de la limite entre le système et son environnement ainsi qu'une description des événements externes auxquels le système doit répondre; et

– le modèle de comportement, contenant les schémas décrivant la transformation effectuée par le système en réponse aux événements ainsi qu'une description des données que le système doit gérer pour répondre.

Le modèle d'implémentation se divise également en sous-modèles, couvrant l'allocation des processus individuels aux processeurs et la décomposition des processus en modules logiciels.

La technique d'acquisition de ces modèles comporte un certain nombre d'autres techniques bien connues: diagrammes de flux de données, graphes de transformation, langage structuré, diagrammes de transition d'états et réseaux de Pétri. De plus, cette méthode comporte des techniques pour simuler la conception du système proposé, soit sur papier, soit mécaniquement à partir des modèles réalisés.

Références:

Real-time Systems Development. R. Williams. Butterworth-Heinemann, 2006, ISBN 0750664711, 9780750664714

Structured Development for Real-Time Systems (3 Volumes). P. T. Ward and S. J. Mellor. Yourdon Press, 1985

C.2.2 Diagrammes de flux de données

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5 et B.7 de la CEI 61508-3.

But: Décrire le flux de données d'un programme sous forme de diagramme.

Description: Les diagrammes de flux de données montrent comment les données d'entrée sont transformées en données de sortie, chaque étape de l'organigramme montrant une transformation distincte.

Les diagrammes de flux de données comportent trois éléments:

– des flèches annotées: elles représentent le flux de données entrant et sortant associé à chaque centre de transformation, avec des annotations qui documentent ce que représente les données;

– des bulles annotées: elles représentent les centres de transformation, avec des annotations qui documentent la transformation;

– des opérateurs (et, ou exclusif): ces opérateurs servent à relier les flèches annotées.

Chaque bulle d'un diagramme de flux de données peut être considérée comme une boîte noire autonome qui transforme ses entrées en sorties dès que les premières sont disponibles. Un des principaux avantages des bulles réside dans le fait qu'elles montrent les transformations sans formuler aucune hypothèse concernant la manière dont ces transformations sont implémentées. Un diagramme de flux de données pur ne comporte pas d'éléments de commande ou de séquencement, ces éléments étant fournis par des extensions temps réel aux notations, comme dans le cas de la méthode Yourdon temps réel (voir C.2.1.4).

La meilleure méthode pour la préparation des diagrammes de flux de données consiste à considérer les entrées du système puis à traiter les sorties correspondantes. Chaque bulle doit représenter une transformation distincte - il convient que la sortie de la bulle soit, d'un certain point de vue, différente de son entrée. Il n'existe pas de règles pour déterminer la structure générale du diagramme, la construction d'un diagramme de flux de données constituant par ailleurs l'un des aspects créatifs de la conception d'un système. A l'instar de toutes les conceptions, il s'agit d'une procédure itérative dont les premières ébauches sont affinées en vue d'obtenir le diagramme final.

Références:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

ISO 5807:1985, *Traitement de l'information – Symboles de documentation et conventions applicables aux données, aux organigrammes de programmation et d'analyse, aux schémas des réseaux de programmes et des ressources de système*

ISO/CEI 8631:1989, *Technologies de l'information – Structures de programmes et normes pour leur représentation*

C.2.3 Diagrammes de structures

NOTE Cette technique/mesure est mentionnée dans le Tableau B.5 de la CEI 61508-3.

But: Montrer la structure d'un programme sous forme de diagramme.

Description: Les diagrammes de structures constituent une notation complémentaire aux diagrammes de flux de données. Ils décrivent le système de programmation et la hiérarchie des différentes parties sous forme graphique, comme un arbre. Ils montrent comment les éléments d'un diagramme de flux de données peuvent être mis en œuvre en tant que hiérarchie d'unités de programme.

Un diagramme de structure montre les relations entre les modules de programme sans inclure les informations concernant l'ordre d'activation de ces unités. Ces diagrammes sont réalisés à l'aide des quatre symboles suivants:

– un rectangle annoté du nom du module;

– une ligne reliant ces rectangles en créant une structure;

– une flèche cerclée (cercle vide), annotée du nom des données échangées entre les éléments du diagramme de structure (normalement, la flèche cerclée est parallèle à la ligne reliant les rectangles du diagramme);

– une flèche cerclée (cercle plein), annotée du nom du signal de commande qui passe d'un module à un autre dans le diagramme, la flèche étant une fois encore parallèle à la ligne reliant les deux modules.

Il est possible d'obtenir un certain nombre de diagrammes de structures différents à partir de tout diagramme de flux de données même s'il n'est pas évident.

Les diagrammes de flux de données représentent la relation entre les informations et les fonctions du système. Les diagrammes de structures représentent la manière dont les éléments du système sont implémentés. Les deux techniques, bien que différentes, donnent une représentation valable du système.

Références:

Software Design & Development. G. Lancaster. Pascal Press, 2001, ISBN 1741251753, 9781741251753

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

C.2.4 Méthodes formelles

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.1, A.2, A.4 et B.5 de la CEI 61508-3.

C.2.4.1 Généralités

But: Développement du logiciel basé d'une certaine manière sur les mathématiques. Ces méthodes comprennent des techniques de conception et de codage formelles.

Description: Les méthodes formelles permettent de développer la description d'un système à tout instant de sa spécification, de sa conception ou de son implémentation. La description résultante est constituée d'une notation stricte qui peut être soumise à une analyse mathématique en vue de détecter différentes classes d'incohérence ou d'inexactitude. De plus, la description peut, dans certains cas, être analysée par une machine avec une rigueur similaire à la vérification de la syntaxe d'un programme source par un compilateur, ou animée afin de visualiser différents aspects du comportement du système décrit. L'animation peut fournir une assurance supplémentaire en ce qui concerne la conformité du système à l'exigence réelle ainsi qu'à l'exigence formellement spécifiée, étant donné qu'elle améliore l'identification par l'opérateur du comportement spécifié.

Une méthode formelle offre généralement une notation (en règle générale par l'utilisation d'une certaine forme de représentation mathématique discrète), une technique permettant d'obtenir une description dans le cadre de cette notation et différentes formes d'analyse permettant de vérifier la conformité des différents aspects de la description.

Plusieurs méthodes formelles sont décrites dans les paragraphes suivants de la présente norme: CCS, CSP, HOL, LOTOS, OBJ, logique temporelle, VDM et Z. Noter que les autres techniques, telles que les automates finis et les réseaux de Pétri (voir Annexe B), peuvent être considérées comme des méthodes formelles selon le degré de stricte conformité de ces techniques, telles qu'elles sont utilisées, à une base mathématique rigoureuse.

Références:

Formal Specification: Techniques and Applications. N.Nissanke, Springer-Verlag Telos, 1999, ISBN 1852330023

The Practice of Formal Methods in Safety-Critical Systems. S. Liu, V. Stavridou, B. Dutertre, J. Systems Software 28, 77-87, Elsevier, 1995

Formal Methods: Use and Relevance for the Development of Safety-Critical Systems. L. M. Barroca, J. A. McDermid, The Computer Journal 35 (6), 579-599, 1992

How to Produce Correct Software – An Introduction to Formal Specification and Program Development by Transformations. E. A. Boiten et al, The Computer Journal 35 (6), 547-554, 1992

C.2.4.2 Calculus of Communicating Systems (CCS)

But: CCS est un moyen de décrire et comparer le comportement de systèmes appartenant à des processus de communication concurrents.

Description: La méthode CCS est une technique de calcul mathématique applicable au comportement des systèmes. La conception des systèmes est modélisée comme un réseau de processus indépendants fonctionnant séquentiellement ou en parallèle. Les processus peuvent communiquer au moyen de ports (similaires aux voies utilisées pour la méthode CSP), la communication n'étant établie que lorsque les deux processus sont prêts. Le non-déterminisme peut être modélisé. A partir d'une description abstraite de haut niveau du système complet (connue sous le nom d'analyse), il est possible d'affiner le système par paliers afin d'obtenir une composition de plusieurs processus de communication dont le comportement total correspond à celui requis pour le système entier. Il est également possible d'utiliser une conception ascendante en combinant les processus et en déduisant les propriétés du système résultant en utilisant les règles d'inférence liées aux règles de composition.

Référence:

Communication and Concurrency. R. Milner. Pearson Education, 1989, ISBN 9780131150072

C.2.4.3 Communicating Sequential Processes (CSP)

But: CSP est une technique de spécification de systèmes logiciels concurrents, c'est-à-dire de systèmes de processus de communication fonctionnant concurremment.

Description: La méthode CSP fournit un langage pour la spécification de systèmes de processus, ainsi que la preuve permettant de vérifier que la mise en œuvre des processus satisfait à leurs spécifications (connue sous le nom d'analyse, constituée d'une séquence d'événements admissible).

Un système est modélisé comme un réseau de processus indépendants combinés séquentiellement ou en parallèle. Chaque processus est décrit en termes de tous ses comportements possibles. Les processus peuvent communiquer (synchronisation ou échange de données) au moyen de canaux, la communication n'étant établie que lorsque les deux processus sont prêts. Le rythme relatif des événements peut être modélisé.

La théorie sous-jacente à la méthode CSP a été directement incorporée dans l'architecture du transputer INMOS, et le langage OCCAM permet à un système spécifié grâce à la méthode CSP d'être directement mis en œuvre dans un réseau de transputers.

Référence:

Communicating Sequential Processes: The First 25 Years. A. Abdallah, C. Jones, J. Sanders (Eds.). Springer, 2004, ISBN 3540258132, 9783540258131

C.2.4.4 Higher Order Logic (HOL)

But: Ce langage formel est prévu pour servir de base à la spécification et la vérification du matériel.

Description: La méthode HOL se réfère à une notation logique particulière et à son système d'appui, ces deux éléments ayant été développés par le laboratoire informatique de l'Université de Cambridge. La notation logique est en grande partie dérivée de la théorie simple des types de Church et le système d'appui est basé sur le système LCF (logique des fonctions calculables).

Référence:

Higher-Order Computational Logic. J. Lloyd. In Computational Logic: Logic Programming and Beyond, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, ISBN 978-3-540-43959-2

C.2.4.5 LOTOS

But: LOTOS est un moyen permettant de décrire et de comparer le comportement de systèmes appartenant à des processus de communication concurrents.

Description: La méthode LOTOS (language for temporal ordering specification) est basée sur la méthode CCS avec des fonctions supplémentaires des algèbres associées CSP et CIRCAL (calcul circuit). Elle surmonte la faiblesse de la méthode CCS en ce qui concerne la prise en compte des structures de données et des expressions de valeur en la combinant avec une deuxième composante basée sur le langage de type de données abstrait ACT ONE. La composante de description du processus de la méthode LOTOS pourrait, toutefois, être utilisée avec d'autres formalismes pour la description des types de données abstraits.

Références:

Model Checking for Software Architectures. R. Mateescu. In Software Architecture, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2004, ISBN 978-3-540-22000-8

ISO 8807:1989, *Systèmes de traitement de l'information – Interconnexion de systèmes ouverts – LOTOS – Technique de description formelle basée sur l'organisation temporelle de comportement observationnel*

C.2.4.6 COBJ

But: Fournir une spécification de système précise avec retour d'information de l'utilisateur et validation du système avant implémentation.

Description: OBJ est un langage de spécification algébrique. Les utilisateurs spécifient les exigences en termes d'équations algébriques. Les aspects liés au comportement ou à la réalisation du système sont spécifiés en termes d'opérations agissant sur des types de données abstraits (ADT). Un type de donnée abstrait est similaire à un paquetage ADA où le comportement de l'opérateur est visible alors que les détails d'implémentation sont «cachés».

Une spécification OBJ, ainsi que l'implémentation ultérieure par paliers, sont soumises aux mêmes techniques de démonstration formelles que les autres études formelles. De plus, étant donné que les aspects de la spécification OBJ liés à la réalisation sont exécutables par une

machine, il convient d'effectuer la validation du système directement à partir de la spécification. L'exécution consiste essentiellement à évaluer une fonction par substitution d'équations (réécriture) jusqu'à ce qu'une valeur spécifique de sortie soit obtenue. Cette exécutabilité permet à l'utilisateur final du système envisagé d'avoir une «vue» de celui-ci au moment de la spécification du système sans être nécessairement au courant des techniques de spécification formelles associées.

Comme toutes les autres techniques ADT, OBJ est uniquement applicable aux systèmes séquentiels ou aux aspects séquentiels des systèmes concurrents. OBJ a été utilisé pour la spécification d'applications industrielles aussi bien restreintes qu'importantes.

Référence:

Software Engineering with OBJ: Algebraic Specification in Action. J. Goguen, G. Malcolm. Springer, 2000, ISBN 0792377575, 9780792377573

C.2.4.7 Logique temporelle

But: Expression directe des exigences de sécurité et d'exploitation et démonstration formelle du maintien de ces caractéristiques au cours des étapes de développement ultérieures.

Description: La logique des prédicats de premier ordre standard ne comporte aucune notion de temps. La logique temporelle augmente les possibilités de la logique de premier ordre en ajoutant des opérateurs modaux (par exemple «désormais» et «éventuellement»). Ces opérateurs peuvent être utilisés pour qualifier des assertions concernant le système. Par exemple, il peut être exigé que des propriétés de sécurité maintiennent l'opérateur «désormais» alors qu'il peut être exigé que d'autres états du système souhaités puissent être atteints «éventuellement» à partir d'un autre état initiateur. Les formules temporelles sont interprétées pendant les séquences d'états (comportements). La notion d' « état » dépend du niveau de description choisi. Cela peut s'appliquer au système entier, à un composant du système ou au programme informatique.

Les intervalles de temps et les contraintes quantifiés ne sont pas traités de manière explicite dans la logique temporelle. La datation absolue doit être traitée par la création d'états temporels supplémentaires comme partie intégrante de la description des états.

Référence:

Mathematical Logic for Computer Science. M. Ben-Ari. Springer, 2001, ISBN 1852333197, 9781852333195

C.2.4.8 VDM, VDM++ – Vienna Development Method

But: Spécification et implémentation systématiques de programmes séquentiels (VDM) et de programmes temps réel (VDM++) concurrents.

Description: VDM est une technique de spécification fondée sur les mathématiques et une technique d'affinement des implémentations de manière à permettre un contrôle de conformité de ces dernières par rapport à la spécification.

La technique de spécification est basée sur des modèles, en ce sens que l'état du système est modélisé en termes de structures suivant la théorie des ensembles, servant à décrire les invariants (prédicats), et les opérations associées à cet état sont modélisées en spécifiant leurs conditions initiales et leurs conditions finales en termes d'état du système. Les opérations peuvent être contrôlées afin de préserver les invariants du système.

L'implémentation de la spécification est effectuée par la réification de l'état du système en termes de structures de données dans le langage d'exécution et par le perfectionnement des

opérations en termes de programme dans le langage cible. Les phases de réification et de perfectionnement entraînent des obligations de contrôle qui permettent d'établir qu'elles sont correctes. Le concepteur choisit de rendre obligatoire ou non l'exécution de ces contrôles.

La méthode VDM est principalement utilisée durant la phase de spécification, mais elle peut être également utilisée durant les phases de conception et d'implémentation aboutissant au code source. Elle est uniquement applicable à des programmes à structure séquentielle ou aux processus séquentiels de systèmes concurrents.

L'extension orientée objet et temps réel concurrente de VDM, VDM++, est un langage de spécification formelle basé sur le langage ISO VDM-SL, et sur le langage orienté objet Smalltalk.

VDM++ fournit une large gamme de constructions de telle sorte qu'un utilisateur peut spécifier formellement et de façon orientée objet des systèmes temps réel concurrents. Dans VDM++, une spécification formelle complète consiste en une variété de spécifications de classes, et optionnellement en un espace de travail.

Les possibilités temps réel de VDM++ sont:

– les expressions temporelles, destinées à noter à la fois l'instant effectif et l'instant lié à la méthode dans un corps de méthodes donné;
– une post-expression datée, pouvant être ajoutée à une méthode pour spécifier les limites supérieures (ou inférieures) de l'instant d'exécution pour des implémentations correctes;
– des variables temporelles continues ont été introduites. Il est possible, sur la base d'hypothèses portant sur les résultats, de spécifier des relations (par exemple, des équations différentielles) entre ces fonctions du temps. Cette caractéristique a démontré une très grande utilité dans la spécification d'exigences relatives à des systèmes qui fonctionnent dans un environnement temporel continu. Les étapes de perfectionnement conduisent à des solutions logicielles discrètes pour ce type de systèmes.

Références:

/CEI 13817-1:1996, *Technologies de l'information – Langages de programmation, leurs environnements et interfaces logiciel système – Méthode de développement de Vienne – Langage de spécification – Partie 1: Langage de base*

Systematic Software Development using VDM. C. B. Jones. Prentice-Hall. 2nd Edition, 1990

Conformity Clause for VDM-SL, G. I. Parkin and B. A. Wichmann, Lecture Notes in Computer Science 670, FME'93 Industrial-Strength Formal Methods, First International Symposium of Formal Methods in Europe. Editors: J. C. P. Woodcock and P. G. Larsen. Springer Verlag, 501-520

C.2.4.9 Z

But: Z est une notation de langage de spécification pour des systèmes séquentiels et une technique de conception qui permet au développeur de passer d'une spécification Z à des algorithmes exécutables d'une manière qui permet de prouver leur conformité par rapport à la spécification.

Z est principalement utilisé durant l'étape de spécification, une méthode ayant toutefois été imaginée pour passer de la spécification à la conception et à l'implémentation. Cette méthode est la plus adaptée au développement de systèmes séquentiels orientés données.

Description: Tout comme VDM**,** la technique de spécification est basée sur des modèles, en ce sens que l'état du système est modélisé en termes de structures suivant la théorie des ensembles, servant à décrire les invariants (prédicats), et les opérations associées à cet état sont modélisées en spécifiant leurs conditions initiales et leurs conditions finales en termes

d'état du système. Les opérations peuvent être prouvées du point de vue de la préservation des invariants du système, démontrant ainsi leur cohérence. La partie formelle d'une spécification est divisée en schémas permettant la structuration des spécifications par affinage.

Normalement, une spécification Z est un mélange de méthode Z formelle et de texte explicatif informel en langage naturel. (Un texte formel proprement dit peut être trop concis pour permettre une lecture facile et nécessite souvent une explication, alors que le langage naturel informel peut facilement devenir vague et imprécis).

Contrairement à VDM, Z est une notation plutôt qu'une méthode complète. Toutefois, une méthode associée (appelée méthode B) a été développée et peut être utilisée conjointement avec la méthode Z. La méthode B est basée sur le principe d'affinage par paliers.

Références:

Formal Specification using Z, 2nd Edition. D. Lightfoot. Palgrave Macmillan, 2000, ISBN 9780333763278

The B-Method. S. Schneider. Palgrave Macmillan, 2001, ISBN 9780333792841

C.2.5 Programmation défensive

NOTE Cette technique/mesure est mentionnée dans le Tableau A.4 de la CEI 61508-3.

But: Produire des programmes capables de détecter les flux de commandes ou de données anormaux ou les valeurs de données anormales en cours d'exécution et de réagir à ces anomalies de manière prédéterminée et acceptable.

Description: Un grand nombre de techniques peuvent être utilisées pendant la programmation en vue de vérifier l'absence d'anomalies relatives aux commandes ou aux données. Ces techniques peuvent être appliquées systématiquement pendant toute la programmation d'un système en vue de réduire la probabilité d'erreurs concernant le traitement des données.

Il y a deux domaines de techniques défensives qui se superposent. Un logiciel intrinsèquement dépourvu d'erreurs est prévu pour prendre en compte ses propres défauts de conception. Ces défauts peuvent être dus à des erreurs de conception ou de codage ou à des exigences erronées. Exemples de techniques défensives:

– il convient de vérifier les limites applicables aux variables;
– il convient de vérifier la vraisemblance des valeurs, dans toute la mesure du possible;
– il convient de vérifier le type, le dimensionnement et les limites des paramètres de procédures en début de procédure.

Ces trois premières recommandations permettent d'assurer que les nombres manipulés par le programme sont plausibles, à la fois en termes de fonction du programme et de signification physique des variables.

Il convient que les paramètres de consultation seule et de lecture-écriture soient séparés et leur accès vérifié. Il convient que les fonctions traitent tous les paramètres comme des paramètres à lecture seule. Il convient que les constantes littérales ne soient pas accessibles en écriture. Cela permet de détecter tout écrasement accidentel ou toute mauvaise utilisation des variables.

Un logiciel tolérant aux anomalies est conçu pour «prévoir» les défaillances dans son propre environnement ou utiliser les conditions s'écartant des conditions nominales ou prévues, et réagir d'une manière prédéterminée. Les techniques comprennent les éléments suivants.

– il convient de vérifier la vraisemblance des variables d'entrée et des variables intermédiaires avec signification physique;

– il convient de vérifier l'effet des variables de sortie, de préférence par observation directe des changements d'état du système associés;

– il convient que le logiciel vérifie sa configuration, y compris à la fois l'existence et l'accessibilité du matériel prévu, ainsi que sa complétude. Cela est particulièrement important pour maintenir l'intégrité après les procédures de maintenance.

Certaines techniques de programmation défensive, telles que le contrôle des séquences de flux de commandes, prennent également en compte les défaillances extérieures.

Références:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Dependability of Critical Computer Systems: Guidelines Produced by the European Workshop on Industrial Computer Systems, Technical Committee 7 (EWICS TC7, Systems Reliability, Safety, and Security). Elsevier Applied Science, 1989, ISBN 1851663819, 9781851663811

C.2.6 Règles de conception et de codage

NOTE Cette technique/mesure est mentionnée dans le Tableau A.4 de la CEI 61508-3.

C.2.6.1 Généralités

But: Faciliter l'aptitude à la vérification, encourager une étude objective en équipe et mettre en œuvre une méthode de conception standard.

Description: Les règles à observer sont approuvées par les participants au début du projet. Ces règles comprennent les méthodes de conception et développement à observer (par exemple, les méthodes JSP, les réseaux de Pétri, etc.) et les règles de codage associées (voir C.2.6.2).

Ces règles sont établies pour faciliter le développement, la vérification, l'évaluation et la maintenance. Par conséquent, il convient qu'elles tiennent compte des outils disponibles, en particulier les analyseurs et les outils de rétro-ingénierie.

Références:

CEI 60880:2006, *Centrales nucléaires de puissance – Instrumentation et contrôle-commande importants pour la sûreté – Aspects logiciels des systèmes programmés réalisant des fonctions de catégorie A*

Verein Deutscher Ingenieure. Software-Zuverlassigkeit – Grundlagen, Konstruktive Mass-nahmen, Nachweisverfahren. VDI-Verlag, 1993, ISBN 3-18-401185-2

C.2.6.2 Règles de codage

NOTE Cette technique/mesure est mentionnée dans le Tableau B.1 de la CEI 61508-3.

But: Réduire la probabilité d'erreurs dans le code relatif à la sécurité et faciliter sa vérification.

Description: Les principes suivants indiquent comment les règles de codage relatives à la sécurité (pour tout langage de programmation) peuvent faciliter la conformité aux exigences normatives de la CEI 61508-3 et l'obtention des propriétés souhaitables informatives (voir Annexe F). Il convient de tenir compte des outils de support disponibles.

Exigences et recommandations de la CEI 61508-3	*Propositions de règles de codage*
Approche modulaire (Tableau A.2-7, Tableau A.4-4)	Limitation de la taille des modules logiciels (Tableau B.9-1) et maîtrise de la complexité du logiciel (Tableau B.9-2). EXEMPLES • Spécification d'une taille « locale », ainsi que de mesures et limites complexes (pour les modules) • Spécification de mesures et limites complexes « globales » (pour l'organisation globale des modules) • Limitation du nombre de paramètres / nombre fixe de paramètres de sous-programme (Tableau B.9-4) Masquage/encapsulation des informations (Tableau B.9-3): par exemple mesures incitatives pour l'utilisation de caractéristiques de langage particulières. Interface totalement définie (Tableau B.9–6). EXEMPLES • Spécification explicite de signatures de fonctions • Programmation par assertion des défaillances (Tableau A.2-3a) et vérification des données (7.9.2.7) avec spécification explicite des conditions initiales et finales pour les fonctions, des assertions et des invariants de types de données
Intelligibilité des codes • Promouvoir l'intelligibilité des codes (7.4.4.13) • Etre lisible, compréhensible et pouvant être soumis à essai (7.4.6)	Règles d'affectation des noms favorisant des noms significatifs non ambigus. Exemple: Non utilisation de noms susceptibles de prêter à confusion (par exemple IO et I0). Noms symboliques pour des valeurs numériques. Procédures et lignes directrices pour la documentation du code source (7.4.4.13). EXEMPLES • Explication des motifs et des significations (et pas uniquement l'objet) • Mises en garde • Effets secondaires Dans toute la mesure du possible, le code source doit contenir les informations suivantes (7.4.4.13): • L'entité légale (par exemple: la société, le ou les auteurs, etc.) • La description • Les entrées et sorties • L'historique de la gestion de configuration (Voir également approche modulaire)

Exigences et recommandations de la CEI 61508-3	*Propositions de règles de codage*
Vérifiabilité et testabilité • Faciliter la vérification et les essais (7.4.4.13) • Faciliter la détection des erreurs de conception ou de programmation (7.4.4.10) • Vérification formelle (Tableau A.5 - 9) • Preuve formelle (Tableau A.9 - 1)	• « Wrappers » pour les fonctions critiques de la bibliothèque, afin de vérifier les conditions initiales et finales • Mesures incitatives pour l'utilisation de caractéristiques de langage susceptibles d'exprimer des restrictions portant sur l'utilisation d'éléments ou de fonctions de données particuliers (par exemple, const) • Pour une vérification à l'aide d'outils: règles de conformité aux limites applicables aux outils sélectionnés (à condition que cela n'altère pas des objectifs plus essentiels) • Utilisation limitée de la récursion (Tableau B.1 – 6) et autres formes de dépendances circulaires (Voir également approche modulaire)
Vérification par des méthodes statiques de la conformité à la conception spécifiée (7.9.2.12)	Recommandations de codage pour l'application de concepts ou de contraintes de conception spécifiques. EXEMPLES • Recommandations de codage pour un comportement cyclique, avec temps de cycle maximal garanti (Tableau A.2-13a) • Recommandations de codage pour une architecture à déclenchement temporel (Tableau A.2-13b) • Recommandations de codage pour une architecture dirigée par les événements, avec temps de réponse maximal garanti (Tableau A.2-13c) • Boucles avec un nombre maximal d'itérations à détermination statique (à l'exception de la boucle infinie de la conception cyclique) • Recommandations de codage pour allocation de ressource statique (Tableau A.2-14) et absence d'objets dynamiques (Tableau B.1–2) • Recommandations de codage pour synchronisation statique d'accès aux ressources partagées (Tableau A.2-15) • Recommandations de codage afin de satisfaire à une utilisation limitée des interruptions (Tableau B.1–4) • Recommandations de codage afin d'éviter les variables dynamiques (Tableau B.1-3a) • Contrôle en ligne de l'installation de variables dynamiques (Tableau B.1–3b) • Recommandations de codage pour assurer la compatibilité avec les autres langages de programmation utilisés (7.4.4.10) Recommandations pour faciliter la traçabilité avec la conception.

Exigences et recommandations de la CEI 61508-3	*Propositions de règles de codage*
Sous-ensemble de langage (Tableau A.3 – 3) • Interdire les caractéristiques de langage peu sûres (7.4.4.13) • Utiliser uniquement des caractéristiques de langage définies (7.4.4.10) • Programmation structurée (Tableau A.4 - 6) • Langage de programmation fortement typé (Tableau A.3 - 2) • Pas de conversion de type automatique (Tableau B.1 – 8)	Exclusion de caractéristiques de langage générant des conceptions non structurées. EXEMPLES • Utilisation limitée des pointeurs (Tableau B.1–5) • Utilisation limitée de la récursion (Tableau B.1–6) • Utilisation limitée d'unions de type C • Utilisation limitée d'exceptions de type ADA ou C++ • Pas de branchements inconditionnels dans les programmes en langages de haut niveau (Tableau B.1-7) • Un point d'entrée/un point de sortie dans les sous-programmes et les fonctions (Tableau B.9-5) • Pas de conversion de type automatique • Utilisation limitée des effets secondaires non visibles des signatures de fonctions (par exemple des variables statiques). Aucun effet secondaire de l'évaluation des conditions et de toutes les formes d'assertions. Utilisation limitée ou documentée uniquement de caractéristiques spécifiques au compilateur. Utilisation limitée d'éléments de langage potentiellement trompeurs. Règles à appliquer lorsque ces caractéristiques de langage sont utilisées malgré tout.

Exigences et recommandations de la CEI 61508-3	*Propositions de règles de codage*
Bonne pratique de la programmation (7.4.4.13)	Le cas échéant: • Recommandations de codage visant à assurer que, si nécessaire, les expressions à virgule flottante sont évaluées dans le bon ordre (par exemple, “a-b+c” n’est pas toujours égal à “a+c-b”) • Dans les comparaisons des expressions à virgule flottante: utiliser uniquement des inégalités (<, ≤, >, ≥) et non des égalités strictes • Recommandations concernant la compilation conditionnelle et le «traitement préalable» • Vérification systématique des conditions de retour (réussite / échec) Documentation et, dans toute la mesure du possible, automatisation de la génération d’un code exécutable (façonnage de fichiers). Prévention des effets secondaires non visibles des signatures de fonctions. Lorsque de tels effets secondaires existent, prévoir des recommandations de documentation. Mise entre parenthèses lorsque la préséance des opérateurs n’est pas absolument évidente. Déclenchement de situations présumées impossibles (par exemple, un cas d’ « anomalie » avec les « interrupteurs » de type C). Utilisation de « wrappers » pour les modules critiques, notamment pour vérifier les conditions initiales et finales, ainsi que les conditions de retour. Recommandations de codage afin de satisfaire aux erreurs de compilateur connues et aux limites établies par l’évaluation du compilateur.

C.2.6.3 Pas de variables dynamiques ni d'objets dynamiques

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.2 et B.1 de la CEI 61508-3.

But: Exclure

– tout recouvrement de mémoire non désiré ou non détecté;

– tout goulot d'étranglement des ressources pendant la durée d'exécution (liée à la sécurité).

Description: Dans le cas de cette mesure, les variables dynamiques et les objets dynamiques sont celles et ceux qui font l'objet d'une allocation de mémoire et dont les adresses absolues sont déterminées au moment de l'exécution. La valeur de la mémoire allouée et ses adresses correspondantes dépendent de l'état du système au moment de l'allocation, ce qui signifie l'impossibilité d'un contrôle par le compilateur ou tout autre outil hors ligne.

Etant donné que le nombre de variables et d'objets dynamiques, ainsi que l'espace mémoire disponible existant pour l’allocation de nouveaux objets ou variables dynamiques, dépendent de l'état du système au moment de l’allocation, il est possible que des anomalies surviennent au moment de l’allocation ou de l'utilisation des variables ou des objets. Par exemple, si

l'espace mémoire disponible à l'emplacement alloué par le système est insuffisant, l'écrasement accidentel du contenu mémoire d'une autre variable peut se produire. Ces anomalies sont évitées lorsque aucun objet ou variable dynamique n'est utilisé(e).

Des restrictions d'utilisation des objets dynamiques se révèlent nécessaires lorsqu'une analyse statique ne permet pas de prévoir avec précision le comportement dynamique (c'est-à-dire préalablement à l'exécution du programme), et par conséquent une exécution prévisible dudit programme ne peut être garantie.

C.2.6.4 Contrôle en ligne pendant la création de variables ou d'objets dynamiques

NOTE 1 Cette technique/mesure est mentionnée dans le Tableau B.1 de la CEI 61508-3.

But: Contrôler que la mémoire qui doit être allouée à des variables et à des objets dynamiques est libre avant cette allocation, en s'assurant que l'allocation de variables et d'objets dynamiques pendant la durée d'exécution n'a aucune incidence sur les variables, les données ou le code existants.

Description: Dans le cas de cette mesure, les variables dynamiques sont celles qui font l'objet d'une allocation de mémoire et dont les adresses absolues sont déterminées au moment de l'exécution (dans ce sens, les variables sont également les attributs des instances d'objets).

Grâce à des moyens matériels ou logiciels, la mémoire est contrôlée en vue d'assurer qu'elle est libre avant l'allocation d'une variable ou d'un objet dynamique (par exemple pour éviter tout débordement de pile). Si l'allocation n'est pas autorisée (par exemple, si la mémoire à l'adresse déterminée est insuffisante), une action appropriée doit être entreprise. Après l'utilisation d'une variable ou d'un objet dynamique (par exemple, après la sortie d'un sous-programme), toute la mémoire allouée à cette variable doit être libérée.

NOTE 2 Une solution alternative consiste à démontrer statistiquement que la mémoire convient dans tous les cas.

C.2.6.5 Utilisation limitée des interruptions

NOTE Cette technique/mesure est mentionnée dans le Tableau B.1 de la CEI 61508-3.

But: S'assurer que le logiciel est vérifiable et peut être soumis à essai.

Description: Il convient que l'utilisation des interruptions soit limitée. Les interruptions peuvent être utilisées si elles simplifient le système. Il convient que le traitement logiciel des interruptions soit inhibé durant les opérations critiques (par exemple, les opérations à durée critique ou les changements critiques pour les données) des fonctions exécutées. Si des interruptions sont utilisées, il convient alors que les opérations non interruptibles aient un temps de traitement maximal spécifié, de manière à pouvoir calculer le temps maximal durant lequel une interruption est inhibée. Il convient que l'utilisation et le masquage des interruptions soient soigneusement documentés.

C.2.6.6 Utilisation limitée des pointeurs

NOTE Cette technique/mesure est mentionnée dans le Tableau B.1 de la CEI 61508-3.

But: Eviter les problèmes résultant de l'accession à des données sans contrôle préalable des limites et du type de pointeur. Permettre l'essai des modules et la vérification du logiciel. Limiter les conséquences des défaillances.

Description: Les opérations arithmétiques relatives à un pointeur du logiciel d'application peuvent être utilisées au niveau du code source uniquement si le type de données et les limites de valeur du pointeur (afin d'assurer que la référence du pointeur est située dans l'espace d'adresse correct) sont contrôlés avant l'accès. Il convient que la communication entre les différentes tâches du logiciel d'application ne soit pas réalisée par référence directe

entre les tâches. Il convient que les échanges de données soient effectués au moyen du système d'exploitation.

C.2.6.7 Utilisation limitée de la récursion

NOTE Cette technique/mesure est mentionnée dans le Tableau B.1 de la CEI 61508-3.

But: Eviter l'utilisation non vérifiable et ne pouvant être soumise à essai des appels de sous-programme.

Description: En cas d'utilisation de la récursion, un critère clair permettant de prédire la profondeur de récursion doit être appliqué.

C.2.7 Programmation structurée

NOTE Cette technique/mesure est mentionnée dans le Tableau A.4 de la CEI 61508-3.

But: Concevoir et implémenter le programme de manière à ce qu'il soit pratique de l'analyser sans exécution. Le programme peut contenir uniquement un comportement absolu minimal ne pouvant statistiquement pas être soumis à essai.

Description: Il convient d'appliquer les principes suivants en vue de minimiser la complexité structurelle:

– diviser le programme en modules logiciels de taille plus petite et appropriée en s'assurant que ces modules sont découplés autant qu'il est possible et que toutes les interactions sont explicites;

– composer le flux de commandes des modules logiciels à l'aide de constructions structurées, c'est-à-dire les séquences, les itérations et la sélection;

– maintenir le nombre de cheminements potentiels à travers un module logiciel aussi faible que possible, et la relation entre les paramètres d'entrée et de sortie aussi simple que possible;

– éviter les branchements compliqués et, en particulier, les branchements inconditionnels (goto) dans les langages de haut niveau;

– lorsque cela est possible, lier les contraintes de bouclage et les branchements aux paramètres d'entrée;

– éviter les décisions de branchement et de bouclage reposant sur l'utilisation de calculs complexes.

Il convient que les fonctions du langage de programmation qui encouragent l'approche ci-dessus soient utilisées de préférence aux autres fonctions qui sont (supposées) plus efficaces, sauf lorsque l'efficacité constitue une priorité absolue (par exemple, dans le cas de certains systèmes critiques relatifs à la sécurité).

Références:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

A Discipline of Programming. E. W. Dijkstra. Englewood Cliffs NJ, Prentice-Hall, 1976

C.2.8 Masquage/encapsulation des informations

NOTE Cette technique/mesure est mentionnée dans le Tableau B.9 de la CEI 61508-3.

But: Prévenir tout accès involontaire aux données ou procédures et ainsi maintenir une bonne structure du programme.

Description: Les données qui sont globalement accessibles à toutes les composantes logicielles peuvent être modifiées accidentellement ou de manière erronée par l'un de ces éléments. Toute modification de ces structures de données peut nécessiter l'examen détaillé du code et des modifications importantes.

Le masquage des informations est une approche générale permettant de minimiser ces difficultés. Les structures des données essentielles sont «masquées» et ne peuvent être manipulées qu'à l'aide d'un jeu déterminé de procédures d'accès. Cela permet de modifier les structures internes et d'ajouter des procédures supplémentaires sans affecter le comportement fonctionnel du reste du logiciel. Par exemple, un répertoire de noms peut comporter les procédures d'accès «insertion», «annulation» et «recherche». Les procédures d'accès et les structures des données internes peuvent être réécrites à l'aide de ces procédures (par exemple pour utiliser une méthode de consultation différente ou pour stocker les noms sur un disque dur) sans affecter le comportement logique du reste du logiciel.

A ce propos, il convient d'utiliser la notion de types de donnée abstraits. Si une aide directe n'est pas fournie, il peut alors être nécessaire de vérifier que l'abstraction n'a pas été rompue par inadvertance.

Références:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

On the Design and Development of Program Families. D. L. Parnas. IEEE Trans SE-2, March 1976

C.2.9 Approche modulaire

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.4 et B.9 de la CEI 61508-3.

But: Décomposition d'un système logiciel en petites parties compréhensibles de manière à limiter la complexité du système.

Description: Une approche modulaire ou modularisation comporte plusieurs règles concernant les phases de conception, de codage et de maintenance d'un projet de logiciel. Ces règles varient selon la méthode de conception employée. La plupart des méthodes comportent les règles suivantes:

– il convient qu'un module logiciel (ou sous-programme, de manière équivalente) ait une seule tâche ou fonction bien définie à remplir;

– il convient que les connexions entre modules logiciels soient limitées et strictement définies, la cohérence au niveau d'un module logiciel devant par ailleurs être forte;

– il convient que les ensembles de sous-programmes soient réalisés de manière à obtenir plusieurs niveaux de modules logiciels;

– il convient que la taille des sous-programmes soit limitée à une certaine valeur spécifiée, par exemple, deux à quatre fois la taille de l'écran;

– il convient que les sous-programmes aient une seule entrée et une seule sortie;

– il convient que les modules logiciels communiquent avec d'autres modules logiciels à travers leurs interfaces - lorsque des variables globales ou communes sont utilisées, il convient qu'elles soient bien structurées, que leur accès soit contrôlé et que leur utilisation soit chaque fois justifiée;

– il convient que toutes les interfaces des modules logiciels soient parfaitement documentées;

– il convient que toute interface de modules logiciels contienne uniquement les paramètres nécessaires à sa fonction. Toutefois, cette recommandation est difficile à appliquer du fait

de la possibilité pour un langage de programmation d'autoriser les paramètres par défaut, ou de l'utilisation d'une approche orientée objet.

Références:

The Art of Software Testing, Second Edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

C.2.10 Utilisation de composants logiciels éprouvés/ vérifiés

NOTE 1 Cette technique/mesure est mentionnée dans les Tableaux A.2, C.2, A.4 et C.4 de la CEI 61508-3.

But: Eviter la nécessité pour les conceptions et les composantes logicielles de devoir être amplement revalidées ou reconçues pour chaque application nouvelle. Profiter des conceptions qui n'ont pas été vérifiées formellement ou rigoureusement mais qui bénéficient d'une expérience opérationnelle considérable. Tirer avantage d'un élément logiciel préexistant qui a été vérifié pour une application différente et pour laquelle des éléments de preuve de vérification existent.

Description: Cette mesure vérifie que les composantes logicielles sont suffisamment exemptes d'anomalies de conception systématiques et/ou de défaillances opérationnelles.

Les éléments les plus rudimentaires ne permettent généralement pas de constituer un système complexe. Il est généralement nécessaire d'utiliser les principaux sous-ensembles (« éléments », voir 3.4.5 et 3.2.8 de la CEI 61508-4) qui ont été développés précédemment afin de fournir certaines fonctions utiles et qui peuvent être réutilisés pour implémenter certaines parties du nouveau système.

Les PES (systèmes électroniques programmables) correctement conçus et structurés sont constitués d'un certain nombre de composantes logicielles clairement distinctes et dont les interactions sont clairement spécifiées. La constitution d'une bibliothèque comprenant de telles composantes logicielles d'application générale pouvant être réutilisées dans plusieurs applications, permet de partager la plupart des ressources nécessaires à la validation des conceptions entre plusieurs applications.

Toutefois, il est essentiel, pour les applications relatives à la sécurité, d'être suffisamment confiant dans le fait que le nouveau système qui intègre ces éléments préexistants présente l'intégrité de sécurité requise, et que la sécurité n'est pas remise en cause par le comportement incorrect de l'un des éléments préexistants.

Deux points de vue peuvent être adoptés afin d'assurer que le comportement d'un élément préexistant peut être déterminé avec précision:

- analyser l'historique d'exploitation complet de l'élément afin de démontrer que ce dernier a été éprouvé (par une utilisation antérieure);
- évaluer les éléments de preuves de vérification qui ont été rassemblés sur le comportement de l'élément afin de déterminer si ce dernier satisfait aux exigences de la présente norme.

C.2.10.1 Eprouvé par les utilisations antérieures

Dans de rares cas seulement, la notion d'éprouvé par une utilisation antérieure (voir 3.8.18 de la CEI 61508-4) sera un argument suffisant indiquant qu'un élément logiciel éprouvé

atteint l'intégrité de sécurité nécessaire. Pour les composants complexes comportant un grand nombre de fonctions possibles (par exemple, un système d'exploitation), il est essentiel de déterminer quelles fonctions du composant concerné ont été réellement et suffisamment éprouvées (par une utilisation antérieure). Par exemple, lorsqu'un programme d'essai automatique est utilisé en vue de détecter les anomalies, il ne peut pas être considéré que ce programme de détection des anomalies a été éprouvé (par une utilisation antérieure) si aucune défaillance n'est intervenue pendant la période de fonctionnement.

Un élément logiciel peut être considéré comme éprouvé (par une utilisation antérieure) s'il remplit les critères suivants:

– spécification inchangée;
– systèmes utilisés dans des applications différentes;
– au moins un an d'historique en service;
– temps de fonctionnement dépendant du niveau d'intégrité relative à la sécurité ou du nombre approprié de sollicitations; démonstration d'un taux de défaillance non relative à la sécurité inférieur à:
  – 10^{-2} par sollicitation (an) avec un niveau de confiance de 95 % nécessite 300 exploitations opérationnelles (ans),
  – 10^{-5} par sollicitation (an) avec un niveau de confiance de 99,9 % nécessite 690 000 exploitations opérationnelles (ans);

NOTE 1 Voir à l'Annexe D quelques aspects mathématiques sous-jacents aux estimations numériques ci-dessus. Voir également en B.5.4 une approche de mesure et statistique similaire.

– toute l'expérience en exploitation doit se rapporter à un profil de sollicitation connu des fonctions de l'élément logiciel, afin d'assurer qu'une expérience en exploitation accrue conduit véritablement à une connaissance également accrue du comportement de l'élément logiciel par rapport à ce profil de sollicitation;
– pas de défaillances relatives à la sécurité.

NOTE 2 Une défaillance qui, dans un certain contexte, peut ne pas être critique par rapport à la sécurité, peut, dans un autre contexte, être critique par rapport à la sécurité, et inversement.

Les éléments suivants doivent être documentés en vue de vérifier qu'un élément logiciel remplit les critères énumérés:

– l'identification exacte de chaque système et de ses composants, y compris les numéros de version (à la fois pour les composants logiciels et matériels);
– l'identification des utilisateurs et durée d'utilisation;
– la durée de fonctionnement;
– la procédure de sélection des systèmes appliqués à l'utilisateur et des cas d'application;
– les procédures de détection et de prise en compte des défaillances et d'élimination des anomalies.

C.2.10.2 Evaluer des éléments de preuve de vérification

Un élément logiciel préexistant (voir 3.2.8 de la CEI 61508-4) est un élément déjà existant et n'ayant pas été spécifiquement développé pour le projet ou SRS (système relatif à la sécurité) actuel. Le logiciel préexistant peut être un produit disponible dans le commerce ou avoir été développé par une organisation pour un produit ou système antérieur. Le logiciel préexistant peut avoir été ou non développé conformément aux exigences de la présente norme.

Pour évaluer l'intégrité de sécurité du nouveau système intégrant le logiciel préexistant, des éléments de preuve de vérification sont nécessaires pour déterminer le comportement de l'élément préexistant. Ce comportement peut être déduit (1) des documents propres du fournisseur des éléments, ainsi que des registres du processus de développement de l'élément, ou (2) peut être généré ou complété par des activités de qualification

supplémentaires entreprises par le développeur du nouveau système relatif à la sécurité, ou par des tierces parties. Le "manuel de sécurité d'article conforme" définit les capacités et les limites de l'élément logiciel potentiellement réutilisable.

Dans tous les cas, un manuel de sécurité d'article conforme doit exister (ou doit être produit) et permettre d'effectuer une évaluation de l'intégrité d'une fonction de sécurité spécifique qui dépend en totalité ou en partie de l'élément réutilisé. A défaut, il doit en être conclu, de façon prudente, que l'élément n'a pas été justifié pour une réutilisation relative à la sécurité. (Il est évident que l'élément ne peut être justifié dans tous les cas, mais que des preuves insuffisantes ont simplement été établies dans ce cas particulier).

La présente norme comprend des exigences spécifiques relatives au contenu du manuel de sécurité d'article conforme, voir l'Annexe D de la CEI 61508-2, l'Annexe D de la CEI 61508-3 et 7.4.2.12 et 7.4.2.13 de la CEI 61508-3.

Noter, pour une indication très succincte de son contenu, que le manuel de sécurité d'article conforme traitera des éléments suivants:

– la conception de l'élément est connue et documentée;
– l'élément a fait l'objet d'une vérification et d'une validation à l'aide d'une approche systématique comprenant des essais documentés et une revue de toutes les parties constitutives de la conception de l'élément et de son code;
– les fonctions non utilisées et inutiles de l'élément n'empêcheront pas le nouveau système de satisfaire à ses exigences de sécurité;
– tous les mécanismes de défaillance crédibles de l'élément dans le nouveau système ont été identifiés et une mesure de réduction appropriée a été implémentée.

Une évaluation de sécurité fonctionnelle du nouveau système doit établir que l'élément réutilisé est appliqué strictement dans les limites de capacité qui ont été justifiées par les éléments de preuve et les hypothèses contenus dans le manuel de sécurité d'article conforme.

Références:

Component-Based Software Development: Case Studies. Kung-Kiu Lau. World Scientific, 2004, ISBN 9812388281, 9789812388285

Software Reuse and Reverse Engineering in Practice. P. A. V. Hall (ed.), Chapman & Hall, 1992, ISBN 0-412-39980-6

Software criticality analysis of COTS/SOUP. P.Bishop, T.Clement, S.Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003

C.2.11 Traçabilité

But: Maintenir une cohérence entre les phases du cycle de vie.

Description: Il est essentiel, afin d'assurer que le logiciel issu des activités du cycle de vie satisfait aux exigences relatives au fonctionnement correct du système relatif à la sécurité, de garantir une certaine cohérence entre les phases du cycle de vie. Le concept clé sous-jacent est celui de la traçabilité entre les activités. Il s'agit essentiellement d'une analyse d'impact qui consiste à vérifier (1) que les décisions prises à une phase précédente sont mises en œuvre de manière appropriée dans les phases ultérieures (traçabilité ascendante), et (2) que les décisions prises à une phase ultérieure sont effectivement requises et autorisées par les décisions prises précédemment.

La traçabilité ascendante consiste essentiellement à vérifier qu'une exigence est prise en compte de manière appropriée dans les phases ultérieures du cycle de vie. La traçabilité ascendante est constructive en plusieurs points du cycle de vie de sécurité:

- des exigences de sécurité du système aux exigences pour la sécurité du logiciel;
- de la spécification des exigences pour la sécurité du logiciel à l'architecture de ce dernier;
- de la spécification des exigences pour la sécurité du logiciel à la conception de ce dernier;
- de la spécification de conception du logiciel aux spécifications du module et de l'essai d'intégration;
- des exigences de conception du système et du logiciel pour l'intégration du matériel/logiciel, aux spécifications de l'essai d'intégration de ces derniers;
- de la spécification des exigences pour la sécurité du logiciel au plan de validation de la sécurité de ce dernier;
- de la spécification des exigences pour la sécurité du logiciel au plan de modification de ce dernier (y compris la revérification et la revalidation);
- de la spécification de conception du logiciel au plan de vérification de ce dernier (y compris la vérification des données);
- des exigences de l'Article 8 de la CEI 61508-3 au plan d'évaluation de la sécurité fonctionnelle du logiciel.

Une traçabilité descendante consiste essentiellement à vérifier que chaque décision d'implémentation (interprétée dans un contexte large et non limitée à l'implémentation d'un code) est clairement justifiée par une exigence donnée. En l'absence de cette justification, l'implémentation comporte un ou plusieurs éléments inutiles qui renforcent la complexité et ne prennent pas nécessairement en compte toutes les exigences véritables du système relatif à la sécurité. La traçabilité descendante est constructive en plusieurs points du cycle de vie de sécurité:

- des exigences de sécurité aux besoins de sécurité perçus;
- de l'architecture du logiciel à la spécification des exigences pour la sécurité de ce dernier;
- de la conception détaillée du logiciel à l'architecture du logiciel;
- du code du logiciel à la conception détaillée du logiciel;
- du plan de validation du logiciel à la spécification des exigences pour la sécurité de ce dernier;
- du plan de modification du logiciel à la spécification des exigences pour la sécurité de ce dernier;
- du plan de vérification du logiciel (y compris la vérification des données) à la spécification de conception de ce dernier.

Référence:

*Requirements Engineering***.** E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794

C.2.12 Conception du logiciel sans état (ou conception à état limité)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Limiter la complexité du comportement du logiciel.

Description: Considérer un programme logiciel qui traite une séquence de transactions: le programme reçoit une séquence d'entrées et génère une sortie en réponse à chaque entrée. Il peut également mémoriser tout ou partie de son historique dans un « état », et peut tenir compte de cet état lorsqu'il calcule le mode de réponse à utiliser pour les entrées futures.

Lorsque la sortie du programme en réponse à une entrée spécifique est déterminée entièrement par cette entrée, le programme est alors déclaré être sans mémoire ou sans état. Chaque transaction entrée/sortie est complète en ce sens qu'aucune transaction n'est influencée par une transaction antérieure, et une entrée spécifique génère toujours la même sortie associée.

En revanche, lorsque le programme tient compte, pour le calcul de sa sortie, à la fois de son entrée spécifique et de son état mémorisé, il est alors capable de présenter un comportement plus complexe dans la mesure où il peut fournir des sorties différentes en réponse à la même entrée, et ce, en différentes occasions. La réponse à une entrée spécifique peut dépendre du contexte (c'est-à-dire les entrées et sorties précédentes) de réception de cette dernière. Un autre élément à prendre en considération, et pertinent pour certaines applications (généralement dans le cas de communications), réside dans le fait que le comportement du programme peut être particulièrement sensible aux changements opérés dans l'état mémorisé, que ce soit de manière non intentionnelle ou mal intentionnée.

Une conception sans état (ou à état limité) est une approche générale dont le but est de réduire au minimum la complexité potentielle du comportement du logiciel en prévenant ou en réduisant au minimum l'utilisation des informations relatives à l'état dans la conception du logiciel.

Références:

Introduction to Automata Theory, Languages, and Computation (3rd Edition). J. Hopcroft, R. Motwani, J. Ullman, Addison-Wesley Longman Publishing Co, 2006, ISBN:0321462254

Stateless connections. T. Aura, P Nikander. In Proc International Conference on Information and Communications Security (ICICS'97), ed Yongfei Han. Springer, 1997, ISBN 354063696X, 9783540636960

C.2.13 Analyse numérique hors ligne

NOTE Cette technique/mesure est mentionnée dans le Tableau A.9 de la CEI 61508-3.

But: Garantir la précision des calculs numériques.

Description: Une imprécision numérique peut apparaître dans le calcul d'une fonction mathématique comme conséquence de l'utilisation de représentations finies de fonctions et de nombres idéaux. Une erreur de troncature est introduite lorsqu'une fonction est calculée de manière approchée par un nombre fini de termes d'une série infinie telle qu'une série de Fourier. Une erreur d'arrondi est introduite par la représentation précise finie de nombres réels dans un ordinateur physique. Lorsque le calcul en virgule flottante est effectué sur la base de tout calcul à l'exception du calcul le plus simple, la validité de ce dernier doit être vérifiée afin d'assurer que la précision requise par l'application est effectivement obtenue.

Référence:

Guide to Scientific Computing. P.R. Turner. CRC Press, 2001, ISBN 0849312426, 9780849312427

C.2.14 Diagrammes de séquence de messages

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.7 et C.17 de la CEI 61508-3.

But: Faciliter la prise en compte des exigences du système dans les premières phases de conception du développement du logiciel, y compris les exigences et la conception de

l'architecture du logiciel. Pour le langage de modélisation unifié (UML)[7], la désignation "diagramme de séquence du système" est utilisée pour cette notation.

Description: Le diagramme de séquence de messages est un mécanisme qui permet de décrire le comportement d'un système en termes de la communication qui a lieu entre les composants du système (le composant pouvant être un être humain, un système informatique ou un composant ou objet logiciel, selon la phase de conception). Pour chaque composant, une "ligne de vie" verticale est tracée sur le diagramme et des flèches entre les lignes de vie sont utilisées pour représenter les messages. A la réception des messages, les actions peuvent également être représentées sur les diagrammes par des cases. Un ensemble de scénarios (décrivant à la fois le comportement souhaitable et non souhaitable) est constitué comme spécification du comportement requis du système. Ces scénarios peuvent être utilisés dans plusieurs applications. Ils peuvent être animés afin de démontrer le comportement du système aux utilisateurs finals. Ces scénarios peuvent être transformés en une implémentation exécutable du système. Ils peuvent constituer par ailleurs la base des données d'essai.

Le langage de modélisation unifié (UML) comprend des extensions au concept initial de diagramme de séquence de messages sous la forme de constructions de sélection et d'itération qui permettent de réaliser des scénarios de branchement et de boucle, sous réserve d'une notation plus compacte. Les sous-diagrammes peuvent également être définis et référencés à partir d'un certain nombre de diagrammes de séquence de niveau supérieur. Le temporisateur et les événements externes peuvent également être représentés.

Références:

Message Sequence charts, D. Harel, P. Thiagarajan. In UML for Real: Design of Embedded Real-Time Systems. ed. L. Lavagno. Springer, 2003, ISBN 1402075014, 9781402075018

ISO/CEI 19501:2005, *Technologies de l'information – Traitement distribué ouvert – Langage de modélisation unifié (UML), version 1.4.2*

C.3 Conception de l'architecture

C.3.1 Détection d'anomalie et diagnostic

NOTE Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Détecter les anomalies d'un système susceptibles d'occasionner une défaillance en vue d'établir les bases des contre-mesures à utiliser pour minimiser les conséquences des défaillances.

Description: La détection d'anomalie est l'activité permettant de vérifier la présence d'états erronés à l'intérieur d'un système (états causés par une anomalie du système ou sous-système à contrôler). L'objectif principal de la détection d'anomalie est de neutraliser les effets de résultats erronés. Un système qui agit conjointement avec des composants parallèles, et qui «rend la main» lorsqu'il détecte que ses propres résultats sont incorrects, est appelé «dispositif d'autocontrôle».

La détection d'anomalie est fondée sur les principes de redondance (principalement pour la détection des anomalies matérielles - voir l'Annexe A de la CEI 61508-2) et de diversité (anomalies logicielles). Un système de vote est nécessaire pour décider de la conformité des résultats. Les méthodes spéciales applicables sont: la programmation par assertion, la programmation N-versions et la technique diversifiée de surveillance; ainsi que l'introduction d'éléments matériels, tels que: capteurs additionnels, boucles de contrôle, codes détecteurs d'erreur, etc.

---

7 UML = *Unified Modeling Language*.

La détection d'anomalie peut être obtenue par des contrôles effectués sur les valeurs ou les temps à différents niveaux, en particulier au niveau physique (température, tension, etc.), logique (codes détecteurs d'erreur), fonctionnel (assertions) ou externe (contrôles de vraisemblance). Les résultats de ces contrôles peuvent être mémorisés et associés aux données concernées en vue de faciliter la recherche des défaillances.

Les systèmes complexes sont constitués de sous-systèmes. L'efficacité de la détection d'anomalie, du diagnostic et de la compensation des anomalies dépend de la complexité des interactions entre les sous-systèmes, laquelle complexité a un effet sur la propagation des anomalies.

Il convient que le diagnostic d'anomalie soit utilisé au niveau du plus petit sous-système, puisque les petits sous-systèmes permettent un diagnostic d'anomalie plus détaillé (détection des états erronés).

Des systèmes d'information intégrés au niveau de l'entreprise peuvent communiquer l'état des systèmes relatifs à la sécurité de manière périodique aux autres systèmes de supervision (y compris les informations d'essai de diagnostic). Toute détection d'anomalie peut être mise en évidence et utilisée pour déclencher une action corrective avant que la situation ne devienne dangereuse. Enfin, en cas d'incident effectif, la documentation de telles anomalies peut faciliter l'analyse ultérieure.

Référence:

Dependability of Critical Computer Systems 1. F. J. Redmill, Elsevier Applied Science, 1988, ISBN 1-85166-203-0

C.3.2 Codes de détection et correction d'erreurs

NOTE Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Détecter et corriger les erreurs concernant les informations sensibles.

Description: Pour une information de *n* bits, un bloc codé de *k* bits est généré afin de permettre la détection et la correction de *r* erreurs. Deux exemples types sont les codes de Hamming et les codes polynomiaux.

Il convient de noter que, dans les systèmes relatifs à la sécurité, il sera normalement nécessaire d'éliminer les anomalies plutôt que de tenter de les corriger, étant donné que seule une portion prédéterminée d'erreurs peut être corrigée de manière satisfaisante.

Référence:

Fundamentals of Error-correcting Codes, W. Huffman, V. Pless. Cambridge University Press, 2003, ISBN 0521782805, 9780521782807

C.3.3 Programmation par assertion des défaillances

NOTE Cette technique/mesure est mentionnée dans le Tableau A.17 de la CEI 61508-2 et dans les Tableaux A.2 et C.2 de la CEI 61508-3.

But: Détecter les anomalies de conception logicielle résiduelles pendant l'exécution d'un programme en vue d'éviter les défaillances du système critiques pour la sécurité et de poursuivre l'exécution afin d'obtenir un haut niveau de fiabilité.

Description: L'idée directrice de la méthode de programmation par assertion est de contrôler une précondition (la validité des conditions initiales est vérifiée avant l'exécution d'une séquence d'instructions) et une postcondition (les résultats sont vérifiés après l'exécution

d'une séquence d'instructions). Si la précondition ou la postcondition n'est pas remplie, le traitement signale l'erreur.

Par exemple,

```
assertion de < précondition>;
   action 1;
      :
      :
   action x;
assertion de < post-condition>;
```

Références:

Exploiting Traces in Program Analysis. A. Groce, R. Joshi. Lecture Notes in Computer Science vol 3920, Springer Berlin / Heidelberg, 2006, ISBN 978-3-540-33056-1

Software Development – A Rigorous Approach. C. B. Jones, Prentice-Hall, 1980

C.3.4 Dispositif de surveillance diversifiée

NOTE Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Assurer la protection du logiciel contre les anomalies de spécification et d'implémentation résiduelles qui affectent la sécurité.

Description: Deux approches de surveillance peuvent être différenciées: 1) le moniteur et la fonction surveillée dans le même ordinateur, avec une certaine garantie d'indépendance entre eux; et 2) le moniteur et la fonction surveillée avec des ordinateurs séparés.

Un dispositif de surveillance diversifiée est un système de surveillance externe qui est mis en œuvre à partir d'un ordinateur indépendant selon une spécification différente. Ce dispositif a pour seul objectif d'assurer que l'ordinateur principal exécute des opérations sûres mais pas nécessairement correctes. Le dispositif de surveillance diversifiée contrôle l'ordinateur principal de façon permanente. Il évite par ailleurs tout état d'insécurité du système. De plus, si le dispositif de surveillance détecte un état potentiellement dangereux de l'ordinateur principal, le système doit être ramené à l'état sûr soit par ledit dispositif, soit par l'ordinateur principal.

Il convient que le matériel et le logiciel du dispositif de surveillance diversifiée soient classés et qualifiés suivant le SIL approprié.

Référence:

Requirements based Monitors for Real-Time Systems, D. Peters, D. Parnas. IEEE Transactions on Software Engineering, vol. 28, no. 2, 2002

C.3.5 Diversité logicielle (programmation diversifiée)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Détecter et masquer les anomalies résiduelles de conception et d'implémentation logicielle pendant l'exécution d'un programme en vue d'éviter les défaillances du système critiques pour la sécurité et de poursuivre l'exécution afin d'obtenir un haut niveau de fiabilité.

Description: En programmation diversifiée, une spécification de programme donnée est conçue et mise en œuvre *N* fois de différentes manières. Les mêmes valeurs d'entrée sont attribuées aux *N* versions et les résultats produits par les *N* versions sont comparés. Si le résultat est considéré comme valide, il est transmis aux sorties de l'ordinateur.

Une exigence essentielle veut que les *N* versions soient indépendantes les unes des autres dans un certain sens, de sorte qu'elles ne subissent pas toutes une défaillance simultanée ayant la même origine. Dans la pratique, il peut être très difficile de parvenir à l'indépendance des versions et de la démontrer, c'est-à-dire le fondement de l'approche des *N* versions.

Les *N* versions peuvent être exécutées en parallèle sur des ordinateurs séparés, avec l'alternative de pouvoir exécuter toutes les versions sur le même ordinateur et de soumettre les résultats à un vote interne. Différentes stratégies de vote peuvent être utilisées pour les *N* versions selon les exigences de l'application, comme suit.

– Si le système a un état sûr, il est alors possible d'exiger la conformité complète (toutes les *N* versions sont conformes); dans le cas contraire, une valeur de sortie entraînant la mise à l'état sûr du système est utilisée. Pour les systèmes à déclenchement simple, le vote peut être systématiquement orienté dans le sens de la sécurité. Dans ce cas, la réponse sûre consisterait à procéder à un déclenchement lorsque ce dernier est exigé par l'une des versions. Cette approche n'utilise normalement que deux versions (*N*=2).

– Pour les systèmes sans état sûr, des stratégies de vote majoritaire peuvent être utilisées. En cas de non-conformité collective, des approches probabilistes peuvent être utilisées en vue de maximiser les chances de sélectionner la valeur correcte, par exemple, en prenant la valeur moyenne, en figeant temporairement les sorties jusqu'au retour en conformité, etc.

Cette technique n'élimine pas les anomalies résiduelles de conception logicielle, ni les erreurs dans l'interprétation de la spécification, mais elle fournit une mesure de détection et de masquage avant que la sécurité ne soit affectée.

Références:

Modelling software design diversity – a review, B. Littlewood, P. Popov, L. Strigini. ACM Computing Surveys, vol 33, no 2, 2001

The N-Version Approach to Fault-Tolerant Software, A.Avizienis, IEEE Transactions on Software Engineering, vol. SE-11, no. 12 pp.1491-1501, 1985

An experimental evaluation of the assumption of independence in multi-version programming, J.C. Knight, N.G. Leveson. IEEE Transactions on Software Engineering, vol. SE-12, no 1, 1986

In Search of Effective Diversity: a Six Language Study of Fault-Tolerant Flight Control Software. A. Avizienis, M. R. Lyu and W. Schutz. 18th Symposium on Fault-Tolerant Computing, Tokyo, Japan, 27-30 June 1988, IEEE Computer Society Press, 1988, ISBN 0-8186-0867-6

C.3.6 Récupération descendante

NOTE Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Permettre un fonctionnement correct en présence d'une ou plusieurs anomalies.

Description: Lorsqu'une anomalie a été détectée, le système est ramené à un état interne précédent dont la cohérence a déjà été démontrée. Cette méthode nécessite la sauvegarde fréquente de l'état interne au niveau de points de contrôle bien définis. Cela peut être fait globalement (pour la base de données complète) ou par incréments (changements intervenant uniquement entre les points de contrôle). Le système doit alors compenser les changements qui se sont produits pendant ce temps en utilisant la journalisation (vérification à rebours des actions), la compensation (tous les effets de ces changements sont annulés) ou l'interaction externe (manuelle).

Références:

Looking into Compensable Transactions. Jing Li, Huibiao Zhu, Geguang Pu, Jifeng He. In Software Engineering Workshop, 2007. SEW 2007. IEEE, 2007, ISBN 978-0-7695-2862-5

Software Fault Tolerance (Trends in Software, No. 3), M. R. Lyu (ed.), John Wiley & Sons, April 1995, ISBN 0471950688

C.3.7 Mécanismes de récupération d'anomalie par relance

NOTE Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Tenter la récupération fonctionnelle à partir d'une condition d'anomalie détectée, à l'aide de mécanismes de relance.

Description: Suite à une détection d'anomalie ou d'erreur, des tentatives sont effectuées en vue de rétablir la situation en exécutant de nouveau le même code. La récupération par relance peut être aussi complète que de relancer le système ou d'exécuter une procédure de redémarrage ou peut se faire par une petite tâche de replanification et de redémarrage, après dépassement du temps imparti par le logiciel ou après une action de supervision d'une tâche. Les techniques de relance sont communément utilisées pour la récupération des anomalies ou d'erreurs de communication et les conditions de relance peuvent être signalées à partir d'une erreur de protocole de communication (somme de contrôle, etc.) ou à partir d'un dépassement du temps de réponse d'accusé de réception de communication.

Référence:

Reliable Computer Systems: Design and Evaluation, D.P. Siewiorek, R.S. Schwartz. A.K. Peters Ltd., 1998, ISBN 156881092X, 9781568810928

C.3.8 Dégradation « progressive »

NOTE Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Maintenir les fonctions les plus critiques du système disponibles, en dépit des défaillances, par abandon des fonctions les moins critiques.

Description: Cette technique donne la priorité aux différentes fonctions qui doivent être exécutées par le système. La conception garantit qu'en l'absence de ressources suffisantes pour exécuter toutes les fonctions du système, les fonctions présentant la priorité la plus élevée sont exécutées de préférence aux fonctions présentant une priorité moindre. Par exemple, les fonctions de consignation des erreurs et événements peuvent être moins prioritaires que les fonctions de commande du système, ce qui permet de continuer à commander le système en cas de défaillance de la partie matérielle chargée de la consignation des erreurs. De plus, en cas de défaillance de la partie matérielle chargée de la commande du système, sans défaillance toutefois de celle chargée de la consignation des erreurs, c'est cette dernière partie qui assume la fonction de commande.

Cette technique s'applique principalement au matériel mais également au système entier. Elle doit être prise en compte dès le début de la phase de conception.

Références:

Towards the Integration of Fault, Resource, and Power Management, T. Siridakis. In Computer Safety, Reliability, and Security: 23rd International Conference, SAFECOMP 2004. Eds. Maritta Heisel et. al. Springer, 2004, ISBN 3540231765, 9783540231769

Achieving Critical System Survivability Through Software Architectures, J.C. Knight, E.A. Strunk. Springer Berlin / Heidelberg, 2004, 978-ISBN 3-540-23168-4

The Evolution of Fault-Tolerant Computing. Vol. 1 of Dependable Computing and Fault-Tolerant Systems, Edited by A. Avizienis, H. Kopetz and J. C. Laprie, Springer Verlag, 1987, ISBN 3-211-81941-X

Fault Tolerance, Principle and Practices. T. Anderson and P. A. Lee, Vol. 3 of Dependable Computing and Fault-Tolerant Systems, Springer Verlag, 1987, ISBN 3-211-82077-9

C.3.9 Correction d'anomalie en utilisant les techniques d'intelligence artificielle

NOTE 1 Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Etre capable de réagir aux dangers éventuels de manière très souple en utilisant une combinaison de méthodes et de modèles de processus et un certain type d'analyse de sécurité et de fiabilité en ligne.

Description: La prévision des anomalies (tendances obtenues par calcul), la correction des anomalies et les actions de maintenance et de surveillance peuvent être réalisées de manière très efficace sur des voies diversifiées du système en utilisant des systèmes basés sur l'intelligence artificielle (IA), puisque les règles peuvent être directement déduites des spécifications et vérifiées par comparaison avec ces spécifications. Certaines anomalies communes qui sont introduites dans les spécifications, en pensant déjà implicitement à certaines règles de conception et d'implémentation, peuvent être évitées de manière efficace grâce à cette approche, en particulier lorsque l'on applique une combinaison de modèles et de méthodes d'une manière fonctionnelle ou descriptive.

Les méthodes sont sélectionnées de telle manière que les anomalies peuvent être corrigées et les effets des défaillances minimisés en vue d'obtenir l'intégrité de sécurité souhaitée.

NOTE 2 Voir C.3.2 pour un avertissement sur la correction des données erronées, et le point 5 du Tableau A.2 de la CEI 61508-3 pour des recommandations négatives relatives à cette technique.

Référence:

Fault Diagnosis: Models, Artificial Intelligence, Applications. J. Korbicz, J. Koscielny, Z. Kowalczuk, W. Cholewa. Springer, 2004, ISBN 3540407677, 9783540407676

C.3.10 Reconfiguration dynamique

NOTE Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Conserver la fonctionnalité du système en dépit d'une anomalie interne.

Description: L'architecture logique du système doit être telle qu'elle peut être mise en correspondance sur un sous-ensemble des ressources disponibles du système. Cette architecture doit être capable de détecter la défaillance d'une ressource physique et de mettre à nouveau en correspondance l'architecture logique sur les ressources limitées encore en fonctionnement. Bien que le concept soit plus traditionnellement limité à la récupération à partir d'unités matérielles défaillantes, il s'applique également aux unités logicielles défaillantes lorsque la «redondance en exécution» est suffisante pour permettre une relance logicielle, ou lorsqu'il y a suffisamment de données redondantes pour qu'une défaillance individuelle et isolée soit considérée comme peu importante.

Cette technique doit être prise en compte dès la première phase de conception du système.

Référence:

Dynamic Reconfiguration of Software Architectures Through Aspects. C. Costa et. al. Lecture Notes in Computer Science, Volume 4758/2007, Springer Berlin / Heidelberg, 2007, ISBN 978-3-540-75131-1

C.3.11 Sécurité et fonctionnement en temps réel : architecture à déclenchement temporel (TTA)[8]

NOTE 1 Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Composabilité et implémentation transparente de la tolérance aux anomalies dans les systèmes temps réel critiques pour la sécurité dont le comportement est prévisible.

Description: Toutes les activités d'un système TTA sont déclenchées et basées sur la progression d'une base de temps à synchronisation universelle. Un intervalle de temps fixe est assigné au bus à déclenchement temporel pour chaque application, ledit bus comportant les messages échangés entre les tâches de chaque application qui peuvent par conséquent être échangées uniquement selon un calendrier défini. Les activités des systèmes dirigés par les événements sont déclenchées par des événements arbitraires à des points temporels non prévisibles. Les avantages fondamentaux d'un système TTA sont les suivants (se reporter à la référence Scheidler, Heiner et. al.):

- la composabilité qui réduit considérablement l'effort nécessaire à l'essai et à la certification du système;
- l'implémentation transparente de la tolérance aux anomalies qui recommande fortement l'utilisation de l'architecture pour les applications critiques pour la sécurité;
- la disposition d'une base de temps à synchronisation universelle qui facilite la conception des systèmes temps réel répartis.

La communication entre les nœuds s'effectue au moyen du *Protocole à déclenchement temporel TTP/C* (se reporter à la référence Kopetz, Hexel et. al.) selon un calendrier statique, en décidant du moment opportun pour transmettre un message et en déterminant si un message reçu est approprié ou non au module électronique particulier. L'accès au bus est commandé par un schéma *d'Accès Multiple par Répartition dans le Temps (AMRT*) cyclique issu de la notion universelle du temps.

Le protocole TTP/C garantit (se reporter à la référence Rushby) quatre services de base (services essentiels) dans un réseau de nœuds TTA (se reporter à la référence Kopetz, Bauer):

- Transmission de messages déterministe et dans les délais impartis: Transmission de messages du port de sortie de l'élément émetteur aux ports d'entrée des éléments récepteurs dans un délai connu à priori. Un service de transmission tolérant aux anomalies est proposé par un service de communication à déclenchement temporel disponible via l'interface pare-feu temporelle qui élimine la propagation d'erreurs de commande par conception et réduit au minimum le couplage entre les éléments. La transmission des messages dans les délais impartis avec un temps d'attente et une instabilité minimum est fondamentale pour la stabilité de commande des applications en temps réel.
- Synchronisation d'horloge tolérante aux anomalies: Le contrôleur de transmission génère une base de temps universelle synchronisée tolérante aux anomalies (avec une précision de quelques tics d'horloge) qui est transmise au sous-système hôte.
- Diagnostic cohérent des nœuds défaillants (Membership Service): Le contrôleur de transmission informe chaque SRU de l'état de chaque autre SRU contenu dans une grappe, avec un temps d'attente inférieur à une période AMRT.
- Localisation des anomalies graves: Un sous-système hôte (y compris son logiciel) présentant des anomalies malveillantes peut produire des sorties de données erronées, mais ne peut jamais interférer de quelque autre manière que ce soit sur le bon fonctionnement des éléments restants d'une grappe TTP/C. Le comportement à déclenchement temporel du contrôleur de transmission garantit tout silence sur défaillance dans le domaine temporel.

---

8 TTA = *Time-Triggered Architecture.*

NOTE 2 Les autres protocoles à déclenchement temporel sont FlexRay et TT-Ethernet (Ethernet à déclenchement temporel).

Références:

Time-Triggered Architecture (TTA). C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, C. Temple. In Advances in Information Technologies: The Business Challenge, ed. J-Y. Roger. IOS Press, 1998, ISBN 9051993854, 9789051993851

A Synchronisation Strategy for a TTP/C Controller. H. Kopetz, R. Hexel, A. Krueger, D. Millinger, A. Schedl. SAE paper 960120, Application of Multiplexing Technology SP 1137, Detroit, SAE Press, Warrendale, 1996

Time-Triggered Architecture. H. Kopetz, G. Bauer. Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software, October 2002

An Overview of Formal Verification for the Time-Triggered Architecture. J. Rushby: Invited paper, Oldenburg, Germany, September 9-12, 2002. Proceedings FTRTFT 2002, Springer LNCS 2469, 2002, ISBN 978-3-540-44165-6

C.3.12 UML (langage de modélisation unifié)

NOTE Cette technique/mesure est mentionnée dans le Tableau B.7 de la CEI 61508-3.

But: Fournir un ensemble exhaustif de notations pour la modélisation du comportement souhaité de systèmes complexes.

Description: Le langage UML est, comme son nom l'indique, un ensemble d'exigences et de notations de conception, destiné à fournir une aide exhaustive au développement du logiciel. Certaines parties du langage UML sont basées sur des notations initiales introduites dans d'autres méthodes (telles que les diagrammes de séquence de système et les diagrammes de transition d'état) et d'autres notations uniques au langage UML. Le langage UML est fortement polarisé sur des concepts orientés objets bien que certaines notations puissent être utilisées sans recourir à une programmation orientée objets. Le langage UML est pris en charge par de nombreux outils CASE disponibles dans le commerce, dont bon nombre sont en mesure de générer des codes automatiquement à partir des modèles UML.

Les notations UML les plus couramment applicables à la spécification et à la conception des systèmes relatifs à la sécurité, sont les suivantes:

– diagrammes de classe
– cas d'utilisation
– diagrammes d'activité
– diagrammes de transition d'état (diagrammes d'état - Statecharts)
– diagrammes de séquence de système

D'autres notations UML s'appliquent à l'expression de la conception de l'architecture du logiciel (structure du logiciel) mais ne sont pas indiquées de manière spécifique dans le présent document.

Les diagrammes de transition d'état sont décrits en B.2.3.2 et les diagrammes de séquence de système en C.2.14. Les autres notations sont décrites dans les trois paragraphes suivants.

C.3.12.1 Diagrammes de classe

Les diagrammes de classe définissent les classes d'objets que doit traiter le logiciel. Ils sont basés sur les diagrammes entité-relation-attribut antérieurs mais sont adaptés à la conception orientée objets. Chaque classe (dont il existe une ou plusieurs instances connues au moment

de l'exécution) est représentée par un rectangle et les diverses relations entre les classes sont illustrées par des lignes ou des flèches. Les opérations ou méthodes proposées par chaque classe, et leurs attributs de données, peuvent être ajoutés au diagramme. Les relations qu'il est possible de représenter sont les relations de référence avec leur cardinalité (une instance de la classe A peut faire référence à une ou plusieurs instances de la classe B) et les relations de spécialisation (la Classe X est un affinement de la classe Y) avec d'autres méthodes et attributs possibles. Plusieurs héritages peuvent être illustrés.

C.3.12.2 Cas d'utilisation

Les cas d'utilisation fournissent une description textuelle du comportement souhaité du système en réponse à un scénario particulier, généralement du point de vue des acteurs externes, y compris les utilisateurs humains du système et les systèmes externes. Il est possible d'utiliser d'autres sous-scénarios dans un cas d'utilisation donné pour représenter un comportement facultatif, notamment dans les cas de réponse aux erreurs. Un ensemble de cas d'utilisation est développé pour fournir une spécification suffisamment complète des exigences du système. Les cas d'utilisation peuvent constituer le point de départ de modèles de développement plus rigoureux tels que les diagrammes de séquence de système et les diagrammes d'activité.

Les diagrammes de cas d'utilisation permettent une représentation schématique du système et des acteurs impliqués dans les cas d'utilisation, mais ils ne sont pas rigoureux et ne considèrent pour la spécification que le texte du cas d'utilisation.

C.3.12.3 Diagrammes d'activité

Un diagramme d'activité illustre la séquence prévue des actions réalisées par un composant logiciel (souvent un objet dans une conception orientée objets), y compris le comportement séquentiel et itératif (certains aspects s'apparentent fortement à un organigramme). Les diagrammes d'activité permettent cependant de décrire en parallèle les actions d'un certain nombre de composants, avec les interactions entre les composants représentées par des flèches sur le diagramme. Les points de synchronisation où une activité doit attendre une ou plusieurs entrées provenant d'autres activités avant de pouvoir être exécutée, sont représentés par un symbole similaire à un nœud de réseau de Pétri.

Référence:

ISO/CEI 19501:2005, *Technologies de l'information – Traitement distribué ouvert – Langage de modélisation unifié (UML), version 1.4.2*

C.4 Outils de développement et langages de programmation

C.4.1 Langages de programmation fortement typés

NOTE Cette technique/mesure est mentionnée dans le Tableau A.3 de la CEI 61508-3.

But: Réduire la probabilité d'anomalies en utilisant un langage qui permette un contrôle de haut niveau par le compilateur.

Description: Lorsqu'un langage de programmation fortement typé est compilé, de nombreux contrôles sont effectués en ce qui concerne le mode d'utilisation des types de variable comme, par exemple, dans les appels de procédures et l'accès aux données externes. La compilation échouera et un message d'erreur sera émis pour chaque utilisation non conforme aux règles prédéfinies.

De tels langages permettent normalement de définir des types de données définis par l'utilisateur à partir de types de données liés à un langage de base (comme les nombres entiers ou réels). Ces types peuvent alors être utilisés exactement de la même façon que le type de base. Des contrôles stricts sont imposés afin d'assurer que le type correct est utilisé.

Ces contrôles sont imposés au programme entier, même si celui-ci a été réalisé à partir d'unités compilées séparément. Les contrôles permettent également d'assurer que le nombre et le type d'arguments des procédures sont bien adaptés, même lorsqu'ils sont référencés à partir de modules logiciels compilés séparément.

Les langages fortement typés supportent en général d'autres aspects de bonne pratique de génie logiciel tels que des structures de contrôle facilement analysables (par exemple si.. alors.. sinon, faire.. pendant, etc.) qui conduisent à des programmes bien structurés.

Des exemples types de langages fortement typés sont les langages Pascal, Ada et Modula 2.

Référence:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

C.4.2 Sous-ensembles de langage

NOTE Cette technique/mesure est mentionnée dans le Tableau A.3 de la CEI 61508-3.

But: Réduire la probabilité d'introduction d'anomalies de programmation et augmenter la probabilité de détection des anomalies restantes.

Description: Le langage est examiné afin de déterminer les structures de programmation qui sont soit sujettes aux erreurs soit difficiles à analyser, par exemple en utilisant des méthodes d'analyse statique. Un sous-ensemble de langage est alors défini, qui exclut ces structures.

Références:

Practical Experiences of Safety- and Security-Critical Technologies, P. Amey, A.J. Hilton. Ada User Journal, June, 2004

Safer C: Developing Software for High-integrity and Safety-critical Systems. L. Hatton, McGraw-Hill, 1994, ISBN 0077076400, 9780077076405

Requirements for programming languages in safety and security software standard. B. A. Wichmann. Computer Standards and Interfaces. Vol. 14, pp 433-441, 1992

C.4.3 Outils certifiés et traducteurs certifiés

NOTE Cette technique/mesure est mentionnée dans le Tableau A.3 de la CEI 61508-3.

But: Des outils sont nécessaires pour assister les développeurs au cours des différentes phases de développement d'un logiciel. Il convient, dans toute la mesure du possible, que les outils soient certifiés de manière à assurer un certain niveau de fiabilité en ce qui concerne la conformité des sorties.

Description: La certification d'un outil sera généralement réalisée par un organisme indépendant, souvent national, à l'aide de critères déterminés de manière indépendante, typiquement dans des normes nationales ou internationales. Il convient, idéalement, que les outils utilisés dans toutes les phases de développement (spécification, conception, codage, essai et validation) et ceux utilisés pour la gestion de configuration soient soumis à la certification.

Actuellement, seuls les compilateurs (traducteurs) sont soumis régulièrement aux procédures de certification; ces procédures sont établies par les organismes de certification nationaux et permettent de soumettre à essai les compilateurs (traducteurs) par rapport aux normes internationales telles que celles relatives aux langages Ada et Pascal.

Il est important de noter que les outils certifiés et les traducteurs certifiés sont en général certifiés uniquement par rapport à leur propre langage ou leurs propres normes de processus. Ils ne sont en général en aucune façon certifiés en ce qui concerne la sécurité.

Références:

The certification of software tools with respect to software standards, P. Bunyakiati et al. In IEEE International Conference on Information Reuse and Integration, IRI 2007, IEEE, 2007, ISBN 1-4244-1500-4

Certified Testing of C Compilers for Embedded Systems. O. Morgan. In: 3rd Institution of Engineering and Technology Conference on Automotive Electronics. IEEE, 2007, ISBN 978-0-86341-815-0

The Ada Conformity Assessment Test Suite (ACATS), version 2.5, Ada Conformity Assessment Authority, http://www. ic.org/compilerstesting.html , Apr. 2002

C.4.4 Outils et traducteurs : confiance accrue résultant de l'utilisation

NOTE Cette technique/mesure est mentionnée dans le Tableau A.3 de la CEI 61508-3.

But: Eviter des difficultés dues aux défaillances du traducteur susceptibles de se produire pendant le développement, la vérification et la maintenance d'un progiciel.

Description: Utilisation d'un traducteur pour lequel il n'existe pas de preuve de fonctionnement incorrect au cours de nombreux projets antérieurs. Il convient d'éviter l'utilisation de traducteurs sans expérience d'exploitation ou comportant des anomalies graves connues à moins qu'il n'existe quelque autre attestation de fonctionnement correct (voir par exemple C.4.4.1).

Dans le cas où le traducteur a présenté de petites déficiences, les structures de langage correspondantes sont notées et soigneusement évitées au cours d'un projet relatif à la sécurité.

Une procédure différente de cette procédure de travail consiste à limiter l'utilisation du langage à ses seuls éléments communément utilisés.

La présente recommandation est basée sur l'expérience acquise au cours de nombreux projets. Il a été démontré que les traducteurs immatures constituent un sérieux handicap pour le développement de tout logiciel. Ils rendent généralement impossible le développement des logiciels relatifs à la sécurité.

Il est également connu qu'il n'existe actuellement aucune méthode permettant de démontrer la conformité de toutes les parties d'un outil ou d'un traducteur.

C.4.4.1 Comparaison du programme source et du code exécutable

But: Vérifier que les outils utilisés pour produire une image de la PROM n'ont introduit aucune erreur dans l'image de la PROM.

Description: L'image de la PROM est soumise à un processus de rétro-ingénierie pour obtenir les modules «objets» constitutifs. Ces modules «objets» sont soumis à un processus de rétro-ingénierie pour obtenir des fichiers en langage assembleur. En utilisant des techniques appropriées, les fichiers en langage assembleur régénérés sont comparés avec les fichiers source réels utilisés à l'origine pour définir la PROM.

L'avantage majeur de cette technique est que les outils (compilateurs, éditeurs de liens, etc.) utilisés pour produire l'image de la PROM ne nécessitent pas d'être validés pour tous les

programmes. Cette technique vérifie que le fichier source utilisé pour le système relatif à la sécurité considéré est correctement transformé.

Références:

Demonstrating Equivalence of Source Code and PROM Contents. D. J. Pavey and L. A. Winsborrow. The Computer Journal Vol. 36, no. 7, 1993

Formal demonstration of equivalence of source code and PROM Contents: an industrial example. D. J. Pavey and L. A. Winsborrow. Mathematics of Dependable Systems, Ed. C. Mitchell and V. Stavridou, Clarendon Press, 1995, ISBN 0-198534-91-4

Assuring Correctness in a Safety Critical Software Application. L. A. Winsborrow and D. J. Pavey. High Integrity Systems, Vol. 1, no. 5, pp 453-459, 1996

C.4.5 Langages de programmation appropriés

NOTE Cette technique/mesure est mentionnée dans le Tableau A.3 de la CEI 61508-3.

But: Prendre en compte, dans toute la mesure du possible, les exigences de la présente norme internationale, en particulier en ce qui concerne la programmation défensive, le typage fort, la programmation structurée et éventuellement les assertions. Il convient que le langage de programmation choisi aboutisse à un code facilement vérifiable avec un minimum d'effort et facilite le développement, la vérification et la maintenance du programme.

Description: Il convient que le langage soit complètement et clairement défini. Il convient que le langage soit orienté vers l'utilisateur ou le problème plutôt que vers le processeur/la plate-forme de la machine. Les langages communément utilisés ou leurs sous-ensembles sont préférables aux langages spécialisés.

En plus des caractéristiques déjà mentionnées, il convient que le langage permette

– une structure par blocs;
– le contrôle du temps de traduction; et
– le contrôle du type d'exécution et des limites de tableaux.

Il convient que le langage favorise

– l'utilisation de modules logiciels gérables et de petite taille;
– la restriction d'accès aux données dans des modules logiciels spécifiques;
– la définition de sous-ensembles de variables; et
– tout autre type de constructions de limitation des erreurs.

Si le fonctionnement sûr du système dépend de contraintes temps réel, il convient alors que le langage permette également le traitement des exceptions/interruptions.

Il est souhaitable que le langage soit supporté par un traducteur approprié, des bibliothèques appropriées des modules logiciels préexistants, un programme de mise au point et des outils concernant à la fois le contrôle et le développement des versions.

Actuellement, au moment de l'établissement de la présente norme, on ne sait pas encore s'il est préférable d'utiliser des langages orientés objet ou d'autres langages conventionnels.

Les éléments qui rendent toute vérification difficile et dont il convient par conséquent d'éviter l'utilisation sont les suivants:

– les branchements inconditionnels, à l'exception des appels de sous-programmes;
– la récursion;

– les pointeurs, segments de mémoire ou tout type de variables dynamiques ou d'objets;
– la gestion des interruptions au niveau du code source;
– les entrées ou sorties multiples de boucles, blocs ou sous-programmes;
– l'initialisation ou la déclaration implicite de variables;
– les enregistrements de variantes et les équivalences; et
– les paramètres de procédure.

Les langages de bas niveau, en particulier les langages assembleurs, présentent des problèmes liés au fait qu'ils sont par nature, orientés vers le processeur/la plate-forme de la machine.

L'une des propriétés souhaitables d'un langage réside dans le fait qu'il convient que sa conception et son utilisation génèrent des programmes dont l'exécution est prévisible. Pour un langage de programmation défini de façon appropriée, il existe un sous-ensemble de ce langage qui assure que l'exécution d'un programme est prévisible. Ce sous-ensemble ne peut pas (en général) être déterminé statiquement, bien que de nombreuses contraintes statiques puissent aider à assurer une exécution prévisible. Cela nécessiterait typiquement de démontrer que les indices de matrices sont définis dans des limites données, et qu'aucun dépassement numérique ne peut apparaître, etc.

Le Tableau C.1 donne les recommandations applicables aux langages de programmation spécifiques.

Références:

Concepts in Programming Languages. J. C. Mitchell. Cambridge University Press, 2003, ISBN 0521780985, 9780521780988

CEI 60880:2006, *Centrales nucléaires de puissance – Instrumentation et contrôle-commande importants pour la sûreté – Aspects logiciels des systèmes programmés réalisant des fonctions de catégorie A*

CEI 61131-3:2003, *Programmable controllers – Part 3: Programming languages (disponible en anglais seulement)*

ISO/CEI 1539-1:2004, *Technologies de l'information – Langages de programmation – Fortran – Partie 1: Langage de base*

ISO/CEI 7185:1990, *Technologies de l'information – Langages de programmation – Pascal*

ISO/CEI 8652:1995, *Technologies de l'information – Langages de programmation – Ada*

ISO/CEI 9899:1999, *Langages de programmation – C*

ISO/CEI 10206:1991, *Technologies de l'information – Langages de programmation – Pascal étendu*

ISO/CEI 10514-1:1996, *Technologies de l'information – Langages de programmation – Partie 1: Modula-2, langage de base*

ISO/CEI 10514-3:1998, *Technologies de l'information – Langages de programmation – Partie 3: Modula 2 orienté objet*

ISO/CEI 14882:2003, *Langages de programmation – C++*

ISO/CEI/TR 15942:2000, *Technologies de l'information – Langages de programmation – Guide pour l'emploi du langage de programmation Ada dans les systèmes de haute intégrité*

Tableau C.1 – Recommandations applicables aux langages de programmation spécifiques

	Langage de programmation	SIL1	SIL2	SIL3	SIL4
1	ADA	HR	HR	R	R
2	ADA avec sous-ensemble	HR	HR	HR	HR
3	Java	NR	NR	NR	NR
4	Java avec sous-ensemble (y compris avec ou sans récupération de l'espace mémoire ne provoquant pas l'arrêt du code d'application pendant une durée significative). Voir Annexe G pour des recommandations sur l'utilisation des installations orientées objets	R	R	NR	NR
5	PASCAL (voir Note 1)	HR	HR	R	R
6	PASCAL avec sous-ensemble	HR	HR	HR	HR
7	FORTRAN 77	R	R	R	R
8	FORTRAN 77 avec sous-ensemble	HR	HR	HR	HR
9	C	R	–	NR	NR
10	C avec sous-ensemble et règle de codage, et utilisation d'outils d'analyse statique	HR	HR	HR	HR
11	C++ (voir Annexe G pour des recommandations sur l'utilisation des installations orientées objets)	R	–	NR	NR
12	C++ avec sous-ensemble et règle de codage, et utilisation d'outils d'analyse statique (voir Annexe G pour des recommandations sur l'utilisation des installations orientées objets)	HR	HR	HR	HR
13	Assembleur	R	R	–	–
14	Sous-ensemble assembleur avec règle de codage	R	R	R	R
15	Diagrammes à contacts	R	R	R	R
16	Diagramme à contacts avec sous-ensemble de langage défini	HR	HR	HR	HR
17	Diagramme de blocs fonctionnels	R	R	R	R
18	Diagramme de blocs fonctionnels avec sous-ensemble de langage défini	HR	HR	HR	HR
19	Texte structuré	R	R	R	R
20	Texte structuré avec sous-ensemble de langage défini	HR	HR	HR	HR
21	Diagramme fonctionnel séquentiel	R	R	R	R
22	Diagramme fonctionnel séquentiel avec sous-ensemble de langage défini	HR	HR	HR	HR
23	Liste d'instructions	R	–	NR	NR
24	Liste d'instructions avec sous-ensemble de langage défini	HR	R	R	R

NOTE 1 Les recommandations R, HR et – sont explicitées dans l'Annexe A de la CEI 61508-3.

NOTE 2 Le logiciel système comprend le système d'exploitation, les pilotes, les fonctions intégrées et les modules logiciels faisant partie intégrante du système. Le logiciel est normalement fourni par le vendeur du système relatif à la sécurité. Il convient que le sous-ensemble de langage soit soigneusement choisi afin d'éviter les structures complexes pouvant conduire à des anomalies d'implémentation. Il convient que des contrôles soient réalisés afin de vérifier l'utilisation correcte du sous-ensemble de langage.

NOTE 3 Le logiciel d'application est un logiciel qui a été développé pour une application relative à la sécurité spécifique. Dans nombre de cas, ce logiciel est développé par l'utilisateur final ou par le fournisseur d'application. Lorsque la même recommandation s'applique à de nombreux langages de programmation, il convient que le développeur choisisse celui qui est couramment utilisé par le personnel dans l'industrie ou dans l'unité de production concernée. Il convient que le sous-ensemble de langage soit soigneusement choisi afin d'éviter les structures complexes pouvant conduire à des anomalies d'implémentation. Il convient que des contrôles soient réalisés afin de vérifier l'utilisation correcte du sous-ensemble de langage.

NOTE 4 Tout langage spécifique ne figurant pas dans le présent tableau ne doit pas être considéré comme un langage exclu. Il convient qu'il soit conforme à la présente norme internationale.

NOTE 5 Il existe un certain nombre d'extensions au langage Pascal, y compris Pascal libre. Les références à Pascal comprennent ces extensions.

NOTE 6 Java est conçu pour avoir un « nettoyeur » pendant le temps d'exécution. Il est possible de définir un sous-ensemble de Java ne nécessitant pas de récupération de l'espace mémoire. Certaines implémentations de Java fournissent une récupération de l'espace mémoire graduelle qui permet de récupérer de la mémoire libre au fur et à mesure de l'exécution du programme et empêche l'arrêt de l'exécution pendant une certaine durée lorsque la mémoire n'est plus disponible. Il convient que les applications matérielles temps réel n'utilisent aucune forme de récupération de l'espace mémoire.

NOTE 7 Si l'implémentation de Java nécessite un interpréteur de temps d'exécution du code intermédiaire de Java, l'interpréteur doit alors être considéré comme partie intégrante du logiciel de sécurité et traité conformément aux exigences de la CEI 61508-3.

NOTE 8 Pour les entrées 15 à 24, voir la CEI 61131-3.

C.4.6 Génération logicielle automatique

NOTE Cette technique/mesure est mentionnée dans le Tableau A.2 de la CEI 61508-3.

But: Automatiser les tâches les plus sujettes à des erreurs de l'implémentation logicielle.

Description: La conception du système est décrite par un modèle (une spécification exécutable) à un niveau d'abstraction plus élevé que le code exécutable classique. Un générateur de code transforme automatiquement le modèle en forme exécutable. L'objectif est d'améliorer la qualité du logiciel en éliminant les tâches de codage manuelles sujettes à des erreurs. Un autre avantage potentiel de cette fonction réside dans le fait que des conceptions plus complexes peuvent être réalisées au niveau d'abstraction plus élevé.

Références:

Embedded Software Generation from System Level Design Languages, H Yu, R. Domer, D. Gajski. In "ASP-DAC 2004: Proceedings of the ASP-Dac 2004 Asia and South Pacific Design Automation Conference, 2004", IEEE Circuits and Systems Society. IEEE, 2004, ISBN 0780381750, 9780780381759

Transforming Process Algebra Models into UML State Machines: Bridging a Semantic Gap?. M.F. van Amstel et. al. In Theory and Practice of Model Transformations: First International Conference, ICMT". ed. A. Vallecillo. Springer, 2008, ISBN 3540699260, 9783540699262

C.4.7 Outils de gestion des essais et d'automatisation

NOTE Cette technique/mesure est mentionnée dans le Tableau A.5 de la CEI 61508-3.

But: Encourager une approche approfondie systématique pour les essais du logiciel et du système.

Description: L'utilisation d'outils de support appropriés contribue à l'automatisation des tâches réclamant la plus forte intensité de main d'œuvre et les plus sujettes à des erreurs dans le cadre du développement des systèmes et contribue à une approche systématique de la gestion des essais. La fonction d'aide encourage une approche plus approfondie tant des essais normaux que des essais de régression.

Référence:

Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. R.Black, John Wiley and Sons, 2002, ISBN 0471223980, 9780471223986

C.5 Vérification et modification

C.5.1 Essai probabiliste

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.5, C.15, A.7 et C.17 de la CEI 61508-3.

But: Obtenir une indication quantitative concernant les propriétés de fiabilité du logiciel examiné.

Description: La méthode produit une estimation statistique de la fiabilité du logiciel. Cette indication quantitative peut prendre en compte les niveaux de confiance et de signification associés et peut générer

– une probabilité de défaillance par sollicitation;
– une probabilité de défaillance pendant une certaine période de temps; et
– une probabilité de confinement des erreurs.

A partir de ces indications, d'autres paramètres peuvent être obtenus, tels que:

– la probabilité d'exécution sans défaillances;
– la probabilité de survie;
– la disponibilité;
– le MTBF ou le taux de défaillance; et
– la probabilité d'exécution sûre.

Les considérations probabilistes sont basées soit sur un essai probabiliste soit sur l'expérience en exploitation. Normalement, le nombre de cas d'essai ou de cas observés en exploitation est très élevé. Typiquement, l'essai du mode de fonctionnement sur sollicitation nécessite un temps beaucoup plus court que pour le mode de fonctionnement en continu.

Les outils d'essai automatisés sont normalement utilisés pour fournir les données d'essai et superviser les sorties d'essai. Des essais importants sont réalisés sur les grands ordinateurs centraux avec les équipements périphériques de simulation de processus appropriés. Les données d'essai sont sélectionnées d'un point de vue matériel à la fois systématique et aléatoire. Par exemple, l'essai global utilise un profil de données d'essai alors que la sélection aléatoire peut être utilisée pour la réalisation d'essais individuels détaillés.

L'utilisation de bancs d'essai logiciels, ainsi que l'exécution et la supervision d'essais individuels, sont déterminées en fonction des objectifs détaillés des essais, comme décrit ci-dessus. D'autres conditions importantes résultent des conditions mathématiques préalables qui doivent être remplies si l'évaluation de l'essai doit satisfaire à l'objectif prévu de ce dernier.

Les chiffres probabilistes concernant le comportement de tout objet en essai peuvent être également obtenus d'après l'expérience en exploitation. Sous réserve que les mêmes

conditions soient remplies, les mêmes méthodes mathématiques que celles utilisées pour l'évaluation des résultats d'essai peuvent être appliquées.

Dans la pratique, il est très difficile de démontrer des niveaux de fiabilité extrêmement élevés à l'aide de ces techniques.

Références:

A discussion of statistical testing on a safety-related application. S Kuball, J H R May, Proc IMechE Vol. 221 Part O: J. Risk and Reliability, Institution of Mechanical Engineers, 2007

Estimating the Probability of Failure when Testing Reveals No Failures, W.K. Miller, L.J. Morell, et al.. IEEE Transactions on Software Engineering, VOl. 18, NO.1, pp33-43, January 1992

Reliability estimation from appropriate testing of plant protection software, J. May, G. Hughes, A.D. Lunn. IEE Software Engineering Journal, v10 n6 pp March 206-218, Nov 1995 (ISSN: 0268-6961)

Validation of ultra high dependability for software based systems, B. Littlewood and L. Strigini. Comm. ACM 36 (11), 69-80, 1993.

C.5.2 Enregistrement et analyse de données

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.5 et A.8 de la CEI 61508-3.

But: Documenter toutes les données, décisions et justifications associées du projet de logiciel afin de faciliter la vérification, la validation, l'évaluation et la maintenance.

Description: Une documentation détaillée est maintenue pendant un projet, pouvant inclure

– les essais réalisés sur chaque module logiciel;

– les décisions et leurs justifications;

– les problèmes et leurs solutions.

Cette documentation peut être analysée au cours et à la fin du projet en vue d'obtenir une grande variété d'informations. En particulier, l'enregistrement des données est très important pour la maintenance des systèmes informatiques étant donné que la raison de certaines décisions prises au cours du projet de développement n'est pas toujours connue des ingénieurs de maintenance.

Référence:

Dependability of Critical Computer Systems 2. F. J. Redmill, Elsevier Applied Science, 1989, ISBN 1851663819, 9781851663811

C.5.3 Essai d'interface

NOTE Cette technique/mesure est mentionnée dans le Tableau A.5 de la CEI 61508-3.

But: Détecter les erreurs dans les interfaces de sous-programmes.

Description: Plusieurs niveaux de détail ou de complétude des essais sont possibles. Les niveaux les plus importants sont les essais pour

– toutes les variables d'interface à leurs valeurs extrêmes;

– toutes les variables d'interface considérées individuellement à leurs valeurs extrêmes avec les autres variables d'interface à leurs valeurs normales;

– toutes les valeurs du domaine de chaque variable d'interface avec les autres variables d'interface à leurs valeurs normales;

– toutes les valeurs de toutes les variables combinées (ceci peut être réalisé uniquement sur les petites interfaces);

– les conditions d'essai spécifiées concernant chaque appel de chaque sous-programme.

Ces essais sont particulièrement importants si les interfaces ne contiennent pas d'assertions permettant de détecter les valeurs de paramètres incorrectes. Ils sont également importants après l'élaboration de nouvelles configurations de sous-programmes préexistants.

C.5.4 Analyse des valeurs aux limites

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2, B.3 et B.8 de la CEI 61508-3.

But: Détecter les erreurs logicielles aux limites ou frontières des paramètres.

Description: Le domaine d'entrée du programme est divisé en un certain nombre de classes d'entrée d'après la relation d'équivalence (voir C.5.7). Il convient que les essais couvrent les frontières et les valeurs extrêmes des classes. Les essais vérifient que les frontières du domaine d'entrée de la spécification coïncident avec celles du programme. L'utilisation de la valeur zéro dans le cas d'une traduction directe aussi bien qu'indirecte est souvent sujette à des erreurs et nécessite une attention particulière:

– diviseur par zéro;

– caractères ASCII blancs;

– élément de liste ou pile vide;

– matrice pleine;

– entrée de table à zéro.

Normalement, les limites pour les entrées correspondent directement avec celles de la plage de sortie. Il convient que les cas d'essai soient écrits de manière à forcer la sortie à atteindre ses valeurs limites. Considérer également s'il est possible ou non de spécifier un cas d'essai pour lequel la sortie est amenée à dépasser les valeurs limites de la spécification.

Si la sortie est une séquence de données, par exemple une table imprimée, il convient de porter une attention toute particulière aux premier et dernier éléments ainsi qu'aux listes contenant zéro, un et deux éléments.

Références:

The Art of Software Testing, Second Edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

C.5.5 Estimation des erreurs

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2 et B.8 de la CEI 61508-3.

But: Eliminer les erreurs de programmation communes.

Description: L'expérience acquise durant les essais et l'intuition alliée à la connaissance et à la curiosité portée au système soumis à essai peut contribuer à ajouter quelques cas d'essais hors catégorie au jeu de cas d'essais prévu.

Les valeurs spéciales ou les combinaisons de valeurs peuvent être sujettes à des erreurs. Certains cas d'essais intéressants peuvent être extraits des listes de contrôle d'inspection. La

robustesse du système peut être également mise en question. Par exemple: est-il possible d'appuyer trop rapidement ou trop souvent sur les boutons de la face avant ? Que se passe-t-il lorsque l'on appuie simultanément sur deux boutons ?

Référence:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

C.5.6 Implantation d'erreurs

NOTE Cette technique/mesure est mentionnée dans le Tableau B.2 de la CEI 61508-3.

But: S'assurer qu'un ensemble de cas d'essais est adéquat.

Description: Certains types d'erreurs connues sont insérés (implantés) dans le programme, puis le programme est exécuté avec les cas d'essai dans les conditions d'essai. Si seules quelques erreurs implantées sont découvertes, le jeu de cas d'essai n'est pas adéquat. Le rapport entre les erreurs implantées découvertes et le nombre total d'erreurs implantées correspond à une estimation du rapport entre les erreurs réelles découvertes et le nombre total d'erreurs. Ceci permet d'estimer le nombre d'erreurs restantes et, par conséquent, l'effort d'essai restant.

$$\frac{\text{Erreurs implantées découvertes}}{\text{Nombre total derreurs implantées}} = \frac{\text{Erreurs réelles découvertes}}{\text{Nombre total derreurs réelles}}$$

La détection de toutes les erreurs implantées peut signifier que le jeu de cas d'essai est adéquat ou que les erreurs implantées étaient trop faciles à découvrir. Les limites de la méthode sont celles qui permettent d'obtenir un résultat utilisable, les types d'erreurs ainsi que la position des implantations devant refléter la distribution statistique des erreurs réelles.

Références:

Software Fault Injection: Inoculating Programs Against Errors. J. Voas, G. McGraw. Wiley Computer Pub., 1998, ISBN 0471183814, 9780471183815

Faults, Injection Methods, and Fault Attacks. Chong Hee Kim, Jean-Jacques Quisquater, IEEE Design and Test of Computers, vol. 24, no. 6, pp . 544-545, Nov., 2007

Fault seeding for software reliability model validation. A. Pasquini, E. De Agostino. Control Engineering Practice, Volume 3, Issue 7, July 1995. Elsevier Science Ltd

C.5.7 Classes d'équivalence et essai des partitions d'entrée

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2 et B.3 de la CEI 61508-3.

But: Soumettre à essai le logiciel de manière adéquate à l'aide d'un minimum de données d'essai. Les données d'essai sont obtenues en sélectionnant les partitions du domaine d'entrée requises pour soumettre à essai le logiciel.

Description: Cette stratégie d'essai est basée sur la relation d'équivalence des entrées, qui détermine une partition du domaine d'entrée.

Les cas d'essai sont sélectionnés dans le but de couvrir toutes les partitions précédemment spécifiées. Un cas d'essai au moins est pris sur chaque classe d'équivalence.

Les deux possibilités de base pour la partition des entrées sont les suivantes

– les classes d'équivalence déduites de la spécification - l'interprétation de la spécification peut être orientée "entrée", par exemple les valeurs sélectionnées sont traitées de la même manière, ou orientée "sortie", par exemple le jeu de valeurs conduit au même résultat fonctionnel;
– les classes d'équivalence peuvent être déduites de la structure interne du programme - les résultats des classes d'équivalence sont déterminés à partir de l'analyse statique du programme, par exemple le jeu de valeurs conduisant au même trajet en cours d'exécution.

Références:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

Static Analysis and Software Assurance. D. Wagner, Lecture Notes in Computer Science, Volume 2126/2001, Springer, 2001, ISBN 978-3-540-42314-0

C.5.8 Essais basés sur la structure

NOTE Cette technique/mesure est mentionnée dans le Tableau B.2 de la CEI 61508-3.

But: Appliquer des essais servant à soumettre à essai certains sous-ensembles de la structure du programme.

Description: Fondé sur l'analyse du programme, un jeu de données d'entrée est choisi de manière à soumettre à essai un pourcentage élevé (et souvent un pourcentage cible prédéfini) du code de programme. Les mesures de couverture du code varieront comme suit en fonction du niveau de rigueur requis. Dans tous les cas, il convient de viser 100 % des métriques de couverture sélectionnées; dans le cas contraire, il convient de documenter dans le rapport d'essai les raisons pour lesquelles il n'est pas possible d'assurer 100 % de la couverture (par exemple, le seul code défensif qu'il est possible de saisir en cas de problème matériel). Les quatre premières techniques de la liste suivantes sont tout particulièrement indiquées dans les recommandations du Tableau B.3 de la CEI 61508-3 et sont largement prises en charge par les outils d'essai; il est également possible de tenir compte des autres techniques.

– **Couverture de point d'entrée (graphe d'appel)**: assure que chaque sous-programme (sous-programme ou fonction) a été appelé au moins une fois (il s'agit de la mesure de couverture de structure la moins rigoureuse).

  NOTE Dans les langages orientés objets, plusieurs sous-programmes de même nom peuvent s'appliquer à différentes variantes d'un type polymorphe (sous-programmes d'annulation) pouvant être invoquées par distribution dynamique. Dans ces cas, il convient de soumettre à essai chaque sous-programme d'annulation de ce type.

– **Instructions:** assurent que toutes les instructions du code ont été exécutées au moins une fois.
– **Branchements:** il convient de vérifier les deux côtés de chaque branchement. Ceci peut être impossible pour certains types de codes défensifs.
– **Conditions composées:** chaque condition d'un branchement conditionnel composé (c'est-à-dire avec liaison ET/OU) est soumise à essai. Voir MCDC (modified condition decision coverage, réf. DO178B).

– **LCSAJ:** une séquence de code linéaire avec branchement est une séquence linéaire d'instructions de codage, y compris les instructions conditionnelles, terminée par un branchement. Nombre de sous-cheminements potentiels seront indisponibles à cause des contraintes imposées aux données d'entrée par l'exécution du code précédent.
– **Flux de données:** les cheminements d'exécution sont sélectionnés sur la base de l'utilisation des données; par exemple, un cheminement servant à la fois à l'écriture et à la lecture de la même variable.
– **Chemin de base:** un des cheminements d'un ensemble minimal de cheminements finis qui vont du début à la fin, de telle sorte que tous les arcs y soient inclus. (Les combinaisons de cheminements qui se superposent dans cet ensemble de base peuvent constituer tout cheminement dans la partie correspondante du programme). Les essais de tous les cheminements de base ont démontré leur efficacité à détecter les erreurs.

Références:

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799"

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

RTCA, Inc. document DO-178B and EUROCAE document ED-12B, *Software Considerations in Airborne Systems and Equipment Certification*, dated December 1, 1992

C.5.9 Analyse du flux de commande

NOTE Cette technique/mesure est mentionnée dans le Tableau B.8 de la CEI 61508-3.

But: Détecter les structures de programme faibles et potentiellement incorrectes.

Description: L'analyse du flux de commande est une technique d'essai statique pour identifier des zones suspectes du code qui ne suivent pas une bonne pratique de programmation. Le programme est analysé, un graphe orienté étant par ailleurs produit et pouvant également être analysé pour déterminer l'existence des éléments suivants:

– un code inaccessible, par exemple des branchements inconditionnels partant des blocs de code inaccessibles;
– un code noué. Un code bien structuré comporte un graphe de commande qui peut être réduit à un simple sommet par réductions successives. A l'inverse, un code faiblement structuré peut n'être réduit qu'à un nœud composé de plusieurs sommets.

Références:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

C.5.10 Analyse du flux de données

NOTE Cette technique/mesure est mentionnée dans le Tableau B.8 de la CEI 61508-3.

But: Détecter les structures de programme faibles et potentiellement incorrectes.

Description: L'analyse du flux de données est une technique d'essai statique permettant de combiner les informations obtenues à partir de l'analyse du flux de commande avec celles concernant la lecture ou l'écriture des variables dans différentes parties du code. L'analyse permet de vérifier la présence des éléments suivants:

– variables pouvant être lues avant qu'une valeur ne leur soit affectée - ceci peut être évité en affectant toujours une valeur au moment de la déclaration d'une nouvelle variable;

– variables pouvant être écrites plusieurs fois avant d'être lues - ceci peut indiquer une omission de code;

– variables qui sont écrites mais jamais lues - ceci peut indiquer un code redondant.

Une anomalie de flux de données ne correspondra pas toujours directement à une anomalie de programme, mais le nombre d'anomalies susceptibles d'être contenues dans le code sera d'autant moins important que celles du flux de données seront évitées.

Références:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

C.5.11 Exécution symbolique

NOTE Cette technique/mesure est mentionnée dans le Tableau B.8 de la CEI 61508-3.

But: Montrer la concordance entre le code source et la spécification.

Description: Les variables du programme sont évaluées après avoir remplacé le côté gauche par le côté droit pour toutes les affectations. Les branchements conditionnels et les boucles sont traduits en expressions booléennes. Le résultat final consiste en une expression symbolique pour chaque variable de programme. Cette expression est une formule pour la valeur que le programme calculerait si des données réelles étaient fournies. Cela peut être vérifié par rapport à l'expression prévue.

Une utilisation associée de l'exécution symbolique est une méthode systématique de génération de données d'essai pour les cheminements relevant de la logique du programme. Le dispositif d'exécution symbolique peut être incorporé à une boîte à outils intégrée afin de fournir un dispositif d'essai des éléments logiciels et d'analyse du code.

Références:

Using symbolic execution for verifying safety-critical systems. A. Coen-Porisini, G. Denaro, C. Ghezzi, M. Pezzé. Proceedings of the 8th European software engineering conference, and 9th ACM SIGSOFT international symposium on Foundations of software engineering. ACM, 2001, ISBN:1-58113-390-1

Using symbolic execution to guide test generation. G. Lee, J. Morris, K. Parker, G. Bundell, P. Lam. In Software Testing, Verification and Reliability, vol 15, no 1, 2005. John Wiley & Sons, Ltd

C.5.12 Preuve formelle (vérification)

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.5 et A.9 de la CEI 61508-3.

But: Prouver la conformité d'un programme par rapport à un certain modèle abstrait du programme, à l'aide de modèles et de règles théoriques et mathématiques.

Description: L'essai constitue un moyen courant d'examiner la conformité d'un programme. Toutefois, des essais exhaustifs ne peuvent généralement pas être effectués compte tenu de la complexité des programmes de valeur pratique, et par conséquent seule une fraction du comportement potentiel du programme peut être examinée de cette manière. A l'inverse, une vérification formelle applique les opérations mathématiques à une représentation mathématique d'un programme afin d'établir que le programme se comporte comme défini pour toutes les entrées possibles.

La vérification formelle d'un système nécessite un modèle abstrait du programme et de son comportement requis (c'est-à-dire une spécification) en utilisant un langage ayant une signification mathématique précise. La spécification peut être exhaustive, ou être limitée à des propriétés spécifiques du programme:

– propriétés de conformité fonctionnelle, c'est-à-dire qu'il convient que le programme présente une fonctionnalité particulière;
– propriétés de sécurité (c'est-à-dire qu'un comportement inapproprié ne sera jamais effectif) et d'activité (c'est-à-dire qu'un comportement approprié pourra être finalement effectif);
– propriétés de synchronisation, c'est-à-dire qu'un certain comportement sera effectif à un moment particulier.

Le résultat d'une vérification formelle constitue un argument rigoureux qui stipule que le modèle abstrait du programme est cohérent eu égard à la spécification pour toutes les entrées potentielles, c'est-à-dire que le modèle satisfait aux propriétés spécifiées.

Toutefois, la conformité du modèle ne démontre pas directement la conformité du programme réel, une étape supplémentaire nécessaire consistant à montrer que le modèle est une abstraction précise du programme réel pour les propriétés concernées. Certaines propriétés du programme présentant un intérêt pratique ne peuvent pas faire l'objet d'une formalisation mathématique (par exemple la plupart des propriétés de synchronisation et de programmation, ou des propriétés subjectives telles qu'une interface utilisateur «claire et simple », ou effectivement toute propriété ou tout objectif de conception qu'un langage formel ne peut exprimer de manière immédiate). Par conséquent, une vérification formelle ne se substitue pas entièrement à la simulation et aux essais, mais complète en revanche ces techniques en fournissant des éléments de preuve supplémentaires qui viennent à l'appui du fonctionnement correct du programme pour toutes les entrées. Une vérification formelle peut garantir la conformité du modèle abstrait d'un programme, tandis que les essais garantissent que le programme réel se comporte comme prévu.

L'utilisation de la vérification formelle au cours de la phase de conception peut réduire de manière significative le temps de mise au point en détectant les erreurs et méprises significatives dès le début de la phase de conception, et en réduisant ainsi la durée nécessaire au calcul itératif entre la conception et les essais.

Plusieurs méthodes formelles utilisées dans la pratique sont décrites en C.2.4: par exemple, CCS, CSP, HOL, LOTOS, OBJ, logique temporelle, VDM et Z.

C.5.12.1 Vérification des modèles

La vérification des modèles est une méthode utilisée pour la vérification formelle des systèmes réactifs et concurrents. Compte tenu d'une structure d'état fini qui décrit les comportements du système, une propriété qui se présente sous forme d'une logique temporelle fait l'objet d'une vérification visant à déterminer son caractère approprié par rapport à la structure. Des algorithmes efficaces (par exemple SPIN, SMV et UPPAAL) sont utilisés pour parcourir l'ensemble des états de la structure de manière automatique et exhaustive. Lorsque la propriété n'est pas adaptée, un contre-exemple est généré. Ce dernier montre le non-respect de la propriété par la structure et contient par ailleurs des informations très utiles pour l'analyse du système. La vérification des modèles peut détecter des bogues importants éventuellement non détectés par l'inspection et les essais classiques.

Noter que la vérification des modèles se révèle utile dans l'analyse d'une complexité subtile. Ceci peut être utile dans certaines applications à SIL peu élevé, une attention toute particulière étant toutefois nécessaire si des applications à SIL élevé présentent une complexité subtile.

Références:

Is Proof More Cost-Effective Than Testing?. S. King, R. Chapman, J. Hammond, A. Pryor. IEEE Transactions on Software Engineering, vol. 26, no. 8, August 2000

Model Checking. E. M. Clarke, O. Grumberg, and D. A. Peled. MIT Press, 1999, ISBN 0262032708, 9780262032704

Systems and Software Verification: Model-Checking Techniques and Tools. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. Mckenzie, Springer, 2001, ISBN 3-540-41523-8

Logic in Computer Science: Modelling and Reasoning about Systems. M.Huth and M. Ryan. Cambridge University Press, 2000, ISBN 0521652006, 0521656028

The Spin Model Checker: Primer and Reference Manual. G. J. Holzmann. Addison-Wesley, 2003, ISBN 0321228626, 9780321228628

C.5.12.2 (vacant)

C.5.13 Métriques de complexité

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.9 et C.19 de la CEI 61508-3.

But: Prédire les attributs des programmes à partir des caractéristiques du logiciel proprement dit ou de son développement ou d'après l'historique des essais.

Description: Ces modèles évaluent certaines caractéristiques structurelles du logiciel et les relient à l'attribut souhaité tel que la fiabilité ou la complexité. Des outils logiciels sont nécessaires à l'évaluation de la plupart des mesures. Une partie des métriques applicables est indiquée ci-dessous:

– complexité théorique du graphe - cette mesure peut être appliquée au début du cycle de vie en vue d'évaluer les compromis techniques, et est basée sur la complexité du graphe de commande du programme représenté pas son nombre cyclomatique;

– nombre de manières d'activer un certain module logiciel (accessibilité) - plus un module logiciel est accessible, plus il a de chances de pouvoir être mis au point;

– science des métriques de type Halstead - cette mesure calcule la longueur du programme en comptant le nombre d'opérateurs et d'opérandes. Elle fournit une mesure de complexité et de volume qui sert de base de comparaison lors de l'évaluation des ressources de développement futures;

– nombre d'entrées et de sorties pour chaque module logiciel - la minimisation du nombre de points d'entrée/sortie est un facteur clé dans l'utilisation des techniques de conception et de programmation structurées.

Référence:

Metrics and Models in Software Quality Engineering. S.H. Kan. Addison-Wesley, 2003, ISBN 0201729156, 9780201729153

C.5.14 Inspections formelles

NOTE Cette technique/mesure est mentionnée dans le Tableau B.8 de la CEI 61508-3.

But: Révéler les défauts d'une composante logicielle.

Description: L'inspection formelle est un processus structuré d'examen de l'équipement logiciel exécuté par les homologues de la personne qui produit l'équipement destiné à détecter les défauts et de permettre au producteur d'améliorer l'équipement. Il convient que le producteur ne participe au processus d'inspection que pour informer les inspecteurs au cours de la phase d'adaptation. Les inspections formelles peuvent être effectuées sur des éléments logiciels spécifiques produits au cours de toute phase du cycle de vie de développement du logiciel.

Il convient, préalablement à l'inspection, que les inspecteurs se familiarisent avec les équipements à examiner. Il convient de définir clairement le rôle des inspecteurs dans le processus d'inspection. Il convient d'établir un calendrier d'inspection. Il convient de définir des critères d'entrée et de sortie sur la base des propriétés requises pour la composante logicielle. Les critères d'entrée sont les critères ou les exigences qui doivent être satisfaits préalablement à l'inspection. Les critères de sortie sont les critères ou les exigences qui doivent être satisfaits pour achever un processus spécifique.

Au cours de l'inspection, il convient que le modérateur, dont le rôle est de faciliter l'inspection, consigne officiellement les résultats obtenus. Il convient que tous les inspecteurs parviennent à un consensus portant sur les résultats. Il convient de classer les défauts comme suit: a) défauts requérant une correction préalable à l'acceptation ou b) défauts requérant une correction dans un délai / selon une échéance donné(e). Il convient de signifier au producteur les défauts identifiés en vue d'une correction ultérieure après réalisation de l'inspection. Le modérateur peut, selon le nombre et le domaine d'application des défauts identifiés, déterminer que ce processus est nécessaire pour une inspection ultérieure de l'équipement logiciel.

Références:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

Fagan, M. *Design and Code Inspections to Reduce Errors in Program Development*. IBM Systems Journal 15, 3 (1976): 182-211

C.5.15 Lectures croisées (logiciel)

NOTE Cette technique/mesure est mentionnée dans le Tableau B.8 de la CEI 61508-3.

But: Révéler les écarts entre la spécification et la réalisation.

Description: Les lectures croisées constituent une technique informelle exécutée par le producteur d'une composante logicielle en présence de ses homologues afin de détecter les défauts de la composante logicielle. Ces lectures croisées peuvent être effectuées sur des éléments logiciels spécifiques produits au cours de toute phase du cycle de vie de développement du logiciel.

Les fonctions spécifiées du système relatif à la sécurité sont étudiées et évaluées pour assurer que le système relatif à la sécurité est conforme aux exigences de la spécification. Tout point de doute concernant la réalisation et l'utilisation du produit est documenté afin qu'il puisse être résolu. Contrairement à une inspection formelle, l'auteur est actif au cours de la procédure de lectures croisées.

Références:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

C.5.16 Revue de conception

NOTE Cette technique/mesure est mentionnée dans le Tableau B.8 de la CEI 61508-3.

But. Révéler les défauts de conception du logiciel.

Description: Une revue de conception est un examen formel, documenté, exhaustif et systématique de la conception du logiciel visant à évaluer les exigences de conception et la capacité de la conception proprement dite à satisfaire à ces exigences, à identifier les problèmes et à proposer des solutions.

Les revues de conception permettent d'évaluer l'état de la conception par rapport aux entrées requises, et d'identifier les possibilités d'une amélioration ultérieure. A mesure du déroulement des activités du cycle de vie de développement, et compte tenu de la satisfaction des principales étapes de conception détaillées, il convient que les revues de conception soient réalisées pour examiner tous les aspects de l'interface et assurer que la conception peut être vérifiée afin d'établir avec certitude que la conception satisfait à ses exigences et que la conception la plus appropriée est conforme aux exigences de sécurité. Une revue de ce type est destinée principalement à vérifier le travail des concepteurs. Il convient par ailleurs de la considérer comme une activité de confirmation et d'affinement.

Une technique d'inspection rigoureuse telle qu'une analyse de circuit parasite peut être utilisée pour détecter tout comportement incorrect du logiciel tel qu'un cheminement ou un ordinogramme logique imprévu, des sorties inattendues, une synchronisation inappropriée ou des actions non sollicitées.

Références:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

The Art of Software Testing, second edition. G. J. Myers, T. Badgett, T. M. Codd, C. Sandler, John Wiley and Sons, 2004, ISBN 0471469122, 9780471469124

CEI 61160:2005, *Revue de conception*

Space Product Assurance, Sneak analysis - Part 2: Clue list. ECSS-Q-40-04A Part 2. ESA Publications Division, Noordwijk, 1997, ISSN 1028-396X, http://www.everyspec.com/ESA/ECSS-Q-40-04A_Part-2_14981/

C.5.17 Prototypage/animation

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.3 et B.5 de la CEI 61508-3.

But: Vérifier la faisabilité de la mise en œuvre du système par rapport aux contraintes imposées. Communiquer au client l'interprétation du spécificateur concernant le système afin de mettre en évidence les erreurs de compréhension.

Description: Un sous-ensemble de fonctions du système, des contraintes et des exigences de caractéristiques de fonctionnement sont sélectionnés. Un prototype est construit à l'aide d'outils de haut niveau. A cette étape, il n'est pas nécessaire de prendre en compte les contraintes telles que l'ordinateur cible, le langage d'implémentation, la taille du programme, la maintenabilité, la fiabilité et la disponibilité. Le prototype est évalué par rapport aux critères du client et les exigences applicables au système peuvent être modifiées au vu de cette évaluation.

Référence:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.18 Simulation de processus

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.7, C.7, B.3 et C.13 de la CEI 61508-3.

But: Soumettre à essai la fonction d'un système logiciel et son interface avec le monde extérieur, sans lui permettre de modifier, d'aucune manière, le monde réel.

Description: Création d'un système permettant uniquement la réalisation d'essais et simulant le comportement de l'équipement commandé (EUC).

La simulation ne peut être que de type logiciel ou une combinaison d'éléments matériels et logiciels. Elle doit

– fournir des entrées équivalentes à celles utilisées après l'installation effective de l'équipement commandé;
– répondre aux sorties en provenance du logiciel soumis à essai d'une manière représentant fidèlement l'unité commandée;
– prévoir des entrées opérateur pour toutes les perturbations auxquelles le système soumis à essai doit être confronté.

Lors de l'essai du logiciel, la simulation peut être celle du matériel cible avec ses entrées et ses sorties.

Références:

EmStar: An Environment for Developing Wireless Embedded Systems Software. J.Elson et al. http://cens.ucla.edu/TechReports/9_emstar.pdf

A hardware-software co-simulator for embedded system design and debugging. A. Ghosh et. al. In Proceedings of the IFIP International Conference on Computer Hardware Description Languages and Their Applications, IFIP International Conference on Very Large Scale Integration, 1995. IEEE, 1995, ISBN 4930813670, 9784930813671

C.5.19 Exigences relatives au fonctionnement

NOTE Cette technique/mesure est mentionnée dans le Tableau B.6 de la CEI 61508-3.

But: Etablir les exigences relatives au fonctionnement d'un système logiciel qui soient démontrables.

Description: Une analyse des spécifications concernant à la fois le système et les exigences applicables au logiciel est effectuée afin de spécifier toutes les exigences relatives au fonctionnement générales, spécifiques, explicites et implicites.

Chaque exigence relative au fonctionnement est examinée à tour de rôle afin de déterminer

– les critères de réussite à obtenir;
– si une mesure par rapport aux critères de réussite peut être obtenue;
– la précision potentielle de ces mesures;
– les étapes du projet auxquelles les mesures peuvent être estimées; et
– les étapes du projet auxquelles les mesures peuvent être effectuées.

Le caractère pratique de chaque exigence relative au fonctionnement est alors analysé en vue d'obtenir une liste des exigences relatives au fonctionnement, des critères de réussite et des mesures potentielles. Les objectifs principaux sont les suivants:

– associer chaque exigence relative au fonctionnement avec au moins une mesure;
– dans toute la mesure du possible, sélectionner des mesures précises et efficaces susceptibles d'être utilisées le plus tôt possible au cours du développement;
– spécifier les exigences relatives au fonctionnement et les critères de réussite essentiels et facultatifs; et
– dans toute la mesure du possible, il convient de profiter de la possibilité d'utiliser une seule mesure pour plusieurs exigences relatives au fonctionnement.

Référence:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.20 Modélisation du fonctionnement

NOTE Cette technique/mesure est mentionnée dans les Tableaux B.2 et B.5 de la CEI 61508-3.

But: S'assurer que la capacité de fonctionnement du système est suffisante pour satisfaire aux exigences spécifiées.

Description: La spécification des exigences comprend les exigences de capacité de traitement et de réponse applicables aux fonctions spécifiques, qui peuvent être combinées aux contraintes d'utilisation des ressources totales du système. La conception de système proposée est comparée aux exigences énoncées en

– produisant un modèle des processus du système et de leurs interactions;
– déterminant l'utilisation des ressources par chaque processus, par exemple, le temps d'exécution, la bande passante des communications, les mémoires, etc.;
– déterminant la distribution des sollicitations envoyées au système dans des conditions moyennes et dans les cas les plus défavorables;
– calculant la capacité de traitement et les temps de réponse dans des conditions moyennes et dans les cas les plus défavorables pour chaque fonction individuelle du système.

Pour les systèmes simples, une solution analytique peut être suffisante, alors que pour les systèmes plus complexes, une certaine forme de simulation peut être plus appropriée en vue d'obtenir des résultats précis.

Avant une modélisation détaillée, un contrôle plus simple portant sur le «budget des ressources» peut être effectué en additionnant les ressources requises pour tous les

processus. Si les exigences dépassent la capacité du système prévue, la conception est impossible. Même si la conception satisfait à ce contrôle, la modélisation du fonctionnement peut faire apparaître que des délais et des temps de réponse trop longs sont obtenus à cause de l'insuffisance des ressources. Pour éviter cette situation, les ingénieurs conçoivent souvent des systèmes n'utilisant qu'une partie des ressources totales (par exemple 50 %) afin de réduire la probabilité d'une insuffisance des ressources.

Référence:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.21 Essai d'avalanche/ de contraintes

NOTE Cette technique/mesure est mentionnée dans le Tableau B.6 de la CEI 61508-3.

But: Appliquer à l'objet soumis à essai une charge de travail exceptionnellement élevée en vue de démontrer qu'il supportera facilement les charges de travail normales.

Description: Un grand nombre de conditions d'essai peuvent être appliquées pour les essais d'avalanche/de contraintes. Certaines de ces conditions d'essai sont les suivantes:

– en cas de fonctionnement en mode scrutation, un nombre beaucoup plus important de changements par unité de temps se produisent alors à l'entrée de l'objet soumis à l'essai que dans des conditions normales;
– en cas de fonctionnement sur sollicitations, le nombre de sollicitations envoyées à l'objet soumis à essai par unité de temps dépasse alors celui prévu pour des conditions normales;
– si la taille d'une base de données joue un rôle important, cette taille est alors supérieure à celle prévue pour des conditions normales;
– les dispositifs influents sont respectivement réglés de manière à fonctionner à leur vitesse maximale ou minimale;
– pour les cas extrêmes, tous les facteurs influents sont, dans toute la mesure du possible, simultanément placés dans des conditions limites.

Dans ces conditions d'essai, le comportement dans le temps de l'objet soumis à essai peut être évalué. L'effet des changements de charge peut être observé. La dimension correcte des buffers internes, variables dynamiques, piles, etc. peut être vérifiée.

Référence:

Software Engineering for Real-time Systems. J. E. Cooling, Pearson Education, 2003, ISBN 0201596202, 9780201596205

C.5.22 Temps de réponse et contraintes mémoire

NOTE Cette technique/mesure est mentionnée dans le Tableau B.6 de la CEI 61508-3.

But: S'assurer que le système satisfera aux exigences temporelles et à celles concernant la mémoire.

Description: La spécification des exigences applicables au système et au logiciel comprend les exigences de mémoire et de réponse applicables aux fonctions spécifiques qui peuvent être combinées aux contraintes d'utilisation des ressources totales du système.

Une analyse est effectuée pour déterminer les demandes de répartition dans des conditions moyennes et dans les cas les plus défavorables. Cette analyse nécessite une estimation de l'utilisation des ressources et du temps écoulé pour chaque fonction du système. Ces

estimations peuvent être obtenues de plusieurs manières, par exemple, par comparaison avec un système existant ou par prototypage et évaluation des performances des systèmes à durée critique.

C.5.23 Analyse d'impact

NOTE Cette technique/mesure est mentionnée dans le Tableau A.8 de la CEI 61508-3.

But: Déterminer l'effet d'une modification ou d'une amélioration apportée à un système logiciel sur les autres modules logiciels de ce système ainsi que sur les autres systèmes.

Description: Avant d'apporter une modification ou une amélioration au logiciel, il convient d'effectuer une analyse pour déterminer l'impact de la modification ou de l'amélioration sur le logiciel, ainsi que pour déterminer quels systèmes et modules logiciels sont affectés.

Après l'analyse, une décision doit être prise en ce qui concerne la revérification du système logiciel. Cette décision dépend du nombre et de la criticité des modules logiciels affectés, ainsi que de la nature de la modification. Des décisions possibles sont les suivantes:

– revérification du seul module logiciel modifié;

– revérification de tous les modules logiciels affectés; ou

– revérification du système complet.

Références:

Requirements Engineering. E. Hull, K. Jackson, J. Dick. Springer, 2005, ISBN 1852338792, 9781852338794

C.5.24 Gestion de configuration logicielle

NOTE Cette technique/mesure est mentionnée dans le Tableau A.8 de la CEI 61508-3.

But: Le but de la gestion de configuration logicielle est d'assurer la cohérence des groupes de fournitures de développement au fur et à mesure de la modification de ces fournitures. La gestion de configuration s'applique en général tant au développement du matériel que du logiciel.

Description: La gestion de configuration logicielle est une technique qui est utilisée tout au long du développement (voir CEI 61508-3, 6.2.3). Elle consiste essentiellement à documenter la production de chaque version de chaque fourniture «significative» et de chaque relation entre les différentes versions des différentes fournitures. La documentation résultante permet au développeur de déterminer les effets de la modification d'une fourniture (spécialement d'un de ses composants) sur les autres fournitures. En particulier, les systèmes ou sous-systèmes peuvent être reconstruits de manière sûre à partir d'ensembles cohérents des versions de composants.

Références:

Software engineering: Update. Ian Sommerville, Addison-Wesley Longman, Amsterdam; 8th ed., 2006, ISBN 0321313798, 9780321313799

Software Engineering. Ian Sommerville, Pearson Studium, 8. Auflage, 2007, ISBN 3827372577, 9783827372574

Software Configuration Management: Coordination for Team Productivity. W.A. Babich. Addison-Wesley, 1986, ISBN 0201101610, 9780201101614

CMMI: guidelines for process integration and product improvement, Mary Beth Chrissis, Mike Konrad, Sandy Shrum, Addison-Wesley, 2003, ISBN 0321154967, 9780321154965

C.5.25 Validation de régression

NOTE Cette technique/mesure est mentionnée dans le Tableau A.8 de la CEI 61508-3.

But: S'assurer que l'essai de régression permet de tirer des conclusions valides.

Description: Un essai de régression exhaustif d'un système important ou complexe requiert généralement un effort et des ressources considérables. Il est souhaitable, dans toute la mesure du possible, de limiter l'essai de régression afin de couvrir les seuls aspects du système ayant un intérêt direct à cette étape du développement du système. Il est essentiel que cet essai de régression partiel ait une bonne compréhension de sa portée et permette de tirer uniquement des conclusions valides concernant l'état du système soumis à essai.

Référence:

Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing. R.Black, John Wiley and Sons, 2002, ISBN 0471223980, 9780471223986

C.5.26 Animation de la spécification et de la conception

NOTE Cette technique/mesure est mentionnée dans le Tableau A.9 de la CEI 61508-3.

But: Guider la vérification du logiciel au moyen d'un examen systématique de la spécification.

Description: Une représentation du logiciel plus abstraite que le code exécutable (c'est-à-dire une spécification ou une conception évoluée) est examinée pour déterminer le comportement du logiciel exécutable potentiel. L'examen est automatisé d'une certaine manière (selon les possibilités admises par la nature et le niveau d'abstraction de la représentation évoluée) afin de simuler le comportement et les sorties du logiciel exécutable. Une application de cette approche consiste à générer des essais (ou « oracles ») susceptibles d'être appliqués ultérieurement au logiciel exécutable, consistant ainsi à automatiser d'une certaine manière le processus d'essai. Une autre application consiste à animer une interface utilisateur de sorte que les utilisateurs finals non techniques puissent apprécier d'une certaine manière la signification détaillée de la spécification sur la base de laquelle travailleront les développeurs de logiciels. Ceci fournit une méthode de communication valable entre les deux groupes.

Références:

Supporting the Software Testing Process through Specification Animation. T.Miller, P.Strooper. In Proceedings of the First International Conference on Software Engineering and Formal Methods (SEFM'03), ed. P.Lindsay. IEEE Computer Society, IEEE Computer Society, 2003, ISBN 0769519490, 9780769519494

B model animation for external verification. H.Waeselynck, S.Behnia, In Proceedings. of the Second International Conference on Formal Engineering Methods, 1998. IEEE Computer Society, 1998, ISBN 0-8186-9198-0

C.5.27 Essais basés sur le modèle (génération de cas d'essai)

NOTE Cette technique/mesure est mentionnée dans le Tableau A.5 de la CEI 61508-3.

But: Renforcer l'efficacité de la génération automatique de cas d'essai à partir de modèles de système et générer des séquences d'essais hautement répétables.

Description: L'essai basé sur le modèle (MBT) est une approche « boîte noire » pour laquelle les tâches d'essai communes telles que la génération de cas d'essai (TCG) et l'évaluation

des résultats d'essai, sont basées sur un modèle du système (application) en essai (SUT). En règle générale, sans toutefois s'y limiter, les données du système et le comportement de l'utilisateur sont modélisés au moyen d'automates d'états finis, de processus de Markov, de tables de décision ou analogues (El-Far, 2001, généralisé). Par ailleurs, l'essai basé sur le modèle peut être associé à la mesure de couverture d'essai de niveau de code source, et les modèles fonctionnels peuvent être basés sur un code source existant.

L'essai basé sur le modèle assure la génération automatique de cas d'essai/procédures efficaces au moyen des modèles des exigences du système et des fonctionnalités spécifiées (SoftwareTech, 2009).

Dans la mesure où l'essai est très onéreux, il réclame d'utiliser un nombre très important d'outils de génération de cas d'essai. Par conséquent, l'essai basé sur le modèle représente actuellement un domaine très actif de recherche mettant à disposition un très grand nombre d'outils de TCG (génération de cas d'essai). En règle générale, ces outils extraient une séquence d'essais de la partie comportementale du modèle, garantissant ainsi la conformité à certaines exigences de couverture.

Le modèle est une représentation abstraite et partielle du comportement souhaité du système en essai (SUT). Des modèles d'essai sont déduits de ce modèle et permettent de construire une séquence d'essais abstraite. Les cas d'essai sont déduits de cette séquence d'essais abstraite et exécutés en fonction du système. Les essais peuvent être réalisés également en fonction du modèle du système. Le MBT avec TCG est fondé sur cette méthode et fortement lié à l'utilisation de méthodes formelles, les recommandations correspondantes sont de ce fait similaires pour ce qui concerne les niveaux d'intégrité de sécurité (SIL): HR (fortement recommandé) pour des SIL supérieurs et non requis pour des SIL inférieurs.

En général, les activités spécifiques sont les suivantes:

– construction du modèle (à partir des exigences du système)
– génération des entrées prévues
– génération des sorties prévues
– essais d'exécution
– comparaison des sorties réelles avec les sorties prévues
– décision relative à l'action ultérieure (modifier le modèle, générer plus d'essais, estimer la fiabilité/qualité du logiciel)

Les essais peuvent être déduits selon différentes méthodes et techniques d'expression des modèles de comportement utilisateur/système, en utilisant par exemple

– des tables de décision
– des automates d'états finis
– des grammaires
– des modèles de chaîne de Markov
– des diagrammes d'état
– des démonstrations de théorème
– une programmation logique de contraintes
– la vérification du modèle
– une exécution symbolique
– un modèle de flux d'événements
– des essais de système réactifs: automate fini hiérarchique parallèle
– etc.

L'essai basé sur le modèle a plus récemment pour objectif spécifique le domaine critique de la sécurité. Il permet de révéler très tôt les ambiguïtés dans la spécification et la conception, il fournit la capacité de génération automatique de nombreux essais non répétitifs efficaces permettant d'évaluer les séquences d'essais de régression ainsi que la fiabilité et la qualité du logiciel, il facilite également l'actualisation des séquences d'essais.

ElFar (2001) et SoftwareTech 2009 (voir références) fournissent une présentation détaillée, d'autres détails et problèmes spécifiques au domaine sont traités dans les autres références.

Références:

T. Bauer, F. Böhr, D. Landmann, T. Beletski, R. Eschbach, Robert and J.H. Poore, *From Requirements to Statistical Testing of Embedded Systems* Software Engineering for Automotive Systems - SEAS 2007, ICSE Workshops, Minneapolis, USA

Eckard Bringmann, Andreas Krämer; *Model-based Testing of Automotive Systems* In: ICST, pp.485-493, 2008 International Conference on Software Testing, Verification, and Validation, 2008

Broy M., *Challenges in automotive software engineering*, International conference on Software engineering (ICSE '06),Shanghai, China, 2006

I. K. El-Far and J. A. Whittaker, *Model-Based Software Testing*. Encyclopedia of Software Engineering (edited by J. J. Marciniak). Wiley, 2001

Heimdahl, M.P.E.: *Model-based testing: challenges ahead, Computer Software and Applications Conference* (COMPSAC 2005), 25-28 July 2005, Edinburgh, Scotland, UK, 2005

Jonathan Jacky, Margus Veanes, Colin Campbell, and Wolfram Schulte, *Model-Based Software Testing and Analysis with C#*, ISBN 978-0-521-68761-4, Cambridge University Press 2008.

A. Paradkar, *Case Studies on Fault Detection Effectiveness of Model-based Test Generation Techniques*, in ACM SIGSOFT SW Engineering Notes, Proc. of the first int. workshop on Advances in model-based testing A-MOST '05, Vol. 30 Issue 4. ACM Press 2005

S. J. Prowell, *Using Markov Chain Usage Models to Test Complex Systems*, HICSS '05: 38th Annual Hawaii, International Conference on System Sciences, 2005

Mark Utting and Bruno Legeard, *Practical Model-Based Testing: A Tools Approach*, ISBN 978-0-12-372501-1, Morgan-Kaufmann 2007

Hong Zhu et al. (2008). AST '08: *Proceedings of the 3rd International Workshop on Automation of Software Test*. ACM Press. ISBN 978-1-60558-030-2

Model-Based Testing of Reactive Systems Advanced Lecture Series, LNCS 3472, Springer-Verlag, 2005, ISBN 978-3-540-26278-7

Model-based Testing, SoftwareTech July 2009, Vol. 12, No. 2, Software Testing: A Life Cycle Perspective, http://www.goldpractices.com/practices/mbt/

C.6 Evaluation de la sécurité fonctionnelle

NOTE Les techniques et mesures concernées sont également décrites en B.6.

C.6.1 Tables de décision (tables de vérité)

NOTE Cette technique/mesure est mentionnée dans les Tableaux A.10 et B.7 de la CEI 61508-3.

But: Fournir une spécification et une analyse claires et cohérentes des combinaisons logiques complexes et de leurs relations.

Description: Cette méthode utilise des tables à deux dimensions pour décrire de manière concise les relations logiques entre les variables de programme booléennes.

La concision et la nature tabulaire de la méthode rendent celle-ci appropriée en tant que moyen d'analyse des combinaisons logiques complexes exprimées sous forme de code.

La méthode est potentiellement exécutable lorsqu'elle est utilisée en tant que spécification.

C.6.2 Etude de danger et d'opérabilité des logiciels (CHAZOP, AMDE)

But: Déterminer les dangers mettant en cause la sécurité d'un système proposé ou existant, leurs causes et conséquences possibles, ainsi que les actions recommandées pour minimiser leur chance d'apparition.

Description: Une équipe d'ingénieurs, dont l'expérience couvre l'ensemble du système concerné, effectue un examen structuré d'une conception, au cours d'une série de réunions programmées. Ces ingénieurs examinent les aspects fonctionnels de la conception ainsi que la manière dont le système sera exploité en pratique (y compris les interventions humaines et la maintenance). Le chef d'équipe encourage la créativité des membres de l'équipe en ce qui concerne la dénonciation des dangers potentiels et conduit la procédure en présentant chaque partie du système en fonction de plusieurs mots clés: «aucun», «plus de», «moins de», «partie de», «plus que» (ou «aussi bien que») et «autre que». Chaque type de condition ou de mode de défaillance appliqué est considéré en fonction de sa faisabilité, de la manière dont il peut se produire, des conséquences possibles (y-a-t-il un danger ?), de la manière dont il peut être évité et de la dépense à engager ou non pour l'éviter.

Les études de dangers peuvent être réalisées à plusieurs étapes du développement d'un projet mais sont plus efficaces lorsqu'elles sont réalisées suffisamment tôt afin de pouvoir être prises en compte pour les décisions majeures concernant la conception et l'opérabilité.

La technique HAZOP, qui s'est développée dans l'industrie manufacturière, nécessite d'être modifiée pour une application logicielle. Différentes méthodes dérivées (ou «CHAZOP»: techniques HAZOP pour systèmes informatiques) ont été proposées. Ces méthodes utilisent généralement de nouveaux mots clés et/ou suggèrent des solutions pour la couverture systématique de l'architecture du système et du logiciel.

Références:

OF-FMEA: an approach to safety analysis of object-oriented software intensive systems, T. Cichocki, J. Gorski. In Artificial Intelligence and Security in Computing Systems: 9th *International Conference*, ACS '2002. Ed. J. Soldek. Springer, 2003, ISBN 1402073968, 9781402073960

Software FMEA techniques. P.L.Goddard. In Proc Annual 2000 Reliability and Maintainability Symposium, IEEE, 2000, ISBN: 0-7803-5848-1

Software criticality analysis of COTS/SOUP. P.Bishop, T.Clement, S.Guerra. In Reliability Engineering & System Safety, Volume 81, Issue 3, September 2003, Elsevier Ltd., 2003

C.6.3 Analyse des défaillances de cause commune

NOTE 1 Cette technique/mesure est mentionnée dans le Tableau A.10 de la CEI 61508-3.

NOTE 2 Voir aussi l'Annexe D de la CEI 61508-6.

But: Déterminer les défaillances potentielles des systèmes ou sous-systèmes multiples susceptibles de réduire les avantages de la redondance par l'apparition simultanée des mêmes défaillances dans les multiples parties.

Description: Les systèmes chargés de la sécurité d'une installation utilisent souvent la redondance en ce qui concerne les éléments matériels ainsi que le vote majoritaire. Cette technique est utilisée afin d'éviter les défaillances aléatoires du matériel dans les composants ou les sous-systèmes qui tendraient à empêcher le traitement correct des données.

Toutefois, certaines défaillances peuvent être communes à plusieurs composants ou sous-systèmes. Par exemple, si un système est installé dans une seule pièce, les défauts du système de conditionnement d'air peuvent affecter les avantages de la redondance. La même chose se produit en ce qui concerne les autres effets extérieurs sur le système tels que le feu, l'inondation, les perturbations électromagnétiques, les accidents d'avions et les tremblements de terre. Le système peut être également affecté par des incidents liés à l'exploitation et à la maintenance. Il est par conséquent essentiel que des procédures adéquates et bien documentées soient fournies pour l'exploitation et la maintenance, et que le personnel d'exploitation et de maintenance soit parfaitement entraîné.

Les effets internes contribuent également de façon majeure à l'apparition des défaillances de cause commune. Ils peuvent être dus à des anomalies de conception des composants communs ou identiques et de leurs interfaces, ainsi qu'au vieillissement des composants. L'analyse des défaillances de cause commune a pour but la recherche des défaillances communes potentielles à l'intérieur du système. Les méthodes suivantes sont utilisées pour l'analyse des défaillances de cause commune: contrôle général de la qualité, revues de conception, vérification et essai par une équipe indépendante et analyse des incidents réels avec retour d'expérience acquise sur systèmes similaires. Toutefois, l'analyse porte au-delà des éléments matériels. Même si une certaine diversité logicielle est utilisée pour les différentes voies d'un système redondant, il peut y avoir une certaine similitude dans les approches logicielles susceptibles d'occasionner une défaillance de cause commune comme, par exemple, les anomalies de la spécification commune.

Lorsque des défaillances de cause commune ne se produisent pas exactement en même temps, des mesures de prévention peuvent être prises en utilisant des méthodes de comparaison des différentes voies susceptibles de permettre la détection d'une défaillance avant que cette défaillance soit commune à toutes les voies. Il convient de prendre cette technique en compte dans l'analyse des défaillances de cause commune.

Référence:

Reliability analysis of hierarchical computer-based systems subject to common-cause failures. L.Xing, L.Meshkat, S.Donohue. Reliability Engineering & System Safety Volume 92, Issue 3, March 2007

C.6.4 Diagrammes de blocs de fiabilité

NOTE 1 Cette technique/mesure est mentionnée dans le Tableau A.10 de la CEI 61508-3 et est utilisée en Annexe B de la CEI 61508-6.

NOTE 2 Voir également B.6.6.7 "Diagrammes de blocs de fiabilité".

But: Modéliser, sous forme de diagramme, l'ensemble des événements qui doivent se produire et les conditions qui doivent être satisfaites pour obtenir le fonctionnement correct d'un système ou d'une tâche.

Description: Le but de l'analyse est représenté par un cheminement réussi constitué de cases, de lignes et de jonctions logiques. Un cheminement réussi débute d'un côté du diagramme et continue à travers les cases et les jonctions vers l'autre côté du diagramme. Une case représente une condition ou un événement et le cheminement peut traverser la

case si la condition est vraie ou si l'événement s'est produit. Si le cheminement arrive à une jonction, il continue si la logique de la jonction est satisfaite. S'il atteint un sommet, le cheminement peut continuer le long de toutes les lignes sortantes. Si au moins un cheminement réussi traverse le diagramme, l'analyse est satisfaisante.

Références:

CEI 61025:2006, *Analyse par arbre de panne (AAP)*

From safety analysis to software requirements. K.M. Hansen, A.P. Ravn, A.P, V Stavridou. IEEE Trans Software Engineering, Volume 24, Issue 7, Jul 1998

CEI 61078:2006, *Techniques d'analyse pour la sûreté de fonctionnement – Bloc-diagramme de fiabilité et méthodes booléennes*

Annexe D
(informative)

Une approche probabiliste pour déterminer l'intégrité de sécurité logicielle pour un logiciel prédéveloppé

D.1 Généralités

La présente annexe fournit les lignes directrices initiales concernant l'utilisation d'une approche probabiliste en vue de déterminer l'intégrité de sécurité logicielle d'un logiciel prédéveloppé à partir de l'expérience en exploitation. Cette approche est considérée comme particulièrement adaptée en tant que partie de la qualification des systèmes d'exploitation, des éléments de bibliothèque, des compilateurs et autres logiciels système. Cette annexe fournit une indication sur ce qui est possible, mais il convient que les techniques ne soient utilisées que par les personnes compétentes en analyse statistique.

NOTE La présente annexe utilise le terme niveau de confiance, qui est décrit dans l'IEEE 352. Un terme équivalent, niveau de signification, est utilisé dans la CEI 61164.

Les techniques pourraient également être utilisées pour démontrer l'amélioration du niveau d'intégrité de sécurité logicielle dans le temps. Par exemple, il peut être montré qu'un logiciel réalisé selon les exigences de la CEI 61508-3 de manière à atteindre le niveau d'intégrité de sécurité 1 (SIL 1) peut, après une certaine période d'utilisation satisfaisante dans un grand nombre d'applications, atteindre le niveau SIL 2.

Le Tableau D.1 ci-dessous montre le nombre de sollicitations sans défaillance traitées ou le nombre d'heures d'exploitation nécessaire pour obtenir la qualification correspondant à un certain niveau d'intégrité de sécurité. Ce tableau est un résumé des résultats donnés en D.2.1 et D.2.3.

L'expérience en exploitation peut être traitée de façon mathématique comme présenté en D.2 ci-dessous pour compléter ou remplacer l'essai statistique, et l'expérience en exploitation venant de plusieurs sites peut être combinée (c'est-à-dire par addition du nombre de sollicitations traitées ou d'heures d'exploitation), mais uniquement si

– la version logicielle à utiliser dans le système E/E/PE relatif à la sécurité est identique à la version pour laquelle l'expérience en exploitation est revendiquée;
– le profil opérationnel de l'espace d'entrée est similaire;
– l'existence un système efficace pour rapporter et documenter les défaillances; et
– les conditions préalables significatives (voir D.2. ci-dessous) sont remplies.

Tableau D.1 – Historique nécessaire pour assurer des niveaux d'intégrité de sécurité

SIL	Mode de fonctionnement à faible sollicitation	Nombre de sollicitations traitées		mode de fonction-nement à sollicitation élevée ou continu	Nombre total d'heures de fonctionnement	
	(Probabilité de défaillance concernant la réalisation de la fonction prévue en cas de sollicitation)	1-α = 0,99	1-α = 0,95	(Probabilité de défaillance dangereuse par heure)	1-α = 0,99	1-α = 0,95
4	$\geq 10^{-5}$ à $< 10^{-4}$	$4{,}6 \times 10^5$	3×10^5	$\geq 10^{-9}$ à $< 10^{-8}$	$4{,}6 \times 10^9$	3×10^9
3	$\geq 10^{-4}$ à $< 10^{-3}$	$4{,}6 \times 10^4$	3×10^4	$\geq 10^{-8}$ à $< 10^{-7}$	$4{,}6 \times 10^8$	3×10^8
2	$\geq 10^{-3}$ à $< 10^{-2}$	$4{,}6 \times 10^3$	3×10^3	$\geq 10^{-7}$ à $< 10^{-6}$	$4{,}6 \times 10^7$	3×10^7
1	$\geq 10^{-2}$ à $< 10^{-1}$	$4{,}6 \times 10^2$	3×10^2	$\geq 10^{-6}$ à $< 10^{-5}$	$4{,}6 \times 10^6$	3×10^6
NOTE 1 1-α représente le niveau de confiance.						
NOTE 2 Voir D.2.1 et D.2.3 pour les conditions préalables et les détails concernant l'obtention de ce tableau.						

D.2 Formules d'essais statistiques et exemples d'utilisation

D.2.1 Essai statistique simple en mode de fonctionnement à faible sollicitation

D.2.1.1 Conditions préalables

a) La distribution des données d'essai est égale à la distribution des sollicitations pendant l'exploitation en ligne.

b) Les passages d'essai sont statistiquement indépendants les uns des autres, par rapport à la cause de la défaillance.

c) Un mécanisme adéquat existe pour détecter toute défaillance qui peut se produire.

d) Nombre de cas d'essai $n > 100$.

e) Aucune défaillance ne se produit pendant les n cas d'essai.

D.2.1.2 Résultats

La probabilité de défaillance p (par sollicitation), au niveau de confiance 1-α, est donnée par

$$p \leq 1 - \sqrt[n]{\alpha} \qquad \text{ou} \qquad n \geq -\frac{\ln \alpha}{p}$$

D.2.1.3 Exemple

Tableau D.2 – Probabilités de défaillance en mode de fonctionnement à faible sollicitation

1-α	P
0,95	3/n
0,99	4,6/n

Pour une probabilité de non fonctionnement en cas de sollicitation de niveau SIL 3 avec un niveau de confiance de 95 %, l'application de la formule donne 30 000 cas d'essai en tenant compte des conditions préalables. Le Tableau D.1 résume les résultats pour chaque niveau d'intégrité de sécurité.

D.2.2 Essai d'un espace (domaine) d'entrée pour un mode de fonctionnement à faible sollicitation

D.2.2.1 Conditions préalables

La seule condition préalable est que les données d'essai doivent être sélectionnées pour donner une distribution aléatoire uniforme sur tout l'espace (domaine) d'entrée.

D.2.2.2 Résultats

L'objectif est de trouver le nombre d'essais, n, qui sont nécessaires, en se fondant sur le seuil de précision, δ, des entrées pour la fonction à faible sollicitation (telle qu'un arrêt d'urgence de sécurité) soumise à essai.

Tableau D.3 – Distances moyennes de deux points d'essai

Dimension du domaine	Distance moyenne de deux points d'essai en direction d'un axe arbitraire
1	$\delta = 1/n$
2	$\delta = \sqrt[2]{1/n}$
3	$\delta = \sqrt[3]{1/n}$
k	$\delta = \sqrt[k]{1/n}$
NOTE k peut être tout entier positif. Les valeurs 1,2 et 3 ne sont que des exemples.	

D.2.2.3 Exemple

Considérons un arrêt d'urgence de sécurité qui est dépendant de deux variables uniquement, A et B. S'il a été vérifié que les seuils qui divisent la paire d'entrée des variables A et B sont traités correctement jusqu'à une précision de 1 % de la plage de mesure de A ou de B, le nombre de cas d'essai uniformément distribués requis dans l'espace de A et de B est

$$n = 1/\delta^2 = 10^4$$

D.2.3 Essai statistique simple en mode de fonctionnement à sollicitation élevée ou continu

D.2.3.1 Conditions préalables

a) La distribution des données d'essai est égale à la distribution pendant l'exploitation en ligne.

b) La réduction relative de la probabilité d'absence de défaillance est proportionnelle à la longueur de l'intervalle de temps considéré ou est constante à défaut.

c) Un mécanisme adéquat existe pour détecter toute défaillance qui peut se produire.

d) L'essai dure pendant un temps t.

e) Aucune défaillance ne se produit pendant le temps t.

D.2.3.2 Résultats

La relation entre la probabilité de défaillance λ, le niveau de confiance 1-α et le temps d'essai t est

$$\lambda = -\frac{\ln \alpha}{t}$$

La probabilité de défaillance est indirectement proportionnelle au temps moyen entre défaillances (MTBF):

$$\lambda = \frac{1}{\text{MTBF}}$$

NOTE La présente norme ne fait pas de distinction entre la probabilité de défaillances par heure, et le taux de défaillance en 1 h. Strictement, la probabilité de défaillance, *F*, est liée au taux de défaillance, *f*, par l'équation $F = 1\text{-}e^{-ft}$, mais le domaine d'application de la présente norme concerne les taux de défaillances de moins de 10^{-5} et pour ces valeurs $F \approx ft$.

D.2.3.3 Exemple

Tableau D.4 – Probabilités de défaillance en mode de fonctionnement à sollicitation élevée ou continu

1-α	**λ**
0,95	3/*t*
0,99	4,6/*t*

Pour vérifier que le temps moyen entre défaillances est au moins égal à 10^8 h avec un niveau de confiance de 95 %, un temps d'essai de 3×10^8 h est requis et les conditions préalables doivent être remplies. Le Tableau D.1 résume le nombre d'essais requis pour chaque niveau d'intégrité de sécurité.

D.2.4 Essai complet

Le programme est considéré comme une urne contenant un nombre connu *N* de boules. Chaque boule représente une propriété du programme concernée. Les boules sont tirées au hasard et replacées dans l'urne après inspection. L'essai complet est effectué lorsque toutes les boules sont tirées.

D.2.4.1 Conditions préalables

a) La distribution des données d'essai est telle que chacune des *N* propriétés du programme est soumise à essai avec une probabilité égale.

b) Les passages d'essai sont indépendants les uns des autres.

c) Chaque défaillance qui se produit est détectée.

d) Le nombre de cas d'essai $n >> N$.

e) Aucune défaillance ne se produit pendant les *n* cas d'essai.

f) Chaque passage d'essai concerne une propriété du programme (une propriété du programme est ce qui peut être soumis à essai au cours d'un passage).

D.2.4.2 Résultats

La probabilité *p* de soumettre à essai toutes les propriétés du programme est donnée par

$$p = \sum_{j=0}^{N-1} (-1)^j \binom{N}{j} \left(\frac{N-j}{N}\right)^n \text{ ou } p = 1 + \sum_{j=1}^{N} (-1)^j \, C_{j,N} \left(\frac{N-j}{N}\right)^n$$

où

$$C_{j,N} = \frac{N(N-1)\ldots(N-j+1)}{j!}$$

En ce qui concerne l'évaluation de cette formule, en général seuls les premiers termes sont importants puisque les cas réalistes sont caractérisés par $n >> N$. Le dernier facteur rend très petits les termes correspondant aux grands j. Cela est également visible dans le Tableau D.5.

D.2.4.3 Exemple

Considérons un programme qui a été utilisé dans plusieurs installations depuis plusieurs années. Au total, au moins $7{,}5 \times 10^6$ passages ont été exécutés. Il est estimé que chaque centième passage (d'essai) répond aux conditions préalables susmentionnées. Alors, $7{,}5 \times 10^4$ passages peuvent être utilisés pour une évaluation statistique. Il est estimé que 4 000 passages d'essai correspondraient à un essai exhaustif. Les estimations sont prudentes. Conformément au Tableau D.5, la probabilité de ne pas avoir soumis à essai un élément est égale à $2{,}87 \times 10^{-5}$.

Pour N = 4 000, les valeurs des premiers termes qui dépendent de n sont les suivantes:

Tableau D.5 – Probabilité d'essai de toutes les propriétés du programme

n	***P***
5×10^4	$1 - 1{,}49 \times 10^{-2} + 1{,}10 \times 10^{-4} - \ldots$
$7{,}5 \times 10^4$	$1 - 2{,}87 \times 10^{-5} + 4 \times 10^{-10} - \ldots$
1×10^5	$1 - 5{,}54 \times 10^{-8} + 1{,}52 \times 10^{-15} - \ldots$
2×10^5	$1 - 7{,}67 \times 10^{-19} + 2{,}9 \times 10^{-37} - \ldots$

Dans la pratique, il convient d'effectuer ces estimations avec la plus grande prudence.

D.3 Références

De plus amples informations sur les techniques ci-dessus peuvent être trouvées dans les documents suivants:

CEI 61164:2004, *Reliability growth – Statistical test and estimation methods* (disponible en anglais seulement)

Verification and Validation of Real-Time Software, Chapter 5. W. J. Quirk (ed.). Springer Verlag, 1985, ISBN 3-540-15102-8

Combining Probabilistic and Deterministic Verification Efforts. W. D. Ehrenberger, SAFECOMP 92, Pergamon Press, ISBN 0-08-041893-7

Ingenieurstatistik. Heinhold/Gaede, Oldenburg, 1972, ISBN 3-486-31743-1

IEEE 352:1987, *IEEE Guide for general principles of reliability analysis of nuclear power generating station safety systems*

Annexe E
(informative)

Présentation de techniques et mesures pour la conception des ASIC

NOTE Cette annexe est une présentation des techniques/mesures référencées dans la CEI 61508-2. Il convient de ne pas considérer cette annexe comme étant complète ou exhaustive.

E.1 Description de conception en langage (V)HDL

But: Description fonctionnelle de niveau élevé en langage de description de circuit, par exemple VHDL ou Verilog.

Description: Description fonctionnelle de niveau d'abstraction élevé en langage de description de circuit, par exemple VHDL ou Verilog. Il convient que le langage de description de circuit appliqué permette une description fonctionnelle et/ou orientée vers l'application et soit soustrait aux détails d'implémentation ultérieure. Il convient que l'implémentation des flux de données, branchements, opérations arithmétiques et/ou logiques s'effectue par affectation et par intervention des opérateurs du langage de description de circuit, sans conversion manuelle en portes logiques de la bibliothèque appliquée.

NOTE A des fins de simplification, l'expression « description fonctionnelle de niveau d'abstraction élevé en langage de description de circuit » est désignée dans le reste du document par le terme (V)HDL.

Référence:

IEEE VHDL, *Verilog + Standard VHDL Design guide*

E.2 Entrée schématique

But: Description fonctionnelle des circuits par l'établissement d'un schéma de circuits en utilisant les portes et/ou les macros de la bibliothèque du fournisseur.

Description: Description de la fonctionnalité du circuit par l'entrée schématique du schéma de circuits. Il convient d'implémenter la fonction à exécuter par instanciation (importation) des éléments de circuits logiques élémentaires tels que AND (ET), OR (OU), NOT (NON), avec des macros constituées de fonctions arithmétiques et logiques complexes, qui sont ensuite interconnectés. Il convient de séparer les circuits complexes compte tenu des points de vue fonctionnels, lesdits circuits pouvant par ailleurs être répartis sur des schémas différents, interconnectés de manière hiérarchique. Il convient de définir les signaux d'interconnexion de manière unique, avec des noms de signaux explicites sur l'ensemble de la hiérarchie. Il convient, dans toute la mesure du possible, de ne pas utiliser de signaux généraux (étiquettes).

E.3 Description structurée

NOTE Voir également C.2.7 "Programmation structurée" et E.12 "Modularisation".

But: Il convient de structurer la description de la fonctionnalité du circuit d'une manière facilement lisible, c'est-à-dire que la fonction du circuit peut être comprise de manière intuitive sur la base d'une description sans efforts de simulation.

Description: Description de la fonctionnalité du circuit par le langage (V)HDL ou par l'entrée schématique. Une structure modulaire facilement identifiable est recommandée. Il convient d'implémenter chaque module de la même manière et de le décrire de sorte qu'il soit facilement lisible avec des sous-fonctions clairement définies. Une distinction nette entre la

fonction implémentée et l'interconnexion est recommandée, c'est-à-dire que le module, qui est implémenté par instanciation des autres sous-modules, contient des interconnexions explicites des modules instanciés. Il convient en revanche que le module ne contienne aucun circuit logique.

E.4 Outils éprouvés (par une utilisation antérieure)

But: Application d'outils éprouvés (par une utilisation antérieure) afin d'éviter toute défaillance systématique par une pratique éprouvée suffisante des outils dans des projets divers.

Description: La plupart des outils utilisés pour la conception des ASIC et des FPGA comprennent des logiciels évolués qui ne peuvent pas être considérés comme fonctionnant sans anomalie eu égard à leur fonctionnalité appropriée. Egalement, des anomalies pourraient très vraisemblablement se produire en raison d'un fonctionnement défectueux. Il convient, par conséquent, d'utiliser de préférence uniquement des outils qualifiés d'éprouvés (par une utilisation antérieure) pour la conception des ASIC et des FPGA. Ceci implique:

– L'application d'outils ayant été utilisés (dans une version logicielle comparable) pendant une longue période ou par un nombre élevé d'utilisateurs avec des projets différents présentant une complexité équivalente.
– Une expérience appropriée de chaque concepteur ASIC/FPGA avec utilisation de l'outil pendant une longue période.
– L'utilisation d'outils d'utilisation courante (nombre adéquat d'utilisateurs) de sorte que les informations concernant les défaillances connues avec solutions de rechange (contrôle de version avec liste des points noirs) soient disponibles. Il convient que ces informations soient intégrées immédiatement au flux de calcul et permettent d'éviter toutes défaillances systématiques.
– Le contrôle de cohérence de la base de données d'outils interne et le contrôle de vraisemblance préviennent l'utilisation de données de sortie défaillantes. Les outils normalisés permettent de contrôler la cohérence de la base de données interne, par exemple la cohérence de la base de données entre l'outil de synthèse et l'outil de placement et de routage, afin d'utiliser des données uniques.

NOTE Le contrôle de cohérence étant un attribut intrinsèque de l'outil utilisé, l'influence du concepteur sur ce dernier est limitée. Par conséquent, si la possibilité d'un contrôle de cohérence manuel est offerte, il convient que le concepteur utilise ce dernier de manière appropriée.

E.5 Simulation (V)HDL

NOTE Voir également E.6 "Essai fonctionnel sur niveau de module".

But: Vérification fonctionnelle du circuit décrit en langage (V)HDL au moyen de la simulation.

Description: Vérification de la fonction par simulation du circuit complet ou de chaque sous-module. Le simulateur (V)HDL détecte une séquence de sorties ayant pour origine le changement interne des états du circuit suite à l'application des stimuli d'entrée. La vérification de la séquence de sorties détectée peut être effectuée par une séquence prétracée des signaux de sortie (forme d'onde) ou par un environnement spécial qualifié de banc d'essai, installé au cours du processus de conception. Il convient que le simulateur choisi soit qualifié d'éprouvé (par une utilisation antérieure) afin de fournir des résultats corrects et de masquer tout comportement de synchronisation défaillant des signaux (pointes, tracé à trois états), susceptible d'être provoqué par le simulateur proprement dit ou par une modélisation défaillante.

E.6 Essai fonctionnel sur niveau de module

NOTE Voir également E.5 Simulation (V)HDL et E.13 Couverture des scénarios de vérification.

But: Vérification fonctionnelle ascendante (du bas vers le haut).

Description: Vérification de la fonction implémentée, par exemple par simulation, au niveau du module. Le module soumis à l'essai sera instancié dans un environnement d'essai virtuel type qualifié de banc d'essai et stimulé par la trame d'essai contenue dans le code. Une couverture suffisamment importante de la fonction spécifiée, y compris tous les cas spéciaux s'ils existent, est au moins requise. Il convient de recourir à une vérification automatique de la séquence de sorties par le code du banc d'essai de préférence à une inspection manuelle des signaux de sortie.

E.7 Essai fonctionnel au niveau supérieur

NOTE Voir également E.8 "Essai fonctionnel intégré dans l'environnement système".

But: Vérification de l'ASIC (circuit complet).

Description: L'essai a pour objectif la vérification du circuit complet (ASIC).

E.8 Essai fonctionnel intégré dans l'environnement système

NOTE Voir également E.7 "Essai fonctionnel au niveau supérieur".

But: Vérification de la fonction spécifiée intégrée dans l'environnement système.

Description: Cet essai consiste à vérifier la fonctionnalité complète du circuit (ASIC) dans son environnement système, par exemple avec tous les autres composants situés sur les cartes de circuits imprimés ou tout autre élément. Une modélisation de tous les composants appropriés de la carte de circuits imprimés et une simulation des ASIC associée au modèle créé pour vérifier la fonctionnalité appropriée, y compris le comportement de synchronisation, sont recommandées. Un essai fonctionnel exhaustif comprend également l'essai des modules activés uniquement en cas de défaillance.

E.9 Utilisation restreinte des constructions asynchrones

But: Prévention de problèmes de synchronisation typiques en cours de synthèse, prévention de toute ambiguïté au cours de la simulation et synthèse due à une modélisation insuffisante, conception pour testabilité.

Description: Les constructions asynchrones telles que les signaux SET et RESET déduites selon une logique combinatoire sont sensibles en cours de synthèse et génèrent des circuits avec des signaux de pointe ou une séquence de synchronisation inverse. Il convient par conséquent d'éviter ce type de constructions. Une modélisation insuffisante peut également ne pas être interprétée correctement par l'outil de synthèse, ce qui génère des résultats ambigus lors de la simulation. Les constructions asynchrones peuvent par ailleurs être difficilement soumises à essai ou peuvent ne pas l'être du tout, de sorte que la couverture des essais de fabrication et en ligne est effectivement réduite. La réalisation d'une conception entièrement synchrone avec un nombre limité de signaux d'horloge est par conséquent recommandée. Dans les systèmes comportant des horloges polyphasées, il convient de calculer toutes les horloges à partir d'une seule horloge centrale. Il convient que l'entrée d'horloge de logique séquentielle soit toujours fournie exclusivement par le signal d'horloge, qui ne comporte aucune logique combinatoire. Il convient que les entrées asynchrones SET et RESET des cellules séquentielles soient toujours fournies par des signaux synchrones qui ne comportent aucune logique combinatoire. Il convient que les entrées principales SET et RESET soient synchronisées au moyen de deux circuits bistables.

E.10 Synchronisation des entrées primaires et contrôle des métastabilités

But: Evitement de tout comportement de circuit ambigu suite à la violation du temps d'établissement (set-up) et de maintien (hold)

Description: Les signaux d'entrée des périphériques externes sont généralement asynchrones et peuvent changer d'état de manière arbitraire. Un traitement direct de ce type de signaux par les éléments de circuit séquentiel synchrones des ASIC/FPGA, par exemple des circuits bistables, entraîne la violation du temps d'établissement (set-up) et de maintien (hold) qui génère une synchronisation et un comportement fonctionnel imprévisibles de ces mêmes ASIC/FPGA. En définitive, la métastabilité de la mémoire peut se produire. Il convient, par conséquent, de synchroniser chaque signal d'entrée asynchrone eu égard au circuit ASIC synchrone afin d'éviter toute ambiguïté fonctionnelle. Les mesures suivantes sont recommandées:

– Il convient de synchroniser les signaux d'entrée avec deux éléments de mémoire consécutifs (circuits bistables) ou un circuit équivalent afin d'obtenir un comportement fonctionnel prévisible.
– Il convient de synchroniser fondamentalement chaque signal d'entrée asynchrone comme défini ci-dessus, c'est-à-dire que chaque signal asynchrone est relié à exactement un circuit de synchronisation de ce type. La sortie du circuit de synchronisation peut, si nécessaire, être utilisée pour un accès multiple.
– Il convient d'utiliser le circuit de synchronisation pour l'essai de stabilité des signaux de bus parallèle et pour contrôler la cohérence des données au voisinage du point d'échantillonnage.

E.11 Conception pour testabilité

NOTE 1 Voir également E.31 "Mise en oeuvre des structures d'essai".

But: Evitement des structures ne pouvant être soumises à essai ou qui peuvent l'être difficilement en vue d'une couverture importante des essais de fabrication et en ligne.

Description: La conception pour testabilité est régie par l'évitement de:

– constructions asynchrones;
– verrous et de signaux à trois états incorporés;
– logique câblée-et / câblée-ou et d'une logique redondante.

La profondeur combinatoire des sous-circuits joue un rôle important au cours de l'essai. La trame d'essai nécessaire pour un essai complet augmente exponentiellement en fonction de la profondeur combinatoire du circuit. Par conséquent, les circuits dont la profondeur combinatoire est élevée, ne peuvent être soumis à l'essai que difficilement avec des moyens adéquats.

Une approche orientée vers la conception pour testabilité permet d'assurer la réalisation de la couverture d'essai souhaitée. Dans la mesure où la couverture d'essai réelle peut être déterminée à une phase très tardive du processus de conception, une prise en considération insuffisante des problèmes liés à la conception pour testabilité peut réduire de manière considérable la couverture d'essai réalisable, conduisant à un effort supplémentaire.

NOTE 2 La couverture d'essai est généralement déterminée par le pourcentage de défauts de collage détecté.

E.12 Modularisation

NOTE Voir également C.2.8 "Masquage/encapsulation des informations", C.2.9 "Approche modulaire", et E.3 "Description structurée".

But: Description modulaire des fonctions de circuit.

Description: Répartition distincte de la fonctionnalité totale en des modules différents dont les fonctions sont limitées. La transparence des modules avec l'interface définie avec précision est ainsi établie. Chaque sous-système, à tous les niveaux de la conception, est clairement défini et est de taille limitée (quelques fonctions uniquement). Les interfaces entre les sous-systèmes sont maintenues aussi simples que possible et les interactions (c'est-à-dire les données communes, l'échange d'informations) sont réduites au minimum. La complexité des sous-systèmes individuels est aussi limitée.

E.13 Couverture des scénarios de vérification (bancs d'essai)

But: Evaluation quantitative et qualitative des scénarios de vérification appliqués au cours de l'essai fonctionnel.

Description: Il convient de documenter de manière qualitative et/ou quantitative la qualité des scénarios de vérification qui est définie au cours de l'essai fonctionnel, c'est-à-dire la trame d'essai (stimuli) appliquée afin de vérifier la fonction spécifiée, y compris tous les cas spéciaux, s'ils existent. Il convient, lors d'une approche quantitative, de documenter la couverture d'essai obtenue, ainsi que la profondeur des essais fonctionnels appliqués. Il convient que la couverture d'essai résultante satisfasse aux niveaux établis pour chacune des métriques de couverture. Toute exception sera documentée. Dans le cas d'une approche qualitative, il convient d'estimer le nombre de lignes de code, d'instructions ou de cheminements (couverture de code) vérifiés du code de circuit devant faire l'objet d'une vérification.

NOTE Le caractère pertinent de l'analyse exclusive de couverture de code est limité, du fait du parallélisme élevé de la description du matériel. Cette analyse sera justifiée par des contrôles exhaustifs. La couverture de code permet généralement de démontrer le code fonctionnel non couvert.

E.14 Observation des lignes directrices de codage

But: Une observation stricte du style de codage génère un code de circuit correct du point de vue syntaxique et sémantique.

Description: Les règles de codage syntaxique permettent de créer un code facilement lisible et favorisent une meilleure documentation, y compris le contrôle de version. Les règles d'organisation et de commentaire des blocs ou des modules d'instruction peuvent typiquement être mentionnées dans le présent paragraphe.

Les règles de codage sémantique permettent d'éviter les problèmes d'implémentation typiques en utilisant des constructions qui n'entraînent pas une synthèse défaillante associée à une implémentation ambiguë de la fonction du circuit. Des règles typiques sont par exemple l'évitement de constructions asynchrones ou de constructions qui produisent une séquence de synchronisation imprévisible. En général, l'utilisation de verrous ou le couplage de données avec des signaux d'horloge entraînent de telles ambiguïtés.

Les lignes directrices de conception suivantes sont recommandées pour éviter les défaillances de conception systématiques au cours du processus de développement des ASIC. Le style de codage limite, dans un certain sens, l'efficacité de la conception, mais présente toutefois l'avantage d'éviter toute défaillance au cours du processus de développement des ASIC. Ces éléments sont notamment:

– l'évitement de toute défectuosité ou défaillance de codage typique;
– l'utilisation limitée de constructions problématiques qui génèrent des résultats de synthèse ambigus;
– la conception pour testabilité;

– l'emploi d'un code transparent et pratique.

Exemple d'un style de codage

1) Il convient que le code comporte autant de commentaires que nécessaire pour comprendre la fonction et les détails d'implémentation. Les conventions utilisées doivent être définies avant le début de la conception. Il convient de vérifier la conformité des conventions définies au cours de la phase de conception.

   1.1 Les en-têtes standard comportent l'historique, les références croisées à la spécification, les données liées à la responsabilité et d'accompagnement de la conception telles que le numéro de version, les demandes de changement, etc.

   1.2 Des modèles facilement lisibles: Il convient de décrire les processus équivalents avec la même procédure, c'est-à-dire l'utilisation de modèles prédéfinis pour les processus récurrents (si-alors-sinon, pour etc.).

   1.3 Une règle d'affectation des noms précise et lisible, par exemple majuscule/minuscule, préfixe et suffixe, différenciation précise entre le nom de port, les signaux internes, les constantes, les variables et un niveau de faible activité (xxx_n) etc.

   1.4 Il convient de limiter la taille des modules et le nombre des ports par module afin d'accroître la lisibilité du code.

   1.5 Développement d'un code structurel et défensif, par exemple, il convient que l'information d'état soit encapsulée dans le FSM (masquage d'information) afin de pouvoir modifier le code facilement.

   1.6 Il convient d'effectuer des contrôles de vraisemblance tels que le contrôle des limites, etc.

   1.7 Evitement des constructions /instructions suivantes:

   – utilisation d'une limite ascendante (x à y) pour les signaux de bus;
   – instruction « Désactiver » en langage Verilog (correspond à l'instruction « aller à »);
   – tableaux multidimensionnels (> 2), enregistrements;
   – combinaison des types de données signées et non signées.

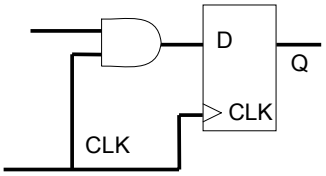
2) Conception synchrone complète (les horloges issues des horloges centrales sont admises):

   2.1 Il convient que les sorties de modules soient synchronisées. Cette synchronisation facilite également l'analyse de testabilité et de temporisation statique.

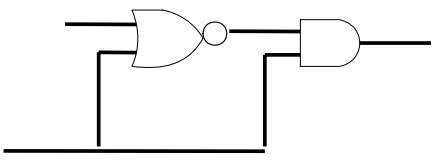
   2.2 Il convient d'éviter les horloges commandées (par une porte).

3) L'évitement de tout couplage des données avec une horloge augmente la testabilité, la reproductibilité entre les données préalables et postérieures à la topologie et la conformité avec le comportement de la logique à résistances et transistors.

4) Il convient d'éviter la logique redondante qui ne peut être soumise à essai:



Couplage des signaux de données et d'horloge



Logique redondante

5) Il convient d'éviter l'utilisation de boucles d'asservissement dans la logique combinatoire dans la mesure où ces dernières génèrent une conception instable et ne pourront pas être soumises à essai.

6) Une conception à scan intégral est recommandée.

7) La non utilisation de verrous augmente la testabilité et réduit les contraintes de synchronisation lors de la synthèse.

8) Il convient de synchroniser la réinitialisation principale et toutes les entrées asynchrones avec deux éléments de mémoire consécutifs (circuits bistables) ou un (des) circuit(s) équivalent(s) (métastabilité).

9) Il est recommandé d'éviter toute initialisation/réinitialisation asynchrone à l'exception de la réinitialisation principale.

10) Il convient que les signaux au niveau du port de module soient du type std_logic ou std_logic_vector.

E.15 Application du système de vérification du code

But: Vérification automatique des règles de codage (style de codage) par un outil de vérification du code.

Description: L'application du système de vérification du code permet, dans une large mesure, d'observer automatiquement le style de codage et génère une documentation en ligne. Toutefois, le système de vérification automatique du code peut généralement vérifier la syntaxe et la sémantique du code. Il convient par conséquent d'accompagner l'application de ce type d'outils par l'extension des règles de codage générales (spécifiques à l'outil), associées aux règles de codage spécifiques au projet que le concepteur doit mettre en œuvre et évaluer séparément.

E.16 Programmation défensive

Voir C.2.5.

E.17 Documentation des résultats de simulation

But: Documentation de toutes les données nécessaires à une simulation réussie afin de vérifier la fonction de circuit spécifiée.

Description: Il convient que toutes les données nécessaires à la simulation fonctionnelle au niveau du module, de la puce ou du système fassent l'objet d'une documentation et d'un archivage appropriés, avec les objectifs suivants:

– Répéter la simulation à toute phase ultérieure selon une procédure clé en main.

– Démontrer la cohérence et l'exhaustivité de toutes les fonctions spécifiées.

Il convient d'archiver la base de données suivante à cette fin:

– Réglage de simulation, y compris le logiciel complet des outils appliqués, par exemple simulateur, synthétiseur avec version correspondante et la bibliothèque de simulation nécessaire.

– Fichier journal de la simulation comportant les détails complets concernant le temps de simulation, les outils appliqués avec la version concernée et le rapport complet de la solution de rechange si ce dernier était nécessaire.

– Tous les résultats pertinents de la simulation, y compris la fluence de signal, notamment dans le cas de l'inspection manuelle et de la documentation des résultats obtenus.

E.18 Inspection de code

NOTE 1 Voir également C.5.14 "Inspections formelles".

But: Revue de la description du circuit.

Description: Il convient d'effectuer la revue de la description du circuit comme suit

– vérification du style de codage.
– vérification de la fonctionnalité décrite par rapport à la spécification.
– vérification du codage défensif et du traitement des erreurs et des exceptions.

NOTE 2 En cas de non réalisation de la simulation (V)HDL, il convient que l'exhaustivité de l'inspection du code et les résultats obtenus aient une qualité équivalente à celle qui serait atteinte avec une simulation (V)HDL.

E.19 Lectures croisées

NOTE 1 Voir également C.5.15 "Lectures croisées".

But: Vérification de la description du circuit par lectures croisées.

Description: Une lecture croisée de code est réalisée par une équipe de lecture croisée qui sélectionne un jeu réduit de cas d'essai représentatifs des jeux d'entrées et de sorties correspondantes prévues du programme. Les données d'essai sont alors tracées manuellement en suivant la logique du programme.

NOTE 2 Il convient, en tant que mesure autonome, de l'appliquer uniquement aux circuits à très faible complexité. En cas de défaillance de la simulation (V)HDL, il convient que l'exhaustivité des lectures croisées et les résultats obtenus aient une qualité équivalente à celle qui sera atteinte avec une simulation (V)HDL.

Référence:

CEI 61160:2005, *Revue de conception*

E.20 Application de fonctions logicielles validées

But: Eviter toute défaillance au cours de la mise en œuvre des fonctions logicielles par application des fonctions logicielles validées.

Description: Si le fournisseur valide les fonctions logicielles, il convient de satisfaire aux exigences suivantes:

– Il convient de valider les fonctions logicielles pour le fonctionnement du système relatif à la sécurité, en ayant au moins un niveau d'intégrité de sécurité équivalent ou supérieur à celui du système planifié.
– Il convient de satisfaire à toutes les hypothèses et autres restrictions, nécessaires pour la validation des fonctions logicielles.
– Il convient que tous les documents nécessaires pour la validation des fonctions logicielles puissent être obtenus facilement, voir également E.17 "Documentation des résultats de simulation".
– Il convient de respecter strictement chaque spécification du fournisseur et de documenter les preuves de conformité.

E.21 Validation des fonctions logicielles

NOTE Voir également E.6 "Essai fonctionnel sur niveau de module".

But: Eviter toute défaillance au cours de la mise en œuvre des fonctions logicielles par validation de ces dernières au cours du cycle de vie de conception.

Description: Si les fonctions logicielles ne sont pas développées de manière explicite pour une utilisation dans un système relatif à la sécurité, il convient de valider le code généré dans les mêmes conditions que celles qui s'appliquent pour la validation de tout code source. Cela signifie qu'il convient de définir et de mettre en œuvre tous les cas d'essai potentiels. Il convient alors de déduire la vérification fonctionnelle par simulation.

E.22 Simulation de la liste d'interconnexions des portes, afin de vérifier les contraintes de synchronisation

But: Vérification indépendante de la contrainte de synchronisation réalisée lors de la synthèse.

Description: Simulation de la liste d'interconnexions des portes générée par la synthèse, y compris la post-annotation des retards de ligne et des temps de propagation. Il convient de calculer les stimuli afin de stimuler le circuit de manière à ce qu'il couvre un pourcentage élevé des contraintes de synchronisation et inclue tous les chemins de synchronisation les plus défavorables. En général, les stimuli nécessaires pour effectuer l'essai fonctionnel sur niveau de module (voir E.6) ou l'essai fonctionnel (voir E.7) fournissent des critères appropriés pour le choix des stimuli, sous réserve qu'une couverture d'essai suffisante puisse être revendiquée lors de l'essai fonctionnel. Il convient de soumettre à essai le circuit dans les conditions les plus et les moins favorables à la fréquence d'horloge spécifiée maximale.

La vérification de synchronisation peut être effectuée par contrôle automatique du temps d'établissement (set-up) et de maintien (hold) des éléments de mémoire (circuits bistables) de la bibliothèque cible, ainsi que par la vérification fonctionnelle du circuit. Il convient d'effectuer la vérification fonctionnelle principalement par l'observation des sorties de la puce. Cette opération peut être automatisée par une comparaison des signaux de sortie du circuit avec un modèle de référence ou un code source (V)HDL adéquat du circuit. Cet essai est qualifié d'essai de régression; il convient de l'utiliser de préférence à une vérification manuelle des signaux de sortie.

NOTE L'application de cette mesure permet de vérifier le comportement de synchronisation des seuls chemins qui sont effectivement stimulés lors de la simulation et, par conséquent, la mesure adaptée ne peut fournir une analyse de synchronisation complète du circuit en général.

E.23 Analyse statique du temps de propagation (STA)

But: Vérification indépendante des contraintes de synchronisation réalisées lors de la synthèse.

Description: L'analyse statique de synchronisation (STA) examine tous les chemins d'une liste d'interconnexions (circuit) générée par l'outil de synthèse compte tenu de la post-annotation, c'est-à-dire les retards de ligne estimés par ce même outil, ainsi que les temps de propagation sans effectuer la simulation réelle. Cette analyse permet en général un examen complet de la contrainte de synchronisation du circuit entier. Il convient d'analyser le circuit à soumettre à essai dans les conditions les plus et les moins favorables en utilisant la fréquence d'horloge spécifiée maximale et en tenant compte de l'instabilité de l'horloge et du décalage du cycle d'utilisation applicables. Le nombre de chemins de synchronisation non adaptés peut être limité à un niveau minimum par l'adoption d'une technique de conception appropriée. Il est recommandé d'étudier, analyser et définir la technique utilisée, qui permet d'obtenir des résultats facilement lisibles avant de lancer la conception.

NOTE Il peut être supposé que la STA couvre explicitement tous les chemins de synchronisation existants si:

a) Les contraintes de synchronisation sont spécifiées de manière appropriée.

b) Le circuit en essai comporte uniquement les chemins de synchronisation qui peuvent être analysés par les outils STA, ce qui est généralement le cas avec les circuits entièrement synchrones.

E.24 Vérification de la liste d'interconnexions des portes par rapport à un modèle de référence par simulation

But: Vérification de l'équivalence fonctionnelle de la liste d'interconnexions des portes synthétisée.

Description: Simulation de la liste d'interconnexions des portes par l'outil de synthèse. Les stimuli appliqués pour la vérification du circuit par simulation correspondent exactement aux stimuli appliqués lors de l'essai fonctionnel sur niveau de module (E.6) ou de l'essai fonctionnel (E.7) pour la vérification de la fonction au niveau du module et au niveau supérieur respectivement. Il convient d'effectuer la vérification fonctionnelle principalement par l'observation des sorties de la puce. Cette opération peut être automatisée par une comparaison des signaux de sortie du circuit avec un modèle de référence ou un code source (V)HDL adéquat du circuit. Cet essai est qualifié d'essai de régression; il convient de l'utiliser de préférence à une vérification manuelle des signaux de sortie.

NOTE L'application de cette mesure permet de vérifier le comportement fonctionnel des seuls chemins qui sont effectivement stimulés lors de la simulation. La couverture d'essai ne peut, par conséquent, être aussi appropriée que lors de l'essai fonctionnel d'origine au niveau du module et au niveau supérieur respectivement. Il est possible de compléter cette mesure par un contrôle d'équivalence formel. Dans tous les cas, il convient d'effectuer une vérification fonctionnelle du code source (V)HDL à l'aide de la liste d'interconnexions définitive générée par l'outil de synthèse.

E.25 Comparaison de la liste d'interconnexions des portes avec le modèle de référence (contrôle d'équivalence formel)

But: Vérification de l'équivalence fonctionnelle indépendante de la simulation.

Description: Comparaison de la fonctionnalité de circuit décrite par le code source (V)HDL avec la fonctionnalité de circuit de la liste d'interconnexions de portes générée par la synthèse. Les outils basés sur le principe d'équivalence formel sont capables de vérifier l'équivalence fonctionnelle d'une forme de représentation différente du circuit, par exemple description (V)HDL ou description de la liste d'interconnexions. L'application de cette mesure rend inutile une simulation fonctionnelle et permet d'effectuer un contrôle fonctionnel indépendant. L'application satisfaisante de cette mesure ne peut être garantie que si l'outil appliqué est capable de démontrer une équivalence complète et si toutes les divergences consignées sont évaluées par inspection manuelle ou de manière automatique.

NOTE L'avantage est de combiner cette mesure avec la "Vérification de la liste d'interconnexions des portes par rapport à un modèle de référence par simulation" (E.24). Dans tous les cas, il convient d'effectuer une vérification fonctionnelle du code source (V)HDL à l'aide de la liste d'interconnexions définitive générée par l'outil de synthèse.

E.26 Vérification des exigences et des contraintes des fournisseurs

But: Evitement de toute défaillance en cours de fabrication par vérification des exigences des fournisseurs.

Description: Une vérification attentive des exigences et des contraintes des fournisseurs, par exemple entrance et sortance minimales et maximales, longueur de câble maximale (retard de ligne), pente maximale des signaux, décalage de l'horloge, etc., par l'outil de synthèse, améliore la fiabilité du produit. Outre l'importance des exigences relatives aux processus de fabrication, leur sévérité a une grande influence sur la validité des modèles appliqués, utilisés pour la simulation. Par conséquent, tout manquement aux exigences et aux contraintes des fournisseurs produit des résultats de simulation erronés qui génèrent une fonctionnalité non souhaitée.

E.27 Documentation des contraintes, résultats et outils de synthèse

But: Documentation de toutes les contraintes définies nécessaires pour une synthèse optimale afin de générer la liste d'interconnexions de portes définitive.

Description: La documentation de tous les résultats et contraintes de synthèse est indispensable pour les raisons suivantes:

– reproduire la synthèse à toute phase ultérieure.

– générer des résultats de synthèse indépendants pour vérification.

Les documents essentiels sont les suivants:

– Réglage de la synthèse, y compris les outils et logiciels de synthèse appliqués avec la version réelle, la bibliothèque de synthèse appliquée et les contraintes et scripts définis.

– Fichier journal de synthèse avec marquage temporel, outil appliqué avec version et documentation complète de la synthèse.

– La liste d'interconnexions générée avec les temporisations estimées (fichier SDF).

E.28 Application d'outils de synthèse éprouvés (par une utilisation antérieure)

But: Conversion à l'aide d'un outil de la description (V)HDL d'un circuit dans la liste d'interconnexions de portes.

Description: Mise en correspondance à l'aide d'un outil du code source (V)HDL de la fonctionnalité du circuit par connexion des portes et des primitives de circuit appropriées de la bibliothèque ASIC cible. L'implémentation choisie parmi plusieurs implémentations possibles qui satisfont à la fonctionnalité souhaitée, dépend du résultat le plus optimal déduit par les contraintes de synthèse telles que la synchronisation (fréquence d'horloge) et la surface occupée.

E.29 Application de bibliothèques cibles éprouvées (par une utilisation antérieure)

NOTE Voir également E.4, Outils éprouvés (par une utilisation antérieure).

But: Evitement des défaillances systématiques dues à une bibliothèque cible défectueuse.

Description: La bibliothèque cible de synthèse et de simulation pour le développement d'un ASIC est issue d'une base de données commune et n'est pas, par conséquent, indépendante. Une défaillance systématique telle que:

– une ambiguïté entre le comportement réel et modélisé des éléments de circuit,

– une modélisation insuffisante, par exemple du temps d'établissement (set-up) et de maintien (hold),

constitue l'un des exemples types.

Il convient par conséquent d'utiliser uniquement des technologies et des bibliothèques cibles éprouvées (par une utilisation antérieure) pour la conception des ASIC qui exécutent des fonctions de sécurité. Cela signifie que:

– l'application de bibliothèques cibles qui ont été utilisées pendant une durée suffisamment longue dans des projets ayant une complexité et une fréquence d'horloge comparables;

– la disponibilité de la technologie et de la bibliothèque cible correspondante pendant une période suffisamment longue, de sorte qu'une précision de modélisation suffisante de la bibliothèque puisse être envisagée.

E.30 Procédures à base de scripts

But: Reproductibilité des résultats et automatisation des cycles de synthèse.

Description: Contrôle automatique à base de scripts des cycles de synthèse, y compris la définition des contraintes appliquées. Outre une documentation précise d'une contrainte de synthèse complète, ce contrôle permet de reproduire la liste d'interconnexions après la modification du code source (V)HDL dans des conditions identiques.

E.31 Mise en œuvre des structures d'essai

But: Conception d'ASIC pouvant être soumis à l'essai, garantissant l'essai de fabrication final.

Description: La conception pour testabilité permet l'utilisation de circuits pouvant être facilement soumis à l'essai par la mise en œuvre de différentes structures d'essai, comme par exemple:

– Scan path: Dans une technique de "scan", tout (conception à "scan" complet) ou partie (conception à "scan" partiel) des circuits bistables sont reliés dans une chaîne unique ou dans plusieurs chaînes constituant une chaîne de registres à décalage. La technique du scan path permet de générer automatiquement une trame d'essai de la logique intégrale d'un circuit. L'outil de génération de la trame d'essai est appelé générateur de vecteurs d'essai automatique (ATPG)[9]. L'application de la technique du scan path améliore considérablement la testabilité d'un circuit et permet une couverture d'essai de plus de 98 % avec un effort raisonnable. Il est par conséquent recommandé d'appliquer, si possible, un scan path complet.

– Arbre NON-ET: Dans un arbre NON-ET, toutes les entrées principales d'un circuit sont connectées en cascade afin de construire une chaîne. L'application d'une trame d'essai appropriée (walking bit) permet de soumettre à essai le comportement de commutation (niveau de synchronisation et de déclenchement) des entrées. L'arbre NON-ET permet une caractérisation directe des entrées primaires. Son application est recommandée, si le comportement de commutation du circuit ne peut être soumis à l'essai d'une autre manière.

– Auto-test intégré (BIST)[10]: L'auto-test du circuit, et notamment l'auto-test de la mémoire intégrée, peut être effectué de manière très efficace par la mise en œuvre d'un générateur de vecteurs d'essai intégré. Le BIST permet une vérification automatique de la structure du circuit par l'application d'une trame d'essai pseudo-aléatoire et l'évaluation de la signature de la structure de circuit mise en œuvre. L'auto-test intégré est recommandé comme mesure supplémentaire, notamment pour l'essai de mémoire. Un auto-test intégré peut se substituer à l'essai du scan path.

– Essai du courant de repos (de fuite) (test IDDQ: essai par mesure du courant d'alimentation au repos): Un circuit CMOS statique consomme du courant principalement lors d'une commutation. Un circuit absolument exempt de défaut consomme par conséquent une quantité négligeable de courant (< 1µA, courant de fuite) tant que la trame d'essai est maintenue fixe. L'essai par mesure du courant d'alimentation au repos est très efficace et assure une couverture d'essai de plus de 50 % juste après l'application d'un couple de trames d'essai. L'essai par mesure du courant d'alimentation au repos peut être appliqué sur des trames d'essai fonctionnel, ainsi que sur des trames d'essai synthétisé générées par l'outil ATPG. Cette méthode d'essai s'est révélée très utile dans la pratique, et est capable de détecter toute défaillance que d'autres essais ne détectent que rarement, voire ne peuvent pas détecter. Il convient par conséquent d'appliquer cette mesure en complément des essais de fabrication habituels.

---

9 ATPG = *Automatic Test Pattern Generator.*

10 BIST = *Built-In Self Test*.

– Essai périphérique: Architecture d'essai mise en œuvre pour la vérification de l'interconnexion des composants sur une carte à circuit imprimé selon la norme JTAG. Le même principe peut également être appliqué pour vérifier l'interconnexion des modules au niveau de la puce. L'essai périphérique est recommandé principalement pour améliorer la testabilité de la carte à circuit imprimé.

E.32 Estimation de la couverture d'essai par simulation

But: Détermination de la couverture d'essai obtenue par l'architecture d'essai mise en œuvre pendant l'essai de fabrication.

Description: La couverture d'essai obtenue par l'essai du scan path, le BIST, la trame d'essai fonctionnel ou toutes autres mesures peut être déterminée par simulation des anomalies. Au cours de la simulation des anomalies, une trame d'essai est appliquée à un circuit auquel les anomalies sont intégrées. Une réponse erronée du circuit aux stimuli appliqués correspond aux anomalies intégrées et contribue ainsi à la couverture d'essai. La simulation des anomalies permet la détection des défauts de collage à 1 et à 0, et la couverture d'essai obtenue représente la qualité de la trame d'essai appliquée. La simulation des anomalies peut en général se révéler très efficace dans la détection des anomalies associées à la logique qui ne constitue pas une partie intégrante du scan path, par exemple dans le cas de scan paths partiels.

E.33 Estimation de la couverture d'essai par application de l'outil ATPG

But: Détermination de la couverture d'essai susceptible d'être obtenue par la trame d'essai synthétisée (scan path, BIST) pendant l'essai de fabrication.

Description: Il existe actuellement un grand nombre de procédures qui génèrent des trames d'essai pseudo-aléatoires ou algorithmiques pour un circuit implémenté à l'aide de la technique du scan path. L'outil de synthèse tel que l'outil ATPG génère, lors de la synthèse, un nombre important d'anomalies non détectées. La couverture d'essai peut ainsi être estimée, et permet de définir la limite inférieure de la couverture d'essai obtenue avec la trame d'essai appliquée. Il est important de noter que la couverture d'essai se limite à la logique de circuit, couverte par le scan path. Les modules tels que la mémoire, le BIST ou une partie des circuits qui ne sont pas intégrés au scan path, ne sont pas pris en compte dans l'estimation de la couverture d'essai.

E.34 Justification des fonctions matérielles appliquées éprouvées (par une utilisation antérieure)

But: Eviter toute défaillance systématique lors de l'application des fonctions matérielles.

Description: Une fonction matérielle est généralement considérée comme une boîte noire représentant la fonctionnalité souhaitée, et est constituée d'une base de données topologique, relevant de la technologie cible, qui fournit le composant de circuit souhaité. La défaillance fonctionnelle potentielle peut être traitée comme les composants discrets tels que les microprocesseurs normalisés, les mémoires, etc. Il est possible d'utiliser ces fonctions matérielles sans vérifier leur fonctionnalité correcte si, pour la technologie cible appliquée, la fonction matérielle peut être considérée comme un composant éprouvé (par une utilisation antérieure). Il convient alors de soumettre le reste du circuit à une vérification intensive.

E.35 Application de fonctions matérielles validées

NOTE Voir également E.6 Essai fonctionnel sur niveau de module.

But: Eviter toute défaillance systématique lors de l'application des fonctions matérielles

Description: En raison de la nature complexe desdites fonctions et des contraintes supposées, il convient que la validation des fonctions soit effectuée par les fournisseurs pendant la phase de conception sur la base du code source (V)HDL. La validation peut être justifiée uniquement pour la configuration et la technologie cible du composant appliqué.

E.36 Essai en ligne des fonctions matérielles

NOTE Voir également E.13 "Couverture des scénarios de vérification (bancs d'essai)".

But: Eviter toute défaillance systématique lors de l'application des fonctions matérielles.

Description: Vérification du fonctionnement et de l'implémentation corrects des fonctions matérielles utilisées par application d'essais en ligne. L'application de cette mesure rend nécessaire l'application d'un essai efficace, tout comme il convient de documenter l'évaluation du concept d'essai appliqué.

E.37 Vérification des règles de conception (DRC)[11]

But: Vérification des règles de conception du fournisseur.

Description: Vérification de la topologie générée par rapport aux règles de conception du fournisseur, par exemple longueurs de câbles minimales et maximales et plusieurs règles concernant le placement des structures de topologie. Il convient de documenter de manière détaillée un cycle complet et correct de vérification des règles de conception.

E.38 Vérification de la topologie par rapport à la liste d'interconnexions (LVS)[12]

But: Vérification indépendante de la topologie.

Description: Cette fonction extrait la fonctionnalité du circuit de la base de données topologique et compare les éléments de circuit extraits, y compris les interconnexions avec la liste d'interconnexions d'entrée. Cette fonction garantit l'équivalence de la topologie du circuit avec la liste d'interconnexions qui spécifie la fonctionnalité du circuit. Il convient de documenter de manière détaillée un cycle complet et correct de vérification de la topologie par rapport à la liste d'interconnexions.

E.39 Marge supplémentaire (> 20 %) pour les technologies de système appliquées depuis moins de trois ans

But: Garantir la robustesse de la fonction de circuit implémentée même en cas de forte variation des systèmes et des paramètres.

Description: Le comportement réel du circuit est défini par le nombre d'effets physiques de superposition, notamment pour les petites structures (par exemple en dessous de 0,5 µm). Le manque de connaissances détaillées et des simplifications nécessaires ne permet pas, en tout état de cause, de déduire un modèle exact d'éléments de circuit. Les retards de ligne sont de plus en plus importants du fait de la réduction du nombre des structures géométriques. Les décalages de signal dans le temps le long des connexions et les capacités de couplage croisé entre les câbles se développent en proportion. Les décalages de signal dans le temps ne sont plus négligeables par comparaison aux temps de propagation. Les

---

11 DRC = *Design Rule Check*.

12 LVS = *Layout Versus Schematic*.

retards de ligne estimés illustrent le risque accru que présente la réduction du nombre de structures géométriques.

Il est par conséquent recommandé de prévoir une marge adéquate (> 20 %) par rapport aux contraintes de synchronisation maximales et minimales pour les circuits conçus au moyen de processus appliqués depuis moins de trois ans, afin de garantir la mise en oeuvre correcte de la fonctionnalité de circuit en présence de paramètres à forte variation lors de la fabrication, ou en raison de l'absence d'une modélisation précise.

E.40 Essai de déverminage

But: Garantir la robustesse de la puce fabriquée. Elimination de défaillances précoces. La robustesse des produits à base de puce nue ne doit pas être vérifiée par déverminage, par exemple par des méthodes de contrainte au niveau du circuit intégré.

Description: Il convient d'effectuer l'essai de déverminage à la température de service maximale tolérable (125 °C généralement). La durée de l'essai dépend du niveau SIL visé ou de recommandations de déverminage spécifiques, par exemple du fabricant des ASIC. Le déverminage peut être utilisé pour:

– éliminer des défaillances précoces (début de la courbe en baignoire avec diminution du taux de défaillance);
– démontrer que les défaillances précoces sont déjà éliminées pendant la fabrication et les essais (c'est-à-dire que les dispositifs hors de la chaîne de production se situent déjà dans la zone de taux de défaillance constant de la courbe du tube de bain.

E.41 Application d'une série de dispositifs éprouvés (par une utilisation antérieure)

But: Garantir la fiabilité des puces fabriquées.

Description: Il convient que le fabricant d'un dispositif de sécurité ait une expérience suffisante en matière d'application de la technologie utilisée avec les dispositifs programmables et des outils de développement concernés.

E.42 Processus de fabrication de puces éprouvé (par une utilisation antérieure)

But: Garantir la fiabilité des puces fabriquées.

Description: Un processus de fabrication éprouvé (par une utilisation antérieure) est caractérisé par une expérience suffisante en matière de fabrication en série.

E.43 Contrôle qualité du processus de fabrication

Les mesures de qualité et les mécanismes de contrôle au cours du processus de fabrication des dispositifs garantissent un contrôle permanent du processus. Citons, à titre d'exemple, le contrôle optique ou électrique des structures d'essai, les évaluations des biais de température et d'humidité ou l'essai de cycle de température (voir CEI 60068-2-1, CEI 60068-2-2 etc.).

E.44 Certificat de qualité de fabrication du dispositif

La qualité du dispositif sera démontrée en effectuant des essais sélectionnés de contrainte partielle, par exemple évaluations des biais de température et d'humidité ou essais de variation de température (voir CEI 60068-2-1, CEI 60068-2-2 etc.). Le fabricant du dispositif fournira les preuves de cette qualité.

E.45 Certificat de qualité fonctionnelle du dispositif

Tous les dispositifs feront l'objet d'un essai fonctionnel. Le fabricant du dispositif fournira les preuves de cet essai.

E.46 Normes de qualité

Il convient que le fabricant des ASIC prévoit un système de management de la qualité suffisant, par exemple documenté dans un manuel de qualité & fiabilité par exemple, certification ISO 9000 ou SSQA, Evaluation normalisée de la qualité des fournisseurs.

Annexe F
(informative)

Définitions des propriétés des phases du cycle de vie du logiciel

Tableau F.1 – Spécification des exigences pour la sécurité du logiciel

(voir 7.2 et le Tableau C.1 de la CEI 61508-3)

	Propriété	Définition
1.1	Complétude par rapport aux besoins de sécurité devant être traités par le logiciel	La spécification des exigences pour la sécurité du logiciel traite de tous les besoins et contraintes de sécurité résultant des phases antérieures du cycle de vie de sécurité et alloués au logiciel. Les besoins et les contraintes de sécurité sont habituellement énoncés dans les éléments entrants de l'activité de spécification des exigences pour la sécurité du logiciel. Ceci peut inclure la spécification de ce que le logiciel ne doit pas effectuer ou doit éviter.
1.2	Exactitude par rapport aux besoins de sécurité devant être traités par le logiciel	La spécification des exigences pour la sécurité du logiciel fournit une réponse appropriée aux besoins et contraintes de sécurité affectés au logiciel. L'objectif est d'assurer que la spécification garantira effectivement la sécurité dans toutes les conditions nécessaires.
1.3	Insensibilité aux anomalies de spécification intrinsèques, y compris exemption de toute ambiguïté	Complétude et cohérence internes de la spécification des exigences pour la sécurité du logiciel: fourniture de toutes les informations nécessaires pour toutes les fonctions et situations qui peuvent être déduites de ses instructions; absence d'instructions contradictoires ou incohérentes. Contrairement à la complétude et à la cohérence par rapport aux besoins de sécurité, la complétude et la cohérence internes peuvent être évaluées sur la seule foi de la spécification des exigences pour la sécurité du logiciel.
1.4	Intelligibilité des exigences de sécurité	La spécification des exigences pour la sécurité du logiciel est pleinement intelligible sans effort excessif de l'ensemble des individus qui doivent la lire, même s'ils n'ont pas pris part plus tôt au projet, mais sous réserve de disposer des connaissances requises. Un objectif important est de faciliter la vérification et, éventuellement, les modifications.
1.5	Insensibilité aux perturbations préjudiciables des fonctions non liées à la sécurité avec les besoins de sécurité devant être traités par le logiciel	La spécification des exigences pour la sécurité du logiciel exclut les exigences qui ne sont pas nécessaires à la sécurité de l'EUC. L'objectif consiste à éviter toute complexité inutile dans la conception et l'implémentation du logiciel, de manière à réduire les risques d'anomalies et d'application de fonctions non importantes pour la sécurité et qui sont préjudiciables, voire qui menacent, celles importantes pour la sécurité.
1.6	Aptitude à fournir une base pour la vérification et la validation	La spécification des exigences pour la sécurité du logiciel génère des essais et des examens qui produisent des preuves objectives de la satisfaction du logiciel à cette même spécification.

Tableau F.2 – Conception et développement du logiciel : conception d'architecture du logiciel

(voir 7.4.3 et le Tableau C.2 de la CEI 61508-3)

	Propriété	Définition
2.1	Complétude par rapport à la spécification des exigences pour la sécurité du logiciel	La conception d'architecture du logiciel traite de tous les besoins et contraintes de sécurité mentionnés dans la spécification des exigences pour la sécurité du logiciel.
2.2	Exactitude par rapport à la spécification des exigences pour la sécurité du logiciel	La conception d'architecture du logiciel apporte une réponse appropriée aux exigences pour la sécurité du logiciel spécifiées.
2.3	Insensibilité aux anomalies de conception intrinsèques	La conception d'architecture du logiciel et la documentation de conception sont insensibles aux anomalies susceptibles d'être identifiées indépendamment de toute exigence pour la sécurité du logiciel spécifiée. Exemples: arrêts nets, accès à des ressources non autorisées, fuites de ressources, incomplétude intrinsèque (c'est-à-dire non traitement de toutes les situations issues de la conception proprement dite).
2.4	Simplicité et intelligibilité. Prévisibilité du comportement.	Conception d'architecture du logiciel permettant une prévision correcte et précise, dans toutes les situations spécifiées, du fonctionnement du logiciel. Ces situations comprennent, notamment, des erreurs et des défaillances. La prévisibilité implique plus particulièrement que le fonctionnement ne dépend pas d'éléments qui ne peuvent pas être contrôlés par les concepteurs ou les utilisateurs.
2.5	Conception vérifiable et pouvant être soumise à essai	La conception d'architecture du logiciel et la documentation de conception permettent et facilitent la production de preuves crédibles attestant que toutes les exigences pour la sécurité du logiciel sont dûment prises en compte par la conception et que cette dernière est insensible aux anomalies intrinsèques. La vérifiabilité peut impliquer des propriétés dérivées telles que simplicité, modularité, transparence, testabilité, prouvabilité, etc., selon les techniques de vérification employées.
2.6	Tolérance aux anomalies	La conception d'architecture du logiciel garantit que le logiciel aura un comportement sûr en présence d'erreurs (erreurs internes, erreurs des opérateurs ou des systèmes externes). Une conception défensive peut être active ou passive. Les conceptions défensives actives peuvent inclure des caractéristiques telles que la détection, le rapport et le confinement des erreurs, la dégradation progressive et l'épuration des effets secondaires éventuels indésirables préalablement à la reprise d'un fonctionnement normal. Les conceptions défensives passives incluent les caractéristiques qui garantissent l'imperméabilité à des types d'erreurs ou de conditions particuliers (entrées innombrables, dates et heures particulières) sans que le logiciel n'exécute une action spécifique.
2.7	Défense contre les défaillances de cause commune dues à des événements externes	La conception d'architecture du logiciel facilite l'identification des modes de défaillance de cause commune et des mesures de prévention efficaces contre les défaillances.

Tableau F.3 – Conception et développement du logiciel : outils de support et langage de programmation

(voir 7.4.4 et le Tableau C.3 de la CEI 61508-3)

	Propriété	Définition
3.1	Aide à la production du logiciel avec les propriétés du logiciel requises	Moyens de détection d'erreurs ou de suppression des constructions sujettes à l'erreur.
3.2	Clarté de l'exploitation et de la fonctionnalité de l'outil	Couverture complète et retour d'information concernant tous les aspects de l'exploitation de l'outil.
3.3	Exactitude et répétabilité de la sortie	Cohérence et précision de la sortie de l'outil pour toute entrée donnée.

Tableau F.4 – Conception et développement du logiciel : conception détaillée

(voir 7.4.5, 7.4.6 et Tableau C.4 de la CEI 61508-3)

	Propriété	Définition
4.1	Complétude par rapport à la spécification des exigences pour la sécurité du logiciel	Adoption de méthodes de conception et de production de logiciel détaillées garantissant que le logiciel obtenu traite de tous les besoins et contraintes de sécurité affectés au logiciel.
4.2	Exactitude par rapport à la spécification des exigences pour la sécurité du logiciel	Il existe des preuves spécifiques attestant que les exigences de sécurité affectées au logiciel ont été satisfaites par le logiciel développé.
4.3	Insensibilité aux anomalies de conception intrinsèques	Le logiciel développé est insensible aux anomalies intrinsèques. Exemples: arrêts nets, accès à des ressources non autorisées, fuites de ressources.
4.4	Simplicité et intelligibilité Prévisibilité du comportement	Le comportement du logiciel développé est prévisible par des essais et une analyse objectifs et probants.
4.5	Conception vérifiable et pouvant être soumise à essai	Le logiciel développé est vérifiable et peut être soumis à essai.
4.6	Tolérance aux anomalies / Détection d'anomalies	Les techniques et les conceptions garantissent que le logiciel développé aura un comportement sûr en présence d'erreurs.
4.7	Insensibilité aux défaillances de cause commune	Les techniques et les conceptions identifient les modes de défaillances de cause commune et fournissent des mesures de prévention efficaces contre les défaillances du logiciel.

Tableau F.5 – Conception et développement du logiciel : essai de module logiciel et intégration

(voir 7.4.7, 7.4.8 et le Tableau C.5 de la CEI 61508-3)

	Propriété	Définition
5.1	Complétude de l'essai et de l'intégration par rapport aux spécifications de conception	L'essai du logiciel permet un examen du comportement de ce dernier suffisamment détaillé pour assurer que toutes les exigences de la spécification de conception du logiciel ont été traitées.
5.2	Exactitude de l'essai et de l'intégration par rapport aux spécifications de conception (réalisation satisfaisante)	La tâche portant sur l'essai du module est achevée, et il existe des preuves spécifiques attestant que les exigences de sécurité ont été satisfaites.
5.3	Répétabilité	La répétition des évaluations individuelles effectuées comme partie intégrante de l'essai du module et de l'intégration permet d'obtenir des résultats cohérents.
5.4	Configuration d'essai définie avec précision	L'essai du module et l'intégration ont été appliqués à la version appropriée des composants et du logiciel, avec les résultats revendiqués, et permettent de relier ces derniers à la configuration spécifique du logiciel définitif.

Tableau F.6 – Intégration de l'électronique programmable (matériel et logiciel)

(voir 7.5 et le Tableau C.6 de la CEI 61508-3)

	Propriété	Définition
6.1	Complétude de l'intégration par rapport aux spécifications de conception	L'intégration fournit la profondeur et la couverture appropriées des composants du système afin de démontrer que ce dernier peut exécuter les fonctions prévues et n'exécute aucune fonction non prévue dans toutes les conditions d'exploitation prévisibles et en cas de défaillance. Ceci couvre les principes appliqués pour la vérification, les niveaux de conception visés et les aspects de l'intégration (par exemple vérification de la complétude de l'interaction entre les modules)
6.2	Exactitude de l'intégration par rapport aux spécifications de conception (réalisation satisfaisante)	L'intégration est basée sur des hypothèses correctes. Par exemple, exactitude des résultats attendus, des conditions d'utilisation considérées, représentativité des environnements d'essai. La tâche d'intégration est achevée, et il existe des preuves spécifiques attestant que les exigences de sécurité ont été satisfaites.
6.3	Répétabilité	La répétition des évaluations individuelles effectuées comme partie intégrante de l'intégration permet d'obtenir des résultats cohérents.
6.4	Configuration d'intégration définie avec précision	L'intégration garantit qu'elle a été appliquée effectivement comme documentée, selon la version appropriée des composants et du logiciel, avec les résultats revendiqués, et permet de relier ces derniers à la configuration spécifique du logiciel définitif.

Tableau F.7 – Aspects de la validation de sécurité du logiciel

(voir 7.7 et le Tableau C.7 de la CEI 61508-3)

	Propriété	Définition
7.1	Complétude de la validation par rapport à la spécification de conception du logiciel	La validation du logiciel traite de toutes les exigences de la spécification de conception du logiciel.
7.2	Exactitude de la validation par rapport à la spécification de conception du logiciel (réalisation satisfaisante)	La tâche de validation du logiciel est achevée, et il existe des preuves spécifiques attestant que les exigences de sécurité ont été satisfaites.
7.3	Répétabilité	La répétition des évaluations individuelles effectuées comme partie intégrante de la validation du logiciel permet d'obtenir des résultats cohérents.
7.4	Configuration de validation définie avec précision	Définition claire et concise: du système; des exigences; de l'environnement.

Tableau F.8 – Modification du logiciel

(voir 7.8 et le Tableau C.8 de la CEI 61508-3)

	Propriété	Définition
8.1	Complétude de la modification par rapport à ses exigences	La modification a été dûment approuvée par le personnel autorisé, avec une compréhension adaptée de ses conséquences fonctionnelles, de sécurité, techniques et opérationnelles.
8.2	Exactitude de la modification par rapport à ses exigences	La modification atteint ses objectifs spécifiés.
8.3	Insensibilité à l'introduction d'anomalies de conception intrinsèques	La modification n'introduit aucune nouvelle anomalie systématique. Exemples: division par zéro, indices ou pointeurs en dehors des limites, utilisation de variables non initialisées.
8.4	Évitement de comportement indésirable	La modification n'introduit aucun comportement qui, selon les contraintes énoncées dans la spécification des exigences pour la sécurité du logiciel, doit être évité.
8.5	Conception vérifiable et pouvant être soumise à essai	La conception du logiciel est telle que l'effet de la modification peut être examiné en profondeur.
8.6	Essais de régression et couverture de vérification	La conception du logiciel est telle que des essais de régression efficaces et approfondis peuvent être effectués pour démontrer que le logiciel, après modification, continue de satisfaire à la spécification des exigences pour la sécurité du logiciel.

Tableau F.9 – Vérification du logiciel

(voir 7.9 et le Tableau C.9 de la CEI 61508-3)

	Propriété	Définition
9.1	Complétude de la vérification par rapport à la phase précédente	La vérification est capable d'établir que le logiciel satisfait à toutes les exigences pertinentes de la spécification des exigences pour la sécurité du logiciel.
9.2	Exactitude de la vérification par rapport à la phase précédente (réalisation satisfaisante)	La tâche de vérification est achevée, et il existe des preuves spécifiques attestant que les exigences de sécurité ont été satisfaites.
9.3	Répétabilité	La répétition des évaluations individuelles effectuées comme partie intégrante de la vérification permet d'obtenir des résultats cohérents.
9.4	Configuration de vérification définie avec précision	La vérification a été appliquée à la version appropriée des composants et du logiciel, avec les résultats revendiqués, et permet de relier ces derniers à la configuration spécifique du logiciel définitif.

Tableau F.10 – Evaluation de la sécurité fonctionnelle

(voir l'Article 8 et le Tableau C.10 de la CEI 61508-3)

	Propriété	Définition
10.1	Complétude de l'évaluation de la sécurité fonctionnelle par rapport à la présente norme	L'évaluation de la sécurité fonctionnelle du logiciel fournit des indications claires concernant l'étendue de conformité constatée, les jugements effectués, les mesures correctives et les échéances recommandées, les conclusions établies et les recommandations inhérentes à l'acceptation, l'acceptation qualifiée ou le rejet, ainsi qu'aux contraintes de temps qui y sont associées.
10.2	Exactitude de l'évaluation de la sécurité fonctionnelle du logiciel par rapport aux spécifications de conception (réalisation satisfaisante)	La tâche d'évaluation de la sécurité fonctionnelle du logiciel est achevée, et il existe des preuves spécifiques attestant que les exigences de sécurité ont été satisfaites.
10.3	Résolution traçable de tous les problèmes identifiés	Indication claire de l'étendue de traitement effectif des problèmes dus à l'évaluation de la sécurité fonctionnelle du logiciel.
10.4	Aptitude à modifier l'évaluation de la sécurité fonctionnelle après modification sans remaniement étendu de l'évaluation	L'évaluation de la sécurité fonctionnelle du logiciel peut être remaniée pour permettre de réévaluer les éléments constitutifs de l'évaluation de la sécurité fonctionnelle du logiciel après modification de ce dernier, et pour parvenir à des conclusions révisées, sans qu'il soit nécessaire de procéder à un remaniement extensif de l'évaluation complète de la sécurité fonctionnelle du logiciel.
10.5	Répétabilité	L'évaluation de la sécurité fonctionnelle est effectuée par rapport à un processus cohérent, planifié et ouvert portant sur des individus et des documents identifiés, qui peut faire l'objet d'un examen détaillé, consacré à la base d'établissement des évaluations et des jugements, par toutes les personnes concernées par les évaluations dudit processus, y compris les fournisseurs, utilisateurs, spécialistes de maintenance et organes régulateurs du ou des systèmes. L'évaluation de la sécurité fonctionnelle permet à du personnel compétent indépendant de répéter les évaluations individuelles effectuées comme partie intégrante de l'évaluation.
10.6	Rapidité d'exécution	L'évaluation de la sécurité fonctionnelle est effectuée selon une fréquence appropriée liée aux phases du cycle de vie de sécurité du logiciel, et au moins préalablement à l'existence avérée de dangers déterminés. Elle permet également une rapidité de signalement des manquements constatés. Les résultats des essais, inspections, analyses etc. sont effectivement disponibles lorsqu'ils doivent être utilisés comme élément d'information dans le cadre d'une décision d'évaluation.
10.7	Configuration définie avec précision	L'évaluation de la sécurité fonctionnelle du logiciel permet de relier les résultats à la configuration spécifique du système, qui doit être justifiée par les résultats de ladite évaluation.

Annexe G
(informative)

Indications pour le développement d'un logiciel orienté objets relatif à la sécurité

Toutes les recommandations de la présente norme relatives à la conception du logiciel s'appliquent aux logiciels orientés objets. Dans la mesure où l'approche orientée objets présente les informations différemment des approches procédurales, la liste suivante contient les recommandations devant faire l'objet d'une considération spécifique:

– compréhension des hiérarchies de classes et identification de la (des) fonction(s) du logiciel qui sera (seront) exécutée(s) sur l'appel d'une méthode donnée (y compris lors de l'utilisation d'une bibliothèque de classes existante);
– essais basés sur la structure (Tableau B.2 de la CEI 61508-3 et C.5.8 de la CEI 61508-7).

Les Tableaux G.1 et G.2 donnent des recommandations informatives sur l'utilisation de logiciel orienté objets venant en complément des recommandations normatives plus générales données dans les Tableaux A.2 et A.4 de la CEI 61508-3.

Tableau G.1 – Architecture du logiciel orienté objets

	Recommandation	Détails	SIL1	SIL2	SIL3	SIL4
G1.1	Traçabilité du concept du domaine d'application selon les classes d'architecture.	Note 1	R	HR	HR	HR
G1.2	Utilisation de trames adaptées, combinaisons de classes et de modèles de conception couramment utilisées. NOTE Lors de l'utilisation des trames et modèles de conception existants, les exigences relatives au logiciel prédéveloppé s'appliquent à ces trames et modèles.	Note 2	R	HR	HR	HR

NOTE 1 La traçabilité du domaine d'application selon l'architecture de classes est moins importante.

NOTE 2 EXEMPLE 1 Pour une partie du projet relatif à la sécurité prévu, un projet non relatif à la sécurité peut comporter une trame qui a permis d'accomplir avec succès une tâche similaire et que les participants au projet connaissent bien. L'utilisation de cette trame est alors recommandée.

EXEMPLE 2 Il peut se révéler nécessaire d'utiliser des algorithmes différents pour exécuter des sous-tâches étroitement liées du projet relatif à la sécurité. Le modèle de stratégie peut être choisi pour accéder aux algorithmes.

EXEMPLE 3 Une partie du projet relatif à la sécurité peut consister à adresser des avertissements appropriés aux parties prenantes internes et externes. Le modèle d'observation peut être choisi pour organiser ces avertissements. L'exigence ne s'applique pas aux bibliothèques.

NOTE 3 Il s'agit généralement d'une classe de base abstraite qui permet d'accéder aux classes concrètes dérivées.

Tableau G.2 – Conception détaillée orientée objets

	Recommandation	SIL1	SIL2	SIL3	SIL4
G2.1	Il convient qu'un seul objectif soit affecté aux classes.	R	R	HR	HR
G2.2	Héritage utilisé uniquement si la classe dérivée constitue un affinement de sa classe de base.	HR	HR	HR	HR
G2.3	Importance de l'héritage limitée par des règles de codage.	R	R	HR	HR
G2.4	Annulation d'opérations (méthodes) soumise à un contrôle strict.	R	HR	HR	HR
G2.5	Héritage multiple utilisé uniquement pour les classes d'interface.	HR	HR	HR	HR
G2.6	Héritage de classes inconnues.			NR	NR
G2.7	Vérification que les bibliothèques OO réutilisées satisfont aux recommandations du présent tableau.	HR	HR	HR	HR

NOTE 1 En d'autres termes: Une classe est caractérisée par l'affectation d'une responsabilité, c'est-à-dire veiller aux données qui y sont étroitement liées et aux opérations sur ces données.

NOTE 2 Une attention particulière est requise pour éviter les dépendances circulaires entre objets.

Les termes suivants utilisés ci-dessus sont définis de manière informelle comme suit.

Tableau G.3 – Quelques termes détaillés orientés objets

Terme	Définition informelle
Classe de base	Classe comportant des classes dérivées. Une classe de base est parfois qualifiée de classe supérieure ou classe parent.
Classe dérivée	Classe (ensemble d'attributs et d'opérations) qui hérite d'attributs et/ou opérations d'une autre classe (classe de base). Une classe dérivée est parfois qualifiée de sous-classe ou classe enfant.
Trame	Structure d'un programme, prédéveloppée dans de nombreux cas afin d'être remplie pour l'application spécifique.
Annulation	Substitution d'une opération (méthode, sous-programme) par une autre (méthode, sous-programme) de la même signature et de la même hiérarchie d'héritage en cours d'exécution; propriété des langages ou des programmes orientés objets; application de polymorphisme.
Signature d'une opération	Nom d'une opération (sous-programme, méthode), accompagné de ses paramètres (arguments) et de ses types, parfois également ses types de retour. Deux signatures sont égales si elles ont les mêmes noms, nombre et types de paramètres; dans certains langages, les types de retour doivent également être égaux.

Bibliographie

[1] CEI 60068-1:1988, *Essais d'environnement – Partie 1: Généralités et guide*

[2] CEI 60529:1989, *Degrés de protection procurés par les enveloppes (Code IP)*

[3] CEI 60812:2006, *Techniques d'analyse de la fiabilité du système – Procédure d'analyse des modes de défaillance et de leurs effets (AMDE)*

[4] CEI 60880:2006, *Centrales nucléaires de puissance – Instrumentation et contrôle-commande importants pour la sûreté – Aspects logiciels des systèmes programmés réalisant des fonctions de catégorie A*

[5] CEI 61000-4-1:2006, *Compatibilité électromagnétique (CEM) – Partie 4-1: Techniques d'essai et de mesure – Vue d'ensemble de la série CEI 61000-4*

[6] CEI 61000-4-5:2005, *Compatibilité électromagnétique (CEM) – Partie 4-5: Techniques d'essai et de mesure – Essai d'immunité aux ondes de choc*

[7] CEI/TR 61000-5-2:1997, *Compatibilité électromagnétique (CEM) – Partie 5: Guides d'installation et d'atténuation – Section 2: Mise à la terre et câblage*

[8] CEI 61025:2006, *Analyse par arbre de panne (AAP)*

[9] CEI 61069-5:1994, *Mesure et commande dans les processus industriels – Appréciation des propriétés d'un système en vue de son évaluation – Partie 5: Evaluation de la sûreté de fonctionnement d'un système*

[10] CEI 61078:2006, *Techniques d'analyse pour la sûreté de fonctionnement – Bloc-diagramme de fiabilité et méthodes booléennes*

[11] CEI 61131-3:2003, *Programmable controllers – Part 3: Programming languages* (disponible en anglais seulement)

[12] CEI 61160:2005, *Revue de conception*

[13] CEI 61163-1:2006, *Déverminage sous contraintes – Partie 1: Assemblages réparables fabriqués en lots*

[14] CEI 61164:2004, *Reliability growth – Statistical test and estimation methods* (disponible en anglais seulement)

[15] CEI 61165:2006, *Application des techniques de Markov*

[16] CEI 61326-3-1:2008, *Matériel électrique de mesure, de commande et de laboratoire – Exigences relatives à la CEM – Partie 3-1: Exigences d'immunité pour les systèmes relatifs à la sécurité et pour les matériels destinés à réaliser des fonctions relatives à la sécurité (sécurité fonctionnelle) – Applications industrielles générales*

[17] CEI 61326-3-2:2008, *Matériel électrique de mesure, de commande et de laboratoire – Exigences relatives à la CEM – Partie 3-2: Exigences d'immunité pour les systèmes relatifs à la sécurité et pour les matériels destinés à réaliser des fonctions relatives à la sécurité (sécurité fonctionnelle) – Applications industrielles dont l'environnement électromagnétique est spécifié*

[18] CEI 81346-1:1996, *Systèmes industriels, installations et appareils, et produits industriels – Principes de structuration et désignations de référence – Partie 1: Règles de base*

[19] CEI 61506:1997, *Mesure et commande dans les processus industriels – Documentation des logiciels d'application*

[20] CEI/TR 61508-0:2005, *Sécurité fonctionnelle des systèmes électriques / électroniques / électroniques programmables relatifs à la sécurité – Partie 0: La sécurité fonctionnelle et la CEI 61508*

[21] CEI 61511 (toutes les parties), *Sécurité fonctionnelle – Systèmes instrumentés de sécurité pour le secteur des industries de transformation*

[22] CEI 62061:2005, *Sécurité des machines – Sécurité fonctionnelle des systèmes de commande électriques, électroniques et électroniques programmables relatifs à la sécurité*

[23] CEI 62308:2006, *Fiabilité de l'équipement – Méthodes d'évaluation de la fiabilité*

[24] ISO/CEI 1539-1:2004, *Technologies de l'information – Langages de programmation – Fortran – Partie 1: Langage de base*

[25] ISO 5807:1985, *Traitement de l'information – Symboles de documentation et conventions applicables aux données, aux organigrammes de programmation et d'analyse, aux schémas des réseaux de programmes et des ressources de système*

[26] ISO/CEI 7185:1990, *Technologies de l'information – Langages de programmation – Pascal*

[27] ISO/CEI 8631:1989, *Technologies de l'information – Structures de programmes et normes pour leur représentation*

[28] ISO/CEI 8652:1995, *Technologies de l'information – Langages de programmation – Ada*

[29] ISO 8807:1989, *Systèmes de traitement de l'information – Interconnexion de systèmes ouverts – LOTOS – Technique de description formelle basée sur l'organisation temporelle de comportement observationnel*

[30] ISO/CEI 9899:1999, *Langages de programmation – C*

[31] ISO/CEI 10206:1991, *Technologies de l'information – Langages de programmation – Pascal étendu*

[32] ISO/CEI 10514-1:1996, *Technologies de l'information – Langages de programmation – Partie 1: Modula-2, langage de base*

[33] ISO/CEI 10514-3:1998, *Technologies de l'information – Langages de programmation – Partie 3: Modula 2 orienté objet*

[34] ISO/CEI 13817-1:1996, *Technologies de l'information – Langages de programmation, leurs environnements et interfaces logiciel système – Méthode de développement de Vienne – Langage de spécification – Partie 1: Langage de base*

[35] ISO/CEI 14882:2003, *Langages de programmation – C++*

[36] ISO/CEI/TR 15942:2000, *Technologies de l'information – Langages de programmation – Guide pour l'emploi du langage de programmation Ada dans les systèmes de haute intégrité*

[37] CEI 61800-5-2, *Compatibilité électromagnétique (CEM) – Partie 5: Guides d'installation et d'atténuation – Section 2: Mise à la terre et câblage*

[38] CEI 60601 (toutes les parties), *Appareils électromédicaux*

[39] CEI 60068-2-1, *Essais d'environnement – Partie 2-1: Essais – Essai A: Froid*

[40] CEI 60068-2-2, *Essais d'environnement – Partie 2-2: Essais – Essai B: Chaleur sèche*

[41] ISO 9000, *Systèmes de management de la qualité – Principes essentiels et vocabulaire*

[42] CEI 61508-1:2010, *Sécurité fonctionnelle des systèmes électriques/électroniques /électroniques programmables relatifs à la sécurité – Partie 1: Exigences générales*

[43] CEI 61508-2:2010, *Sécurité fonctionnelle des systèmes électriques/électroniques /électroniques programmables relatifs à la sécurité – Partie 2: Exigences concernant les systèmes électriques/ électroniques/électroniques programmables relatifs à la sécurité*

[44] CEI 61508-3:2010, *Sécurité fonctionnelle des systèmes électriques/électroniques/ électroniques programmables relatifs à la sécurité – Partie 3: Exigences concernant les logiciels*

[45] CEI 61508-6:2010, *Sécurité fonctionnelle des systèmes électriques / électroniques / électroniques programmables relatifs à la sécurité – Partie 6: Lignes directrices pour l'application de la CEI 61508-2 et de la CEI 61508-3*

Index

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch