Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

BS EN 60880:2009

BSI Standards Publication

Nuclear power plants — Instrumentation and control systems important to safety — Software aspects for computer-based systems performing category A functions

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW

raising standards worldwide™

BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

National foreword

This British Standard is the UK implementation of EN 60880:2009. It is identical to IEC 60880:2006. It supersedes BS IEC 60880:2006 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee NCE/8, Reactor instrumentation.

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

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© BSI 2010

ISBN 978 0 580 63962 3

ICS 27.120.20; 35.080

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

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 January 2010

Amendments issued since publication

Amd. No.	Date	Text affected

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 60880

October 2009

ICS 27.120.20

English version

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

Kernkraftwerke -
Leittechnik für Systeme
mit sicherheitstechnischer Bedeutung -
Softwareaspekte für rechnerbasierte
Systeme zur Realisierung
von Funktionen der Kategorie A
(IEC 60880:2006)

This European Standard was approved by CENELEC on 2009-07-01. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

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

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the Central Secretariat has the same status as the official versions.

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

CENELEC

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

Central Secretariat: Avenue Marnix 17, B - 1000 Brussels

© 2009 CENELEC - All rights of exploitation in any form and by any means reserved worldwide for CENELEC members.

Ref. No. EN 60880:2009 E

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Foreword

The text of the International Standard IEC 60880:2006, prepared by SC 45A, Instrumentation and control of nuclear facilities, of IEC TC 45, Nuclear instrumentation, was submitted to the formal vote and was approved by CENELEC as EN 60880 on 2009-07-01 without any modification.

The following dates were fixed:

– latest date by which the EN has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2010-07-01
– latest date by which the national standards conflicting with the EN have to be withdrawn	(dow)	2012-07-01

CLC/TC 45AX experts draw attention to the readers of this European standard to the fact that it should be read in conjunction with IAEA document INSAG-10, 1996, “Defence in Depth in Nuclear Safety” which applies.

Endorsement notice

The text of the International Standard IEC 60880:2006 was approved by CENELEC as a European Standard without any modification.

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

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.

NOTE When an international publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

Publication	Year	Title	EN/HD	Year
IEC 60671	- [1]	Nuclear power plants - Instrumentation and control systems important to safety - Surveillance testing	-	-
IEC 61069-2	1993	Industrial-process measurement and control - Evaluation of system properties for the purpose of system assessment - Part 2: Assessment methodology	EN 61069-2	1994
IEC 61226	- [1]	Nuclear power plants - Instrumentation and control systems important to safety - Classification of instrumentation and control functions	-	-
IEC 61508-4	- [1]	Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 4: Definitions and abbreviations	EN 61508-4	2001 [2]
IEC 61513	- [1]	Nuclear power plants - Instrumentation and control for systems important to safety - General requirements for systems	-	-
ISO/IEC 9126	Series	Software engineering - Product quality	-	-
IAEA guide NS-G-1.2	- [1]	Safety assessment and verification for nuclear power plants	-	
IAEA guide NS-G-1.3	- [1]	Instrumentation and control systems important to safety in nuclear power plants	-	-

[1] Undated reference.

[2] Valid edition at date of issue.

CONTENTS

INTRODUCTION 11

1 Scope and object 17
2 Normative references 17
3 Terms and definitions 19
4 Symbols and abbreviations 29
5 General requirements for software projects 29
5.1 General 29
5.2 Software types 33
5.3 Software development approach 35
5.4 Software project management 39
5.5 Software quality assurance plan 39
5.6 Configuration management 41
5.7 Software security 43
6 Software requirements 47
6.1 Specification of software requirements 47
6.2 Self-supervision 49
6.3 Periodic testing 49
6.4 Documentation 51
7 Design and implementation 51
7.1 Principles for design and implementation 53
7.2 Language and associated translators and tools 57
7.3 Detailed recommendations 59
7.4 Documentation 63
8 Software Verification 63
8.1 Software verification process 63
8.2 Software verification activities 65
9 Software aspects of system integration 73
9.1 Software aspects of system integration plan 75
9.2 System integration 77
9.3 Integrated system verification 77
9.4 Fault resolution procedures 79
9.5 Software aspects of integrated system verification report 79
10 Software aspects of system validation 81
10.1 Software aspects of the system validation plan 81
10.2 System validation 81
10.3 Software aspects of the system validation report 83
10.4 Fault resolution procedures 83
11 Software modification 83
11.1 Modification request procedure 85
11.2 Procedure for executing a software modification 87
11.3 Software modification after delivery 89

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

12 Software aspects of installation and operation 91
12.1 On-site installation of the software 91
12.2 On-site software security 91
12.3 Adaptation of the software to on-site conditions 93
12.4 Operator training 93
13 Defences against common cause failure due to software 95
13.1 General 95
13.2 Design of software against CCF 97
13.3 Sources and effects of CCF due to software 97
13.4 Implementation of diversity 99
13.5 Balance of drawbacks and benefits connected with the use of diversity 99
14 Software tools for the development of software 99
14.1 Introduction 99
14.2 Selection of tools 101
14.3 Requirements for tools 103
15 Qualification of pre-developed software 113
15.1 General 113
15.2 General requirements 113
15.3 Evaluation and assessment process 115
15.4 Requirements for integration in the system and modification of PDS 131

Annex A (normative) Software safety life cycle and details of software requirements 133
Annex B (normative) Detailed requirements and recommendations for design and implementation 137
Annex C (informative) Example of application oriented software engineering (software development with application-oriented language) 163
Annex D (informative) Language, translator, linkage editor 171
Annex E (informative) Software verification and testing 175
Annex F (informative) Typical list of software documentation 191
Annex G (informative) Considerations of CCF and diversity 193
Annex H (informative) Tools for production and checking of specification, design and implementation 201
Annex I (informative) Requirements concerning pre-developed software (PDS) 207
Annex J (informative) Correspondence between IEC 61513 and this standard 211

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

INTRODUCTION

a) Technical background, main issues and organisation of the standard

Engineering of software based Instrumentation and Control (I&C) systems to be used for nuclear safety purposes is a challenge due to the safety requirements to be fulfilled. The safety software used in nuclear power plants (NPP) which are often required only in emergency cases, have to be fully validated and qualified before being used in operation. In order to achieve the high reliability required, special care has to be taken throughout the entire life cycle, from the basic requirements, the various design phases and V&V procedures for operation and maintenance. It is the main aim of this standard to address the related safety aspects and to provide requirements for achieving the high software quality necessary.

The first edition of this standard was issued in 1986 to interpret the basic safety principles applied so far in hardwired systems for the utilisation of digital systems — multiprocessor distributed systems as well as larger scale central processor systems — in the safety systems of nuclear power plants.

It has been used extensively within the nuclear industry to provide requirements and guidance for software of NPP safety I&C systems.

Although many of the requirements within the first edition continued to be relevant, there were significant factors which justified the development of this second edition:

– Since 1986, a number of new standards have been produced which address in detail the general requirements for systems (IEC 61513), hardware requirements (IEC 60987) and a standard to address software for I&C systems performing category B or C functions for NPP systems important to safety (IEC 62138). The Safety Guide 50-SG-D3 of the IAEA has been superseded by the guide NS-G-1.3. Additionally, IEC 60880-2 has been issued.
– Software engineering techniques have advanced significantly in the intervening years.

In this standard, utmost care has been taken to keep transparency with respect to the first edition. Where possible, the phrasing of requirements has been kept, otherwise it has been extended in a traceable way. In the same manner, IEC 60880-2 dealing with software aspects of defence against common cause failures, use of software tools and pre-developed software has been integrated, so that now this current standard covers entirely the software safety issues to be addressed.

It is intended that the standard be used by systems developers, systems purchasers/users (utilities), systems assessors and by licensors.

b) Situation of the current standard in the structure of the SC 45A standard series

IEC 60880 is directly referenced by IEC 61513 which deals with the system aspects of high integrity computer-based I&C used in safety systems of nuclear power plants together.

IEC 60880 is the second level SC 45A document tackling the issue of software aspects for I&C systems performing category A functions.

标准分享网 www.bzfxw.com 免费下载

Software for categories B and C functions is dealt with in IEC 62138.

IEC 60880 and IEC 62138 together cover the domain of the software aspects of computer-based systems used in nuclear power plants to perform functions important to safety.

This second edition of IEC 60880 is to be read in conjunction with IEC 60987 and IEC 61226, the appropriate SC 45A standards on computer hardware and on classification.

For more details on the structure of the SC 45A standard series see item d) of this introduction.

c) Recommendation and limitation regarding the application of this standard

It is important to note that this standard establishes no additional functional requirements for safety systems.

Aspects for which special requirements and recommendations have been produced, are:

1) a general approach to software development to assure the production of the highly reliable software required including hardware and software interdependencies;
2) a general approach to software verification and to the software aspects of the computer-based system validation;
3) procedures for software modification and configuration control;
4) requirements for use of tools;
5) procedures for qualification of pre-developed software.

It is recognised that software technology is continuing to develop at a rapid pace and that it is not possible for a standard such as this to include references to all modern design technologies and techniques.

To ensure that the standard will continue to be relevant in future years the emphasis has been placed on issues of principle, rather than specific software technologies.

If new techniques are developed then it should be possible to assess the suitability of such techniques by applying the safety principles contained within this standard.

d) Description of the structure of the SC 45A standard series and relationships with other IEC documents and other bodies documents (IAEA, ISO)

The top level document of the SC 45A standard series is IEC 61513. This standard deals with requirements for NPP I&C systems important to safety and lays out the SC 45A standards series.

IEC 61513 refers directly to other SC 45A standards for general topics related to categorization of functions and classification of systems, qualification, separation of systems, defence against common cause failure, software aspects of computer-based systems, hardware aspects of computer-based systems, and control room design. The standards referenced directly at this second level should be considered together with IEC 61513 as a consistent document set.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

At a third level, SC 45A standards not directly referenced by IEC 61513 are standards related to specific equipment, technical methods or specific activities. Usually these documents, which make reference to second level documents for general topics, can be used on their own.

A fourth level extending the SC 45A standard series corresponds to the technical reports which are not normative.

IEC 61513 has adopted a presentation format similar to the basic safety publication IEC 61508 with an overall safety life-cycle framework and a system life-cycle framework and provides an interpretation of the general requirements of IEC 61508 parts 1, 2 and 4, for the nuclear application sector. Compliance with this standard will facilitate consistency with the requirements of IEC 61508 as they have been interpreted for the nuclear industry. In this framework, IEC 60880 and IEC 62138 correspond to IEC 61508, part 3 for the nuclear application sector.

IEC 61513 refers to ISO standards as well as to IAEA 50-C-QA for topics related to quality assurance.

The SC 45A standards series consistently implement and detail the principles and basic safety aspects provided in the IAEA Code on the safety of nuclear power plants and in the IAEA safety series, in particular the Requirements NS-R-1, “Safety of Nuclear Power Plants: Design” and the Safety Guide NS-G-1.3, “Instrumentation and control systems important to safety in Nuclear Power Plants”. The terminology and definitions used by SC 45A standards are consistent with those used by the IAEA.

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

NUCLEAR POWER PLANTS – INSTRUMENTATION AND CONTROL SYSTEMS IMPORTANT TO SAFETY – SOFTWARE ASPECTS FOR COMPUTER-BASED SYSTEMS PERFORMING CATEGORY A FUNCTIONS

1 Scope and object

This International Standard provides requirements for the software of computer-based I&C systems of nuclear power plants performing functions of safety category A as defined by IEC 61226.

According to the definition in IEC 61513, I&C systems of safety class 1 are basically intended to support category A functions, but may also support functions of lower categories. However the system requirements are always determined by the functions of the highest category implemented.

For software of I&C system performing only category B and C functions in NPP as defined by IEC 61226, requirements and guidance of IEC 62138 are applicable.

This standard provides requirements for the purpose of achieving highly reliable software. It addresses each stage of software generation and documentation, including requirements specification, design, implementation, verification, validation and operation.

The principles applied in developing these requirements include:

– best available practices;
– top-down design methods;
– modularity;
– verification of each phase;
– clear documentation;
– auditable documents;
– validation testing.

Additional guidance and information on how to comply with the requirements of the main part of this standard is given in Annexes A to I.

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 60671, *Periodic tests and monitoring of the protection system of nuclear reactors*

IEC 61069-2:1993, *Industrial-process measurement and control – Evaluation of system properties for the purpose of system assessment – Part 2: Assessment methodology*

IEC 61226, *Nuclear power plants – Instrumentation and control systems important for safety – Classification of instrumentation and control functions*

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

IEC 61513, *Nuclear power plants – Instrumentation and control for systems important to safety – General requirements for systems*

ISO/IEC 9126, *Software engineering – Product quality*

IAEA guide NS-G-1.2, *Safety Assessment and Verification for Nuclear power Plant*

IAEA guide NS-G-1.3, *Instrumentation and Control Systems Important to Safety in Nuclear Power Plants*

3 Terms and definitions

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

3.1
animation
process by which the behaviour defined by a specification is displayed with actual values derived from the stated behaviour expressions and from some input values

3.2
application function
function of an I&C system that performs a task related to the process being controlled rather than to the functioning of the system itself

[IEC 61513, 3.1]

3.3
application-oriented language
computer language specifically designed to address a certain type of application and to be used by persons who are specialists of this type of application

[IEC 62138, 3.3]

NOTE 1 Equipment families usually feature application-oriented languages so as to provide easy to use capability for adjusting the equipment to specific requirements.

NOTE 2 Application-oriented languages may be used to specify the functional requirements of an I&C system, and/or to specify or design application software. They may be based on texts, on graphics, or on both.

NOTE 3 Examples: function block diagram languages, language defined by IEC 61131-3.

3.4
application software
part of the software of an I&C system that implements the application functions

[IEC 61513, 3.2]

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

3.5
automated code generation
function of automated tools allowing transformation of the application-oriented language into a form suitable for compilation or execution

3.6
channel
arrangement of interconnected components within a system that initiates a single output. A channel loses its identity where single output signals are combined with signals from other channels, e.g., from a monitoring channel, or a safety actuation channel.

[IAEA NS-G-1.3, Glossary]

3.7
code compaction
purposeful reduction in memory size required for a program by the elimination of redundant or extraneous instructions

3.8
common cause failure (CCF)
failure of two or more structures, systems or components due to a single specific event or cause

[IAEA NS-G-1.3, Glossary]

3.9
computer
programmable functional unit that consists of one or more associated processing units and peripheral equipment, that is controlled by internally stored programs and that can perform substantial computation, including numerous arithmetic operations or logic operations, without human intervention during a run

[ISO 2382/1]

NOTE A computer may be a standalone unit or may consist of several interconnected units.

3.10
computer program
set of ordered instructions and data that specify operations in a form suitable for execution by a computer

3.11
computer-based system
I&C system whose functions are mostly dependent on, or completely performed by using microprocessors, programmed electronic equipment or computers

[IEC 61513, 3.10]

NOTE Equivalent to: digital systems, software-based system, programmed system.

3.12
data
presentation of information or instructions in a manner suitable for communication, interpretation, or processing by computers

[IEEE 610, modified]

NOTE Data which are required to define parameters and to instantiate application and service functions in the system are called “application data”.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

3.13
defence in depth
application of more than one protective measure for a given safety objective, such that the objective is achieved even if one of the protective measures fails

[IAEA Safety Glossary]

3.14
diversity
existence of two or more different ways or means of achieving a specified objective. Diversity is specifically provided as a defence against common cause failure. It may be achieved by providing systems that are physically different from each other or by functional diversity, where similar systems achieve the specified objective in different ways.

3.15
dynamic analysis
process of evaluating a system or component based on its behaviour during execution. In contrast to static analysis

[IEEE 610]

3.16
failure
deviation of the delivered service from the intended one

[IEC 61513, 3.21, modified]

3.17
fault
defect in a hardware, software or system component

[IEC 61513, 3.22]

3.18
fault tolerance
built-in capability of a system to provide continued correct execution in the presence of a limited number of hardware or software faults

3.19
functional diversity
application of the diversity at the functional level (for example, to have trip activation on both pressure and temperature limit)

3.20
general-purpose language
computer language designed to address all types of usage

[IEC 62138, 3.17]

NOTE 1 The operational system software of equipment families is usually implemented using general-purpose languages.

NOTE 2 Examples: Ada, C, Pascal.

3.21
human error
human action that produces an unintended result

3.22
initialise
to set counters, switches, addresses, or contents of storage devices to zero or other starting values at the beginning of, or at prescribed points in, the operation of a computer program

标准分享网 www.bzfxw.com 免费下载

3.23
integration tests
tests performed during the hardware/software integration process prior to computer-based system validation to verify compatibility of the software and the computer hardware

3.24
library
collection of related software elements that are grouped together, but which are individually selected for inclusion in the final software product

3.25
N-version software
set of different programs, known as versions, developed to meet a common requirement and common acceptance test. Concurrent and independent execution of these versions takes place, generally in redundant hardware. Identical inputs in test systems or corresponding inputs in redundant systems are used. A predetermined strategy such as voting is used to decide between conflicting outputs in different versions.

3.26
operational system software
software running on the target processor during operation, such as: input/output drivers and services, interrupt management, scheduler, communication drivers, applications oriented libraries, on-line diagnostic, redundancy and graceful degradation management

3.27
postulated initiating event
PIE
events that lead to anticipated operational occurrences or accident conditions and their consequential failure effects

[IEC 61513, 3.41]

3.28
pre-developed software
PDS
software part that already exists, is available as a commercial or proprietary product, and is being considered for use

[IEC 62138, 3.24, modified]

3.29
redundancy
provision of alternative (identical or diverse) structures, systems or components, so that any one can perform the required function regardless of the state of operation or failure of any other

[IAEA NS-G-1.3, Glossary]

3.30
role-based access control
access control based on rules, defining the permitted access of users to objects (functions, data) on the basis of user groups with an identical role instead of individual users

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

3.31
safety function
specific purpose that must be accomplished for safety

[IAEA NS-R-1, Glossary]

3.32
safety system
system important to safety, provided to ensure the safety shutdown of the reactor or the residual heat removal from the core, or to limit the consequences of anticipated operational occurrences and design basis incidents

[IAEA NS-R-1, Glossary]

3.33
signal trajectory
time histories of all equipment conditions, internal states, input signals, and operator inputs which determine the outputs of a system

3.34
software
programs (i.e. sets of ordered instructions), data, rules and any associated documentation pertaining to the operation of a computer-based I&C system

[IEC 62138, 3.27]

3.35
software development
phase of the software lifecycle that leads to the creation of the software of an I&C system or of a software product. It covers all the activities from software requirements specification to validation and installation on site

[IEC 62138, 3.30]

3.36
software modification
change in an already agreed document (or documents) leading to an alteration of the executable code

NOTE Software modifications may occur either during initial software development (e.g. to remove faults found in later stages of development), or after the software is already in service.

3.37
software safety life cycle
necessary activities involved in the development and operation of the software of an I&C system important to safety occurring during a period of time that starts at a concept phase with the software requirements specification and finishes when the software is withdrawn from use

[IEC 62138, 3.31]

3.38
software version
instance of a software product derived by modification or correction of a preceding software product instance

[IEEE 610, modified]

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

3.39
specification
document that specifies, in a complete, precise, verifiable manner, the requirements, design behaviour or other characteristics of a system or component and, often, the procedures for determining whether these provisions have been satisfied

[IEEE 610]

NOTE There are different types of specifications, for example software requirements specification or design specification.

3.40
static analysis
process of evaluating a system or component based on its form, structure, content or documentation. In contrast to dynamic analysis

3.41
system software
part of the software of an I&C system designed for a specific computer or equipment family to facilitate the development, operation and modification of these items and associated programs

[IEC 62138, 3.33]

3.42
system validation
confirmation by examination and provision of other evidence that a system fulfils in its entirety the requirement specification as intended (functionality, response time, fault tolerance, robustness)

3.43
verification
confirmation by examination and by provision of objective evidence that the results of an activity meet the objectives and requirements defined for this activity

[IEC 62138, 3.35]

4 Symbols and abbreviations

CASE Computer Aided Software Engineering

CCF Common Cause Failure (see 3.8)

I&C Instrumentation and Control

PDS Pre-Developed Software (see 3.28)

PIE Postulated Initiating Event (see 3.27)

QA Quality Assurance

V&V Verification and Validation

5 General requirements for software projects

5.1 General

The process of producing instrumentation and control systems for use in nuclear power plants is set down in IEC 61513 that introduces the concept of the system safety lifecycle as a vehicle by which the development process can be controlled and whose adoption should also result in the evidence necessary to justify the operation of safety systems. The system safety lifecycle described in IEC 61513 includes and places requirements on, but does not dictate, the project arrangements to be used for production of systems (see Figure 1).

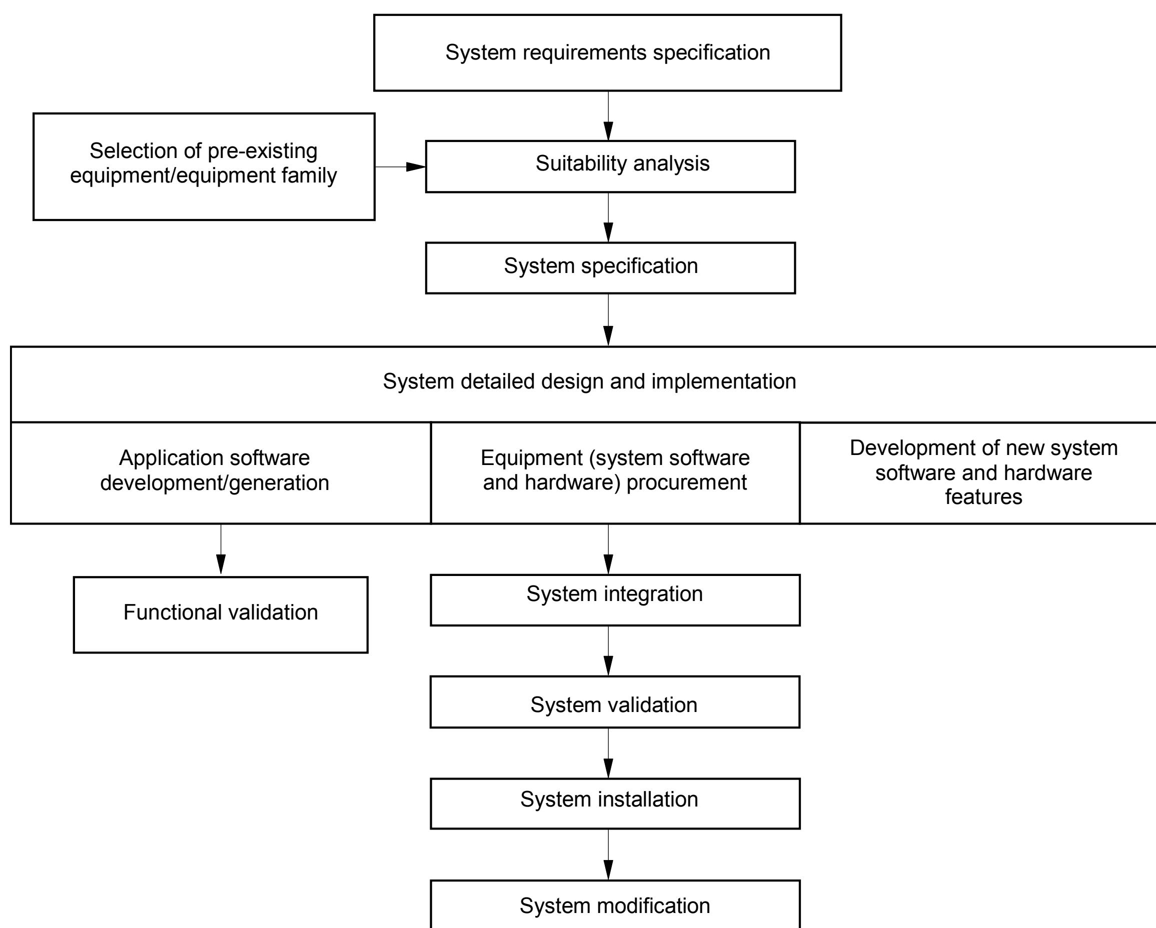


Figure 1 – Activities of the system safety lifecycle (as defined by IEC 61513)

For computer-based (i.e. digital) systems the system safety lifecycle is further developed to introduce the concept of a software safety lifecycle (see Figure 2 for activities). This shows the hardware and software development being undertaken in parallel from a common specification but coming together at the integration and installation phases of the lifecycle.

The following processes support the phased development process of producing the software:

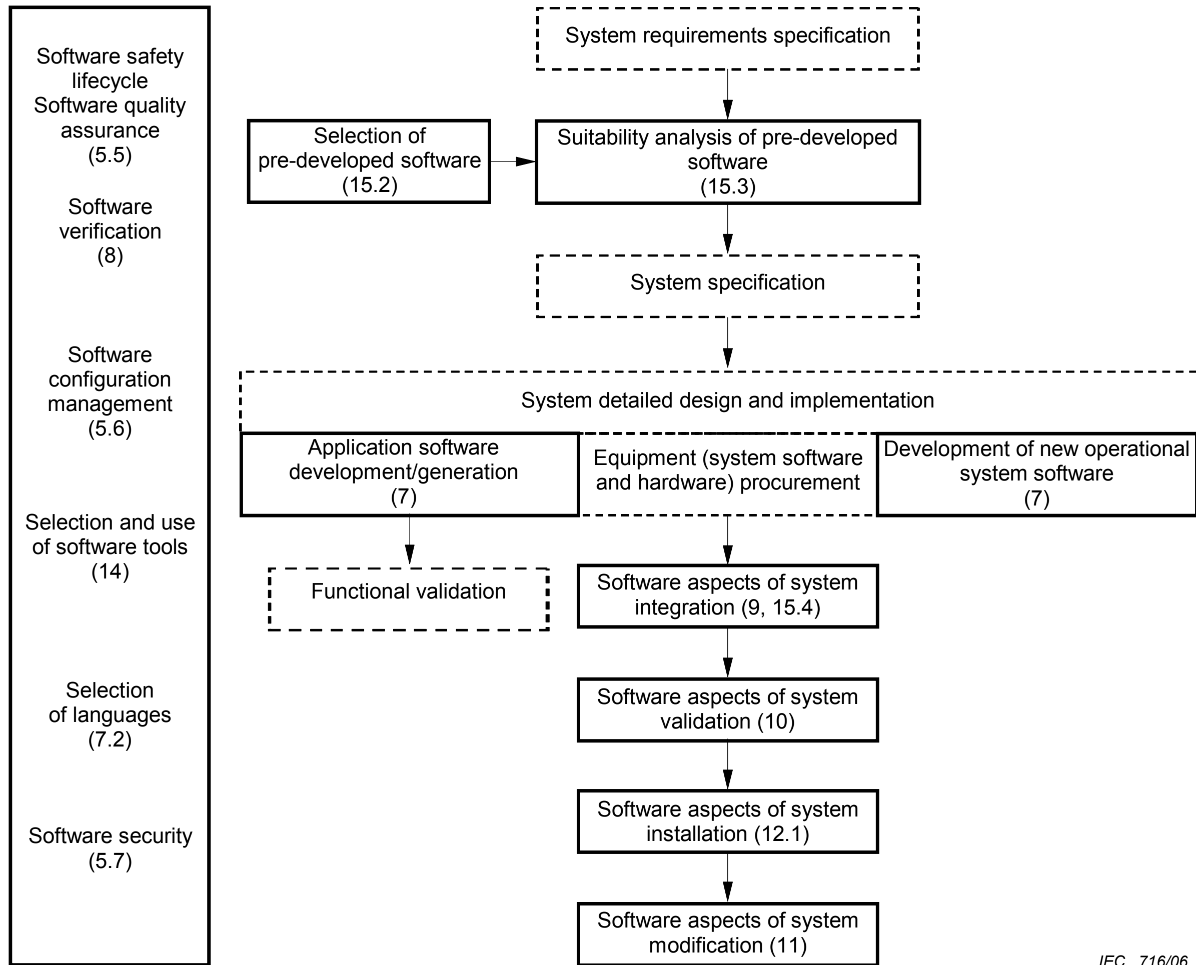
– software project management (5.4);
– software quality assurance and quality control (5.5);
– software configuration management (5.6);
– software security (5.7);
– software verification (Clause 8).

There are also activities involving selection of languages (7.2, Annex D), selection of software tools to support the development (Clause 14), prevention of CCF (Clause 13) and production of documentation (7.4, Annex F).

The resulting software related activities in the system safety lifecycle and the supporting processes are shown below in Figure 2 (boxes in bold lines with reference to related subclauses in brackets).

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载



NOTE Boxes in thin dotted lines represent system activities not addressed in this standard.

Figure 2 – Software related activities in the system safety lifecycle

The approach to software development should be based on the traditional "V" model as this approach has been reflected and promulgated in other standards notably IAEA NS-G-1.3, but allowing necessary adjustments recognizing that some phases of the development can be done automatically by tools and that software development may be iterative.

Subclauses 5.2 and 5.3 introduce the different software types and the development approach considered in this standard.

5.2 Software types

The software components of a system are often defined as being either operational system software (communications, I/O management, standard functions, self-supervision, etc.) or application software (interlock logic, control loops, display formats, alarm logic, etc.).

Application software generally uses the facilities provided by the operational system software, thus reducing the need for duplication of code within modules and thus reducing the overall amount of software.

Application software is usually specific to one project.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Operational system software may be used in different projects.

Many system designs make extensive use of configuration data. Configuration data may be associated with operational system software or with application software. Configuration data associated with application software consists mainly of plant engineering data resulting from the design of the plant, and is often prepared by plant designers who are not required to have software skills.

Configuration data can be divided into:

– data items which are not intended to be modified on-line by plant operators, and which are submitted to the same requirements as apply to the rest of the software;
– parameters, i.e. data items which may be modified by operators during plant operation (for example, alarm limits, set points, data required to calibrate instrumentation) and which need specific requirements.

Many modern I&C equipment platforms are provided with extensive development tools which enable system engineers to design and produce executable codes.

As an example, a typical I&C system, developed using components of an equipment family, includes the following:

– pre-developed software components, such as the operational system software kernel and the application function libraries. Generally, these components have been developed using general-purpose languages;
– configuration data needed to adapt the operational system software kernel to the I/O environment and to the services required by the application;
– application software developed using an application-oriented language.

5.3 Software development approach

Software usually contributes strongly to the functions performed by the I&C system. It may also support additional functions introduced by system design (e.g. initialisation and surveillance of hardware, communication between, and synchronisation of, subsystems). Thus, the software safety lifecycle is in most cases strongly integrated with the system safety lifecycle. In particular, the software requirements specification is a part of, or is derived directly from, system specification and system design.

Though the verification of new software components is definitely a part of the software safety lifecycle, there is often no separate and well-identified boundary between software integration and system integration. Therefore, in this standard, software integration is considered to be a part of system integration. Software validation too is not a purely software activity: in this standard, it is considered a part of system integration and/or system validation.

The standard assumes that the software life cycle, originally intended for the development of software with general-purpose languages, is also applicable to application-oriented languages and configuration of pre-developed software.

However, it recognises differences in the development process by introducing dedicated sub-processes for each software type at the implementation level:

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

– implementation using general-purpose languages;
– implementation using application-oriented languages with associated code generators;
– selection, use and configuration of pre-developed software products.

As boxes "Application software development/generation" and "Development of new operational system software" in Figure 2 represent a large and essential part of the software safety lifecycle, a "zoom" is provided in Figure 3, which illustrates in more detail the activities between software requirements specification and validation, with a clear representation of the three different implementation paths. In that figure, references to related subclauses of this standard are in brackets.

The standard also gives additional requirements for software in Annex B.

The principles, reflected in the requirements of this standard, are related to the quality of the final code, and are applicable regardless of whether the code is developed using general-purpose languages, application-oriented languages with automated code generation, or configuration.

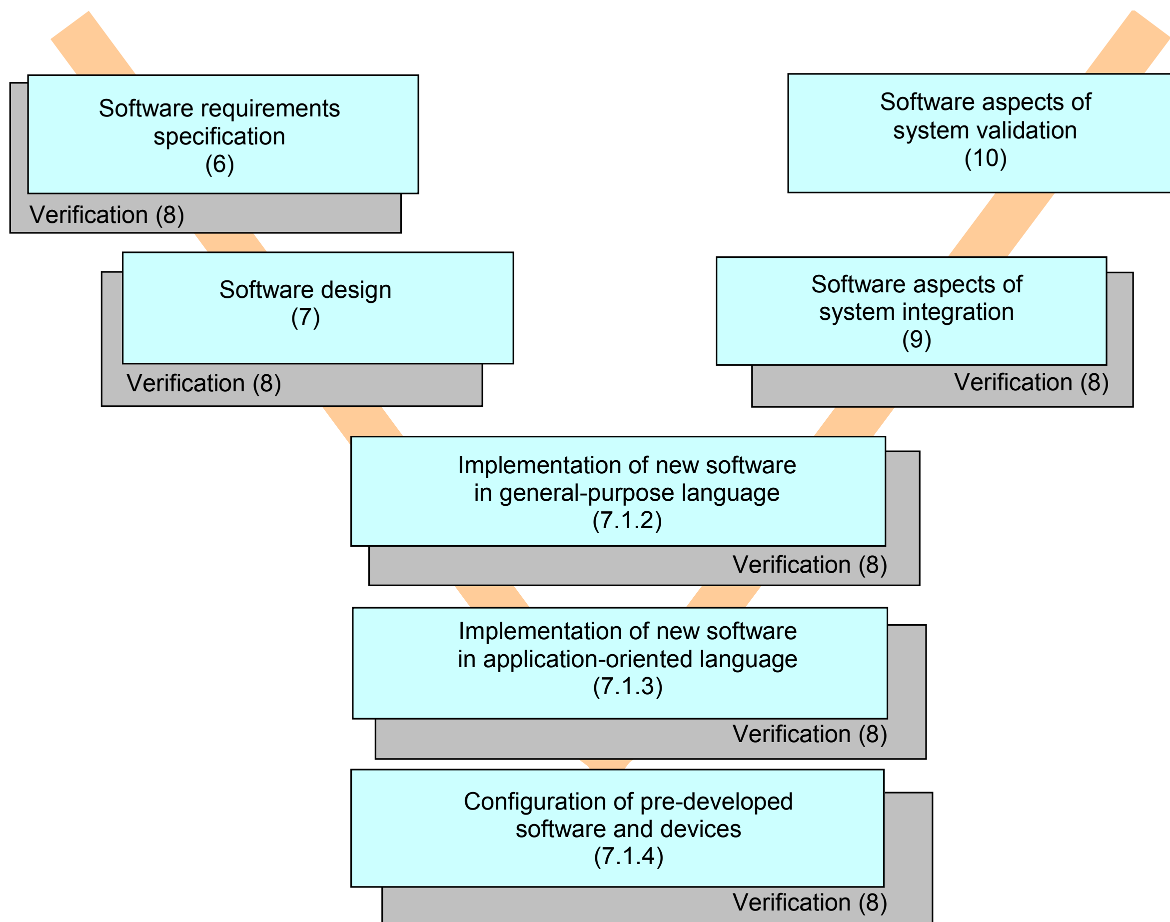


Figure 3 – Development activities of the IEC 60880 software safety lifecycle

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

5.4 Software project management

5.4.1 Any software project shall be structured into a number of phases.

Each phase is to some extent self-contained but will depend on other phases and will, in its turn, be depended on by others. These phases are informally recognizable by the specific activities pertinent to them.

The phases and associated activities defined for a software project describe a process called in this standard the "software development". It is recognized that this process may be iterative provided the requirement from the last paragraph of the introduction to Clause 6 of the IEC 61513 is fulfilled.

The following general factors determine the activities and phases in implementing a software project.

5.4.2 The activities performed during the phases of the development process shall be identified, according to the software development approach chosen for the project (see 5.2 and 5.3).

5.4.3 Software development activities shall address the whole software safety lifecycle.

5.4.4 Each phase of the software development identified in 5.4.1 shall be divided into well-defined activities.

5.4.5 The phases of the software development shall be formalised and none of the identified phases shall be omitted.

5.4.6 When activities of the software development are automated using software tools, the activities automated shall be documented including the documentation of the inputs and outputs relevant to the phase.

5.4.7 The inputs and outputs of each phase shall be defined and documented.

5.4.8 Every output of each phase shall be systematically checked (item c) of B.1 and item g) of B.4).

5.4.9 Each phase shall include generation of the appropriate documents (Annex F).

5.4.10 Each phase shall be systematically terminated by a review including examination of relevant documents.

5.4.11 A list of documents required through the software safety life cycle shall be established during the software development. An example of a typical list is given in Annex F.

5.5 Software quality assurance plan

5.5.1 A quality assurance plan shall exist or be established at an early stage of the software safety life cycle.

Special quality assurance plans may be adopted for individual product phases or particular software components according to national or company standards provided the principles defined in this standard are addressed.

5.5.2 Any deviations from the requirements of this standard and its normative annexes shall be identified and justified.

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

5.5.3 If practices differing from those of normative annexes are used, they shall be documented and auditable according to the requirements of the main parts of this standard.

5.5.4 In particular, the impact of these practices on the I&C system and the software shall be considered.

5.5.5 All technical procedures required during each phase of the software safety life cycle shall be addressed by the quality assurance plan.

5.5.6 The quality assurance plan shall require that the implementation of the activities of the phases is assigned to competent persons equipped with adequate resources.

5.5.7 The quality assurance plan shall require that modifications in already approved documents are identified, reviewed and approved by authorized persons.

5.5.8 The quality assurance plan shall require that the methods, languages, tools, rules and standards used are identified and documented, and known to, and mastered by, the persons concerned.

5.5.9 The quality assurance plan shall require that when several methods, languages, tools, rules and/or standards are used, it is clear which ones were used for each activity.

5.5.10 The quality assurance plan shall require that project specific terms, expressions, abbreviations and conventions are explicitly defined.

5.5.11 The quality assurance plan shall require that any quality issues raised are tracked and resolved.

5.5.12 The quality assurance plan shall require that records resulting from its application are produced.

5.5.13 Every verification step or review shall result in a report on the analysis performed, the conclusions reached and the resolutions agreed. This report shall be included in the documentation.

5.5.14 Any deviation from the quality assurance plan shall be documented and justified.

5.6 Configuration management

Subclause 6.2.1.2 (system configuration management plan) of IEC 61513 provides requirements for configuration management at the I&C system level. This subclause provides additional requirements specific, or of particular importance, to software.

5.6.1 Configuration management for software shall be performed according to the provisions of a configuration management plan or of the quality assurance plan.

5.6.2 These provisions shall be consistent with those for system level configuration management.

5.6.3 Documented software configuration management procedures shall be established early in the lifecycle of a software project to address the following requirements.

5.6.4 Each produced version of each software entity shall be uniquely identified.

5.6.5 It shall be possible to identify the relevant versions of all software documentation associated with each software entity.

5.6.6 Software under development shall be segregated from software which has achieved release or verified status.

5.6.7 It shall be possible to identify the versions of all software entities which together constitute a complete version of the final product.

5.6.8 It shall be possible to verify the integrity of the software entities.

5.6.9 It shall be possible to identify the version of the software in the target system.

5.6.10 It shall be possible to retrospectively identify all software entities affected by the implementation of a modification.

5.6.11 Access to all software entities placed under configuration control shall be subject to adequate controls to ensure that software is not modified by unauthorised persons and that the security of the software is maintained.

5.6.12 It shall be possible to identify all translation tools and tool versions used to produce each executable entity (see 14.3.3).

5.7 Software security

The objective of security is to protect software and data so that unauthorised persons and systems cannot read or modify them and so that authorised persons and systems are not denied access to them.

Subclauses 5.4.2 (overall security plan) and 6.2.2 (system security plan) of IEC 61513 provide requirements for security at a level of the I&C architecture and of an individual I&C system.

Although the use of software does present certain potential security threats, the main countermeasures are usually on system level, for example physical protection measures, hardwired interlocking devices. The main security requirements put on the software may help minimizing the vulnerability and may enable and support the protective measures on the system level.

This subclause provides security requirements specific to or of particular importance to software.

5.7.1 Security analysis

5.7.1.1 An analysis of the potential security threats regarding the software shall be performed. It shall take into account the relevant phases of the system and software safety life cycles. It shall determine the requirements regarding the protection and the accessibility of data and software based upon the requirements of this subclause.

5.7.1.2 The software security analysis shall be taken into account in the software or system quality assurance plan or in the software or system security plan.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

5.7.1.3 If the analysis shows that the countermeasures on the system level are not sufficient then the security analysis shall identify requirements for software design countermeasures.

5.7.2 Security design

5.7.2.1 The requirements for design countermeasures in the software identified in the security analysis shall be included in the software design requirements.

5.7.2.2 Any new software should be designed so as to minimize the vulnerability of the system.

5.7.2.3 Any pre-developed software should be configured and parameterised so as to minimize the vulnerability of the system, for example by minimizing the functions to the necessary extent or using existing security functions of the software.

5.7.2.4 The operator shall be prevented from altering stored programs.

5.7.2.5 If operator access is required to change data to operate the I&C functions then the human-machine-interface devices shall restrict access to the necessary extent.

5.7.2.6 Where required to counter possible security threats then effective protection measures shall be included in the design, configuration and/or parameter assignment of the software concerning:

– user selective access control to the software functions;
– data connections to systems with lower safety importance;
– traceability of software or parameter modifications.

5.7.2.7 The design documentation shall identify and describe the functions critical for security and the security features implemented into the software.

5.7.2.8 During verification of the software the effectiveness of the security functions shall be confirmed.

5.7.2.9 During the validation of the I&C system, effectiveness of the security functions implemented in the software shall be demonstrated through suitable tests.

5.7.3 User access

5.7.3.1 Where required the software shall support technical measures for an effective authentication procedure before user access is permitted.

5.7.3.2 Where user access is a feature critical for security that is implemented in the software then the authentication procedure implemented in the software should support technical means for a combination of knowledge (e.g. password), property (e.g. key, smart-card) and/or personal features (e. g. fingerprint) rather than rely solely on a password.

5.7.3.3 The software and the role-based access control shall be configured and parameterised so as to minimize the user access permitted to the functions and data to the necessary extent.

5.7.3.4 User access capabilities shall be protected to an appropriate extent against the opportunities and the consequences of potential security threats.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Cryptographic methods (data encryption) can be one of the possible ways to correctly implement this requirement.

5.7.3.5 No remote access that can influence the software functions or the data from outside the technical environment of the plant (e. g. from the administrative buildings or from outside of the plant) shall be implemented.

5.7.4 Security during development

5.7.4.1 The software safety development lifecycle should address potential security threats during development and maintenance activities.

5.7.4.2 There shall be provisions against hidden functions in the application software or system software (e.g. software code verification) because they could support potential unauthorised access.

5.7.4.3 If provisions could not be implemented for pre-developed software, the use of such software shall be justified considering the potential security threats, the safety importance of I&C functions concerned and the characteristics of the system and software.

5.7.4.4 Potential means for deliberate modification of software that may cause erroneous behaviour triggered by time or data conditions shall be identified and justified to be detectable during the verification activities.

6 Software requirements

6.1 Specification of software requirements

6.1.1 The software requirements shall be derived from requirements of the safety systems and are part of the computer-based system specification.

6.1.2 The software requirements shall describe what the software has to do and not how the software shall do it.

6.1.3 The software requirements shall specify:

– the application functions to be provided by the software;
– the different modes of behaviour of the software, and the corresponding conditions of transition;
– the interfaces and interactions of the software with its environment (e.g. with operators, with the rest of the I&C system, with the other systems, if any, with which it interacts or shares resources), including the roles, types, formats, ranges and constraints of inputs and outputs;
– the parameters of the software which can be modified manually during operation, if any, their roles, types, formats, ranges and constraints, and the checks to be performed by the software when they are modified;
– the required software performance, in particular response time requirements;
– what the software must not do or must avoid, when appropriate;
– the requirements of, or the assumptions made by, the software regarding its environment, when applicable;
– the requirements if any, of standard software packages.

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

6.1.4 Due to the significance of this phase of software development, the process of laying down software requirements shall be rigorous.

6.1.5 The software requirements specification shall be such that compliance of the I&C system to the requirements of IEC 61513 can be demonstrated.

Additional requirements for the software requirements specification are given in Annex A.

6.1.6 The constraints between hardware and software shall be described (A.2.1).

6.1.7 A reference to the hardware requirements specification shall be made within the software requirements specification for any hardware design impacts.

6.1.8 Special operating conditions such as plant commissioning and refuelling shall be described down to the software level for the functions that are impacted.

6.2 Self-supervision

6.2.1 The software of the computer-based system shall supervise the hardware during operation within specified time intervals and the software behaviour (A.2.2).

This is considered to be a primary factor in achieving high overall system reliability.

6.2.2 Those parts of the memory that contain code or invariable data shall be monitored to detect unintended changes.

6.2.3 The self-supervision should be able to detect to the extent practicable:

- random failure of hardware components;
- erroneous behaviour of software (e.g deviations from specified software processing and operating conditions or data corruption);
- erroneous data transmission between different processing units.

6.2.4 If a failure is detected by the software during plant operation, the software shall take appropriate and timely response. Those shall be implemented according to the system reactions required by the specification and to IEC 61513 system design rules.

This may require giving due consideration to avoiding spurious actuation.

6.2.5 Self-supervision shall not adversely affect the intended system functions.

6.2.6 It should be possible to automatically collect all useful diagnostic information arising from software self-supervision.

6.3 Periodic testing

6.3.1 For computer-based safety systems, the general principles of IEC 60671 should be used for those components which are not covered adequately by self-supervision.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

6.3.2 The software shall be designed so as to meet the requirements of periodic testing which takes place within specified maximum intervals (e.g. shut-down periods).

1) every safety function shall be coverable by periodic testing;

2) any failure of the safety functions shall be detected.

6.3.3 It should be possible to automatically collect all useful diagnostic information arising from software periodic testing.

6.3.4 The quality of the software of auxiliary devices for testing should correspond to the quality of equipment used for validation as mentioned in 10.2.

The software dedicated to auxiliary devices for testing of the class 1 systems need not be designed to comply with all software safety requirements of this standard.

6.4 Documentation

6.4.1 The main purpose of the software requirements specification document is to form the basis for software development. However, the licensing aspects should not be neglected as this document may be submitted to the regulator. Therefore, it may contain aspects of minor importance to software development which are, however, a background for licensing.

Such important aspects for licensing may be:

– risk considerations;
– recommendations for functions or engineered safety features;
– other items that provide the background for specific requirements;
– special regulatory requirements on software structure, code analysis, V&V, etc.

6.4.2 The software requirements specification shall be presented according to a standard whose formality should not preclude readability (A.2.3).

6.4.3 The software requirements specification shall be unequivocal, testable or verifiable, and achievable.

A formal language or an application-oriented language may be used to improve the coherence and completeness of aspects of the software requirements specification.

Automated tools may be used for this purpose.

6.4.4 The software requirements specification shall be provided to the relevant participants in the engineering process.

7 Design and implementation

This clause presents good practice for developing software with the appropriate safety features, which is as fault-free as possible and which is amenable to verification.

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

The software requirements specification shall be available before the design and implementation phases of program development begin.

The design and implementation phases implement the software requirements specification and provide the basis for the verification of the design and implementation of the software.

7.1 Principles for design and implementation

7.1.1 General

7.1.1.1 The software design shall include self-supervision (A.2.2).

7.1.1.2 On failure detection, appropriate action shall be taken in accordance with 6.2.

7.1.1.3 The program structure should be based on a decomposition into modules.

7.1.1.4 The program structure should be simple and easy to understand, both in its overall design and in its details.

7.1.1.5 Tricks, recursive structures and code compaction should be avoided.

7.1.1.6 The source program should be understandable by skilled engineers not involved in the software development process.

7.1.1.7 The source program should conform to documented rules designed to improve clarity, modifiability and testability.

7.1.1.8 Any non conformances against design rules should be justified.

7.1.1.9 Comprehensive and clearly written documentation shall be provided.

7.1.1.10 Communication links shall be designed in compliance with the requirements on data communication given in 5.3.1.3 of IEC 61513.

7.1.1.11 Communication links used inside the same redundancy train shall be deterministic.

7.1.1.12 The following recommendations are derived from these principles:

1) measures to implement the software safety requirements including self-supervision should be chosen at the beginning of the design (Clause B.3);
2) a top down approach to software design should be preferred to a bottom up approach (Clause B.1);
3) a conceptual model of the software architecture should be adopted at the beginning of each software project (Clause B.2);
4) the program should be written to allow easy verification (Clauses B.4 and B.5);
5) where standard software from a manufacturer or supplier is used, the requirements of Clause 15 apply in addition to item c) of Clause B.2);
6) application oriented languages should be used in preference to machine oriented ones (item e) of Clause B.5).

7.1.2 Implementation of new software in general-purpose languages

This subclause considers the situation where part or all of the category A safety functions are provided by developing new software components, using general-purpose languages.

General-purpose languages are usually either high level languages such as Ada, C, Pascal or assembly languages dedicated to the platform in use. They can be used to implement any kind of function, provided appropriate design and coding rules are followed.

7.1.2.1 The software design phase shall identify the software components to be developed with general-purpose languages.

7.1.2.2 For these components, the development process should define a detailed design phase and a coding phase.

7.1.2.3 The detailed design phase activities shall refine the design phase outputs to the point where coding with the chosen language can be performed in a systematic way (e.g. by defining the necessary algorithms, data structures, function interfaces, constraints, etc.).

7.1.2.4 The level of detail of the information provided by the detailed design phase depends on the general-purpose language used. If assembly language is used, the design shall provide detailed structured algorithms and data representation.

7.1.2.5 The coding phase shall translate the detailed design into source code, according to predefined programming rules based on the requirements of Annex B.

7.1.3 Implementation of new software in application-oriented languages

This subclause considers the situation where part or all of the category A safety functions are provided by developing new software components, using application-oriented languages.

Application-oriented languages support formalisms (such as logic diagrams or function block diagrams, etc.) that may be used to express all or part of the software requirements specification and/or part of the software design specification. These parts can be used as input to generate executable code by automated means.

7.1.3.1 The formalisms used should have the following properties: low complexity, clarity and standardisation of layout and presentation, modularity, presence of pertinent comments, avoidance of unsafe features. These properties generally facilitate understanding, verification, testing and later modification.

7.1.3.2 Application-oriented languages should have a format that is comprehensible to those engineers responsible for reviewing the software specification, for example the process engineers and the I&C engineers dealing with the systems for which the I&C functions are specified.

7.1.3.3 The application-oriented language should support a simple software structure, for example linear programs.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

7.1.3.4 The application-oriented language should allow the developers to take into account the specification of the I&C system architecture design, e.g. enable the assignment of functions to system components and support any hardware fault tolerant design features.

7.1.4 Configuration of pre-developed software

This subclause considers the situation where category A safety functions are provided by software configured with application configuration data, i.e. data specific to the intended application.

Pre-developed software may be associated with an equipment family or may be a standalone product which is then integrated with a chosen hardware platform.

7.1.4.1 Where pre-developed software is to be used, the capabilities of the software shall be evaluated and assessed (see 15.3) to ensure that it is suitable for the intended role.

The requirements for the suitability analysis are addressed in 15.3.1.2. Should the use of the software be restricted, this may be achieved by using software to envelop the pre-developed software.

For the purpose of configuring software, a tool-based approach is preferable to reduce the scope for human error.

7.1.4.2 All constraints relevant to the data shall be documented, for example allowable data formats, ranges, calculation rules.

7.1.4.3 The configuration data shall be documented.

7.1.4.4 Adequate justification shall be provided for data values used with cross-reference to design input sources.

7.1.4.5 Traceability – it should be possible to determine when any modifications were made to configuration data and by whom.

7.1.4.6 Maintainability – the development process shall ensure that through a structured approach together with the use of comments in data and/or supporting documentation the data design may be understood and maintained throughout the intended life of the system it belongs to.

See 14.3.5 for further guidance concerning the use of application data tools.

7.2 Language and associated translators and tools

7.2.1 General requirements

Even though the use of specific languages cannot be required, the following may be considered as common basic rules for languages used for the design and implementation of software for class 1 systems.

7.2.1.1 The language in use shall follow strict (or well-defined) semantic and syntax rules.

7.2.1.2 The syntax of the language shall be completely and clearly defined and documented.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

7.2.1.3 The use of the language should be restricted to a 'safe' subset where appropriate, for example be restricted to primitives that are suitable to specify the necessary functions.

7.2.1.4 Languages with a thoroughly tested translator should be used.

7.2.1.5 If no thoroughly tested translator is employed, additional verification shall provide evidence that the result of the translation is correct.

7.2.1.6 Tools for automated testing should be available.

The use of automated tools is recommended. The requirements of Clause 14 apply.

7.2.2 General purpose languages

General purpose languages for class 1 systems and their translators should not prevent the use of error-limiting constructs such as:

– translation-time variable type checking, subroutine parameters checking;
– run-time array bound checking.

Guidance for selection of language, translator, etc., is given in Annex D.

7.2.3 Application-oriented languages and associated automated code generation

7.2.3.1 Application-oriented languages should be transformed into a general purpose language by automated tools (e.g. by a code generator) prior to translation into an executable form.

7.2.3.2 The conformance of the generated code with the requirements for software design and software coding of this standard shall be assessed and non-conformances justified.

7.2.3.3 The program structure to be generated shall be defined generically, for example the position of declarations relative to code statements.

7.2.3.4 The generated code shall not be modified by direct manual action on the code.

7.2.3.5 The code shall be regenerated if the input specification has to be modified, for example with respect to findings from V&V activities.

Additional recommendations on automated code generation are given in item f) of Clause B.5.

7.3 Detailed recommendations

7.3.1 General

A set of recommendations is given in Annex B which specifies in detail the aspects identified in 7.1.

The headings of the individual recommendations of Annex B are applicable to the two major phases of software development, as shown in the following table.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Table 1 – Process and product aspects of design and implementation

Software development phase	Process aspects	Annex B	Product aspects	Annex B
Design	- Modifiability	B1.*a*	- Control and access structures	B2.*a*
	- Top down approach	B1.*b*	- Modules	B2.*b*
	- Verification of intermediate design products	B1.*c*	- Operational system software	B2.*c*
	- Modification control during the development	B1.*d*	- Execution time	B2.*d*
			- Interrupts	B2.*e*
			- Arithmetic expressions	B2.*f*
			- Plausibility checks	B3.*a*
			- Safe output	B3.*b*
			- Branches and loops	B4.*a*
			- Subroutines	B4.*b*
			- Nested structures	B4.*c*
			- Data structures	B4.*e*
			- Application-oriented languages	B5.*e*
Implementation	- Verification of intermediate design products	B1.*c*	- Modules	B2.*b*
	- Modification control during the development	B1.*d*	- Execution time	B2.*d*
	- Unit and integration tests	B4.*g*	- Interrupts	B2.*e*
	- Coding rules	B5.*d*	- Arithmetic expressions	B2.*f*
			- Plausibility checks	B3.*a*
			- Safe output	B3.*b*
			- Memory contents	B3.*c*
			- Error checking	B3.*d*
			- Branches and loops	B4.*a*
			- Subroutines	B4.*b*
			- Nested structures	B4.*c*
			- Addressing and arrays	B4.*d*
			- Data structures	B4.*e*
			- Dynamic changes	B4.*f*
			- Sequences and arrangements	B5.*a*
			- Comments	B5.*b*
			- Assembler	B5.*c*
			- Automated code generation	B5.*f*

7.3.2 Use of the requirements and recommendations

7.3.2.1 The requirements and recommendations given in Annex B shall be met during software development or otherwise justified and documented.

7.3.2.2 The justification should be done at the beginning of the design process.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

7.4 Documentation

7.4.1 During software development, the end of the design phase shall be marked by production of the software design specification.

This document serves as the basis for the formal design review and the subsequent program implementation.

7.4.2 Sufficient detail shall be included so that program implementation can proceed without further design clarification.

7.4.3 The document should be structured according to the levels of the software design process.

The software design specification may be expressed as one document or as an integrated set of documents.

7.4.4 In that case, each document shall have a defined relationship to the other documents and contain a well-bounded subject-matter.

7.4.5 Documentation formats should be selected according to the specific purpose, including:

- narrative description;
- arithmetic expressions;
- graphical representation.

7.4.6 Documents should contain appropriate diagrams and drawings.

As a general rule it is preferable to choose a graphical representation.

7.4.7 The documentation itself should comply with national standards if appropriate.

8 Software verification

8.1 Software verification process

The verification activities undertaken as part of the software development are usually the responsibility of the supplier and are undertaken by staff independent of those performing the software production; the most appropriate way is to engage a verification team.

Additional verification activities may be undertaken as part of a third party assessment of the software and of its development process in order to provide assurance that the software will meet its quality targets. There are many ways by which this independent verification role can be resourced and implemented, this often being a matter of national regulatory preference.

8.1.1 The verification team shall be composed of individuals who are not engaged in production and who have the necessary competencies and knowledge.

The following requirements define explicitly the level of independence required.

8.1.2 The management of the verification team shall be separate and independent from the management of the development team.

标准分享网 www.bzfxw.com 免费下载

8.1.3 Communication between the verification team and the development team, whether for clarification or fault reporting, shall be conducted formally in writing at a level of detail which may be audited.

8.1.4 Interactions between the two parties should aim at maintaining the independence of judgment of the verification team.

8.1.5 The verification team shall be equipped with adequate resources and means. It shall be given the time necessary to perform the verification activities.

8.1.6 The verification team shall have clearly defined responsibilities and obligations.

8.1.7 The verification team shall have the necessary authority to report its conclusions.

8.1.8 The output of each software development phase (Figure 3) shall be verified.

8.1.9 The software verification activities shall confirm the adequacy of the software requirements specification in fulfilling the system requirements assigned to the software by the system specification.

8.1.10 The software verification activities shall confirm the adequacy of the software design specification in fulfilling the software requirements.

8.1.11 The software verification activities shall confirm the compliance of the code to the software design specification as derived by the design phase. When a CASE tool with features such as automated code generation is used, dedicated requirements are given in 8.2.3.2.

8.1.12 Each production activity should be started on a basis of verified input data/ documents.

8.1.13 When undertaken as part of the software development, verification of the product of a phase should be performed before the start of the next phase and shall be performed before the completion (i.e. its verification) of the next phase.

Possible preparatory work for a subsequent phase may be done before the precedent phase has been verified and approved.

8.1.14 If input data/documents for an activity have been modified, that activity and subsequent activities shall be repeated as necessary to address potential impact.

8.1.15 All software verifications shall be completed before the system is placed into active service.

8.2 Software verification activities

The following verification activities are required.

8.2.1 Verification plan

8.2.1.1 The software verification plan shall be established prior to starting software verification activities.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

8.2.1.2 The plan shall document all the criteria, the techniques and tools to be utilized in the verification process.

8.2.1.3 It shall describe the activities to be performed to evaluate each item of software, each tool involved in the software development process, and each phase to show whether the software requirements specification is met.

8.2.1.4 The level of detail shall be such that a verification team can execute the verification plan and reach an objective judgement on whether or not the software meets its requirements.

8.2.1.5 The verification plan shall be prepared by a verification team addressing:

1) selection of verification strategies, either systematic, random or both, with test case selection according to either required functions, special features of program structure, or both (see Annex E);
2) selection and utilisation of the software verification tools;
3) execution of verification;
4) documentation of verification activities;
5) evaluation of verification results gained from verification equipment directly and from tests, evaluation of whether the safety requirements are met.

8.2.1.6 The tests performed should extensively exercise the software. Among the criteria required in the plan, test coverage criteria should be considered of prime importance.

8.2.1.7 The verification plan shall identify any objective evidence required to confirm the extent of testing. For that purpose the test coverage criteria chosen according to the design (see Annex E) shall be justified and documented.

8.2.1.8 There shall be adequate provision for the processing and resolution of all safety issues raised during the verification activities performed either during software development by the supplier or by a third-party assessment.

8.2.1.9 All safety issues shall be resolved through appropriate corrective modifications or mitigating dispositions.

8.2.2 Design verification

8.2.2.1 The design verification shall address:

1) the adequacy of the software design specification for the software requirements with respect to consistency and completeness down to and including the modular level;
2) the decomposition of the design into functional modules and the way they are specified with respect to:
   - technical feasibility of design;
   - testability for further verification;
   - readability by the development and verification teams;
   - modifiability to permit further modification;
3) the correct implementation of safety requirements.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

8.2.2.2 The result of the design verification shall be documented.

8.2.2.3 The documentation shall include the conclusions and identify clearly issues that need actions, such as:

– items which do not conform to the software requirements;
– items which do not conform to the design standards;
– modules, data, structures and algorithms poorly adapted to the problem.

8.2.3 Implementation verification

Regardless how the software code is developed, test methods used to verify the implementation phase output should be selected according to Table E.4.2.

8.2.3.1 Verification of implementation with general-purpose languages

8.2.3.1.1 Code verification

8.2.3.1.1.1 The implementation verification shall include activities based on source code analysis and tests. The source code analysis may be performed using verification methods such as code inspection, possibly with the assistance of automated tools.

8.2.3.1.1.2 Code verification activities should begin with module source code analysis followed by module testing.

8.2.3.1.1.3 Module verification shall show that each module performs its intended function and does not perform unintended functions.

8.2.3.1.1.4 A module integration test shall be performed to show at an early stage of development that all modules interact correctly to perform the intended function. If a CASE tool is used for this, it shall also meet the relevant requirements of Clause 14.

8.2.3.1.1.5 The results of code verification shall be documented.

8.2.3.1.2 Software test specification

The software test specification is one of the principal documents to be addressed by the verification plan.

8.2.3.1.2.1 This document shall be based on the software design specification and a detailed examination of the software requirements.

8.2.3.1.2.2 It shall give detailed information on the tests to be performed addressing each of the components of the software (modules and their constituents).

8.2.3.1.2.3 The software test specification shall include:

1) the environment in which the tests are run;
2) the test procedures;
3) acceptance criteria, i.e. a detailed definition of the criteria to be fulfilled in order to accept modules and major software components on subsystem and system levels;
4) procedures for fault detection;
5) a list of all documents that should be produced.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

8.2.3.1.3 Software test report

8.2.3.1.3.1 The software test report shall present the results of the verification described in the software test specification stating whether or not the software performs in accordance with the software design specification.

8.2.3.1.3.2 This document shall report all software design and implementation discrepancies discovered during the tests.

8.2.3.1.3.3 The software test report shall include the following items both for the module and major design levels:

1) hardware configuration used for the test, identification and justification of any test harness hardware and software used;
2) storage medium used and access requirements of the final code tested;
3) input test values;
4) expected and achieved output values;
5) additional data regarding timing, sequence of events, etc.;
6) conformance with acceptance criteria given in the test specification;
7) fault incident log which describes the characteristics of each fault.

8.2.3.2 Verification of implementation with application-oriented languages

The use of application-oriented languages is generally considered to improve the quality (i.e. reduce the level of design and implementation faults introduced during the engineering process) and maintainability of application software.

8.2.3.2.1 Application software which is automatically generated from a specification using an application-oriented language should have a systematic structure to support effective verification.

8.2.3.2.2 Software written in application-oriented languages shall be verified to be functionally correct and consistent, for example by manual inspection or by the use of automated tools which allow simulated running of the software in a debug environment.

8.2.3.2.3 The verification process shall confirm that:

– all the design features are correctly implemented;
– the software functionality is consistent with the objectives defined in the software requirements specification;
– the software design is compliant with the applicable standards stated in the QA plan.

8.2.3.2.4 An appropriate and justified selection of techniques such as animation, tests, reviews, walkthroughs, formal analyses and proofs shall be applied to improve the understanding of specifications and to verify their functional correctness and consistency.

8.2.3.2.5 Software tools used for verification or validation shall be qualified as required by Clause 14.

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

8.2.3.3 Verification of configuration of pre-developed software

This subclause considers the situation where category A safety functions are provided by pre-developed software configured with application configuration data, i.e. data specific to the intended application. Software may be associated with an equipment family or may be a standalone product which is then integrated with a chosen hardware platform.

A pre-requisite of this subclause is that the pre-developed software has been qualified for its intended application (see Clause 15).

8.2.3.3.1 Verification of data shall be performed using a combination of inspection/analysis and/or test. The appropriate means of testing the impact of the configuration data shall be analysed and documented (giving consideration to the role of simulation, emulation, test rigs and prototypes).

Testing of the configured software is covered in Clause 9.

Some aspects of the required system functionality may be expressed in terms of data, for example a temperature setpoint, and for such items the process is to replicate this data value in the application configuration data for the system. For such data, verification by inspection or by electronic comparison can confirm correctness to very high levels of confidence.

Other aspects of data may have to be developed from the system requirements, for example the allocation of signal inputs to specific input cards, the contents of message buffers.

8.2.3.3.2 The documentation structure adopted for any configuration of pre-developed software shall ensure that intermediate design documents are produced where necessary to document the design process comprehensively, for example to justify how a time response requirement has been addressed by configuring the scan frequency and the frequency of the transmittal of a message buffer.

8.2.3.3.3 The verification process shall confirm that the configuration data is consistent with the design documentation and with any established constraints and rules.

This could be achieved by a manual inspection process, or using a tool based approach, or by a combination of both techniques.

8.2.3.3.4 If data is produced using a tool-based approach and if it is intended to omit the data verification step and proceed directly to system integration, justification shall be provided that sufficient confidence in the correctness of the tool can be achieved.

For further guidance concerning the verification of data produced using application data tools, see 14.3.5.

9 Software aspects of system integration

The process of system integration is the combining of verified hardware and software modules into subsystems (computer units) and finally into the complete system.

This process consists of four activities:

a) assembling and interconnecting hardware modules as defined in design documents;

b) building the target software from software modules;

c) loading the target software into the target hardware;
d) verifying that:
   – the software complies with its design specification;
   – the hardware/software interface requirements have been satisfied;
   – the software is capable of operating in that particular hardware environment.

The software aspects of system integration are the above-mentioned activities b), c) and d).

This clause gives additional requirements for the system integration plan as well as requirements about the software aspects of system integration, in supplement to 6.1.4 of IEC 61513.

9.1 Software aspects of system integration plan

9.1.1 This plan shall be prepared and documented in the design phases and verified against the class 1 system requirements.

9.1.2 This plan shall be prepared sufficiently early in the development process to allow any integration requirements to be included in the design of the system and its hardware and software.

9.1.3 This plan shall specify the standards and procedures to be followed in the system integration.

9.1.4 This plan shall document those provisions of the system quality assurance plan that are applicable to the system integration.

9.1.5 The system integration plan shall take into account the constraints, within any set of hardware and/or software modules, made by the design of the system, of the hardware and of the software. The plan shall include the requirements for procedures and control methods covering:

– system configuration control (see 5.6);
– system integration;
– integrated system verification;
– fault resolution.

9.1.6 The system integration plan shall define both aspects of configuration management (identification and control) according to the requirements of 6.2.1.2 of IEC 61513.

9.1.7 In the process of verifying the individual hardware and software modules, certain aspects of the design of these modules may be verified at the level of subsystems (computer units) or at the level of the complete system if more practical. When verification by testing is not feasible at these levels, then all requirements of the individual module design shall be verified by other means.

9.1.8 All interdependencies between the verification of the individual modules and the verification of the integrated system shall be documented in the system integration plan.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

9.2 System integration

The specific procedures for the system integration depend on the nature of the system design. They cover items a) to c) stated in Clause 9.

9.2.1 Such procedures shall be established and documented by the system integration plan, and shall cover the following activities:

– the acquisition of the correct modules according to the system configuration management plan (6.2.1.2 of IEC 61513);
– the integration of the hardware modules into the system (e.g. module position, memory address, selection, interconnection wiring);
– the linkage of software modules and the loading of the target software into the target hardware;
– the preliminary functional test of the integrated system functions (see requirements below);
– the documentation of the integration process and the system configuration that will be subjected to test;
– the formal release of the integrated system for testing.

9.2.2 If the resolution of a fault requires a modification to any verified software or any design document, that fault shall be reported according to the procedures established by 9.4.

Any faults detected during the system integration that are strictly mistakes in the integration process itself, and do not affect any project document, may be corrected without formal fault report.

9.3 Integrated system verification

The system verification determines whether or not the verified hardware and software modules and the subsystems have been properly integrated into the system and that they are compatible and perform as specified.

9.3.1 The system shall be as complete as is practical for this testing.

9.3.2 The test cases selected for system verification shall exercise all module interfaces as well as the basic operation of the modules themselves.

9.3.3 In the system integration plan, the simulation of any part of the system or its interfaces shall be demonstrated to be essential and equivalent to the actual part.

9.3.4 The system integration plan shall identify the tests to be performed for each computer unit interface requirement.

9.3.5 The integrated system test shall be reviewed and the test results evaluated by a verification team with a good knowledge of the system specification.

9.3.6 Equipment used for system verification shall be calibrated as required.

9.3.7 Quality assurance measures shall be established for software tools used for verification, commensurate with the importance of those tools for verification.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

9.3.8 The verification of the integrated system shall demonstrate that all system components have appropriate performances (e.g. processing units and communication devices).

9.4 Fault resolution procedures

9.4.1 Procedures for the reporting and resolution of faults found during system integration verification shall be established before integrated system verification begins.

9.4.2 These procedures shall apply to all faults found during the system verification as well as those found during the integration functional test that require modifications to verified software or system design documents.

9.4.3 They shall ensure that any required re-verification of system design, hardware or software modules is performed according to the system configuration management plan.

9.4.4 They shall ensure that any required modification of system design, hardware or software is carried out according to the modification procedure of Clause 11 and to the system configuration management plan.

9.4.5 An evaluation of each fault reported shall be made to determine whether any systematic deficiency exists and also to determine whether the fault was of such a nature that it should have been detected at an earlier phase of the verification.

9.4.6 If this is found to be the case (should have been detected at an earlier phase), then an investigation of that earlier stage of the verification shall be conducted to determine whether any systematic deficiency of the verification exists.

9.4.7 If the evaluation of faults shows that there is a systematic deficiency of the verification, causing faults in software modules to remain undetected, then the deficiency shall be identified, corrected or justified.

9.5 Software aspects of integrated system verification report

9.5.1 The results of the integrated system verification shall be documented in a report (Annex F).

9.5.2 This report shall identify the hardware and software used, the test equipment used, its calibration and software/hardware set-up parameters, the simulation of system or interface components, and any test results discrepancy found along with the corrective actions taken according to 9.4.

9.5.3 The test results shall be retained in a form that is auditable by persons not directly engaged in the verification plan.

9.5.4 The resolution of all reported faults and the results of the subsequent evaluation shall be documented in sufficient detail and in a manner that is auditable by persons not directly engaged in the system development and verification plan.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

10 Software aspects of system validation

a) Testing shall be performed to validate the system and its software in accordance with the class 1 systems requirements to be satisfied by the integrated system.

b) Validation shall comprise tests performed on the system in the final assembly configuration including the final version of the software.

10.1 Software aspects of the system validation plan

10.1.1 The system validation shall be conducted in accordance with a formal system validation plan.

10.1.2 The plan shall identify static and dynamic test cases.

10.1.3 The computer system validation plan shall be developed and the results of the validation evaluated by individuals who did not participate in the design and implementation.

10.2 System validation

10.2.1 The system shall be exercised by static and dynamic simulation of input signals present during normal operation, anticipated operational occurrences and accident conditions requiring action by the computer-based system under test.

10.2.2 Each reactor category A function of the system shall be confirmed by representative tests of each trip or protection parameter singly and for relevant combinations.

10.2.3 The tests shall:

– cover all signal ranges, and the ranges of computed or calculated parameters in a fully representative manner;
– cover the voting and other logic and logic combinations comprehensively;
– be made for all trip or protective signals in the final assembly configuration;
– ensure that accuracy and response times are confirmed, and that correct action is taken for any equipment failure or failure combination;
– be made for all other functions which have a direct impact on reactor safety (e.g. vetoes, interlocks).

10.2.4 In addition, the required input signals and their values, the anticipated output signals and the acceptance criteria shall be stated in the system validation plan.

10.2.5 Equipment used for validation shall be calibrated and configured (hardware and software parameters) as appropriate.

10.2.6 Equipment used for validation should be shown to be suited to the purpose of the system validation.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

10.3 Software aspects of the system validation report

10.3.1 The system validation report shall document the results of the software aspects of the validation of the system.

10.3.2 The report shall identify the hardware, the software and the system configuration used, the equipment used and its calibration and the simulation models used.

10.3.3 This report shall also identify any discrepancies.

10.3.4 This report shall summarise the results of the system validation.

10.3.5 This report shall assess the system compliance with all requirements.

10.3.6 The results shall be retained in a form that is auditable by persons not directly engaged in the validation.

10.3.7 Software tools used in the validation process should be identified as an item in the validation report.

10.3.8 Simulations of the plant and its systems used for the validation shall be documented.

10.4 Fault resolution procedures

10.4.1 Procedures for the reporting and resolution of faults found during system validation shall be established and referenced by the system validation plan.

10.4.2 These procedures shall apply to all faults found during system validation that require modifications to system design or software.

10.4.3 They shall ensure that any required re-verification of system design, hardware or software is performed according to the system configuration management plan.

10.4.4 They shall ensure that the modification of system design and software is carried out according to the modification procedure of Clause 11 and to the system configuration management plan.

10.4.5 An evaluation of each fault reported shall be made to determine whether the initiating fault was of such a nature that it should have been detected at an earlier phase.

10.4.6 If this is found to be the case (should have been detected at an earlier phase), then an investigation of that earlier stage shall be conducted to determine whether any systematic deficiency exists.

11 Software modification

A software modification is a change made to the software which usually impacts both the executable code and the documentation.

A software modification may be requested for reasons such as:

– changes to functional requirements;
– changes to the software environment;

标准分享网 www.bzfxw.com 免费下载

– changes to the hardware;
– anomalies found during test or operation.

A software modification may be requested during the development phase or after delivery.

a) Prior to the implementation of any software modification, a formal software modification control procedure shall be established and documented, which shall include requirements for verification and validation.
b) This procedure shall state how the requirements of this clause are addressed.

11.1 Modification request procedure

11.1.1 For a software modification to be considered, the following steps shall be followed:

– generation of a software modification request;
– evaluation of the request;
– decision.

11.1.2 A software modification request shall be generated and uniquely identified, stating:

– reason for request;
– aim;
– functional scope;
– originator;
– date of initiation.

11.1.3 The software modification request shall be included as a part of the documentation of the software modification. If the software is modified in the context of a system design modification, then the documentation of the software modification shall be a part of the documentation of the system design modification.

11.1.4 The modification request shall be evaluated independently.

11.1.5 The evaluation of the modification request shall examine its relevance to ensure that:

– the proposed changes have been clearly and unambiguously defined;
– the proposed changes will correct the causes of the anomaly where a change has resulted from an anomaly report;
– the proposed changes do not degrade the ability of the software to provide required category A safety functions;
– the benefits of implementing any changes are not outweighed by the disruption that implementing them may cause (system failures may be caused by incorrectly designed or implemented changes).

11.1.6 The following items shall also be examined in the evaluation of the modification request:

– technical feasibility;

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

- impact upon the rest of the system (e.g. memory extension) or upon other equipment (e.g. test systems) in which case the request for modification addressing this impact area shall be documented;
- effects of possible changes in the methods, tools or standards to be applied in the execution of the modification (compared to those which were applied for the development of the version of the software to be modified);
- impact upon software itself, including a list of affected modules;
- impact upon performance (including speed, accuracy, etc.);
- strategy and necessary effort for verification and validation to ensure that the correctness of the existing software is maintained; the analysis of the software re-verification needed shall be documented in an auditable form;
- the set of documents to be reviewed.

The evaluation process may consist of a number of phases.

The initial request may be reviewed for relevance and feasibility before any detail impact assessment work has been performed.

When the full impact has been evaluated, a second more thorough evaluation may be performed.

The software modification request is pending until the decision is made, which may be:

- to reject the request; in this case, it is sent back with justification;
- to require a detailed analysis, resulting in a software modification analysis report;
- to accept and process the request.

11.1.7 If a software modification analysis report has been required, this report shall be written by software personnel knowledgeable in the software of the system.

11.2 Procedure for executing a software modification

11.2.1 For modifications which are to be implemented on operational equipment where it is not practical to perform adequate testing due to operational considerations, the software supplier should have access to a test configuration which is identical to the real system in all relevant aspects (including installed machine, translator, testing tools, plant simulator, etc.) to ensure the validity of the modifications.

11.2.2 The modification procedure appropriate for any particular change will depend upon the affected phases of the development process:

- for a change of the software requirements specification, the whole software development process for any part of the I&C system impacted by the change shall be re-examined;
- a change during the development shall be reviewed in terms of its potential impact upon corresponding lower levels;
- the modification shall be carried out according to the rules given in Clause 7.

11.2.3 After implementation of the modification, the whole or part of the verification and validation process described in 8.1 and Clause 10, shall be performed again according to the software modification impact analysis (see 11.1).

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

11.2.4 All the documents affected by the modification shall be corrected and refer to the identification of the software modification request.

11.2.5 A software modification report shall sum up all the actions made for modification purposes.

11.2.6 All these documents shall be dated, numbered and filed in the software modification control history for the project.

11.2.7 The strategy for deployment of modified software in an operating plant shall be evaluated in respect of its impact on voting strategy, maintenance and data communications. Depending upon the outcome of this evaluation, software modifications may be deployed gradually, allowing testing of new software on each single redundancy train in turn with parallel operation with the non-modified trains.

11.3 Software modification after delivery

The reasons for such modification include:

– an anomaly report;
– a functional requirements change after delivery;
– technological evolution;
– a change in operating conditions.

11.3.1 In case of an anomaly, an anomaly report shall be written giving the symptoms, the system environment and system status at the time at which the anomaly was discovered and the suspected causes.

11.3.2 If unexpected, apparently incorrect, unexplained or abnormal behaviour is experienced after delivery, an anomaly report should be raised by the operating staff giving details of the behaviour, the software and hardware configurations and the activities in hand at the time.

11.3.3 The report should include the originator, location, date, circumstances and a serial number. The reports should be reviewed by the development team, assigned to categories of importance and resolved by a response to the operational staff.

It is an advantage if the anomaly report and clearance procedure is developed from a similar process adopted during verification and validation.

11.3.4 Fault correction requires the generation of a software modification request, the execution of which shall follow the procedure described in 11.1.

11.3.5 In case of a change in the software requirements specification, the whole software development process shall be re-examined for that part of the system impacted by the change.

11.3.6 Any new hardware requirements and capabilities shall be examined with respect to their potential impact on the software.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

11.3.7 This evaluation should include all hardware considerations reviewed in the original software design.

If it can be shown that the modified system does not impact the software requirements specification, a simplified procedure may be used to implement the modification either at the design or coding phase.

Item b) of 6.3.6.1 of IEC 61513 recommends that when the software modification is completed, it is included in a change package. It also requires that the documentation of this package describes the means of implementing the modification upon the operational equipment, or a reference to approved existing procedures may be provided.

11.3.8 In all cases, after implementation of the modification upon the operational equipment, a document shall be issued which gives the date of the implementation and the results of any specified tests or observations required by the implementation procedure.

11.3.9 This document shall be filed in the software modification control history for the project.

12 Software aspects of installation and operation

This clause provides requirements for the interaction of operators with class 1 computer-based systems during installation and operation. These requirements address:

– the on-site installation of software;
– the on-site software security;
– the adaptation of software to on-site conditions;
– training.

12.1 On-site installation of the software

A test procedure shall be provided to verify the integrity of the installed software with respect to response, calibration, functional operation and interaction with other systems.

12.2 On-site software security

12.2.1 A security assessment of the on-site configuration and parameter assignment shall take place to verify that suitable countermeasures have been implemented against potential security threats.

12.2.2 When applicable the software shall be configured and parameterised so as to collect all relevant information for the periodic security auditing and reporting of the I&C system.

12.2.3 Software modification activities shall be systematically prepared taking into account potential security threats.

12.2.4 Proceeding from the security concept for normal plant power operation, the deviations which are necessary to perform the planned software commissioning and modification activities shall be identified.

Deviations to be considered may include special software options which are blocked during power operation, alarm annunciation functions which are blocked during modification, the use of interfaces which are blocked or switched off during power operation and the use of service stations and tools, as well as the required local presence of modification personnel.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

12.2.5 Any deviations shall be compensated for by additional means of quality assurance, such as additional administrative and analytical measures during and/or after the modification activities to ensure the integrity of the software.

12.2.6 Tools and equipment used for on-site software modification should match a level appropriate to their potential threat to the security of the system.

12.2.7 New data files or new software versions implementing a security-related change shall be verified to confirm that the security requirements have been properly addressed.

12.2.8 The procedure for installing software or data on-site shall include and specify checks of the software integrity to be performed before the I&C system is put into full operational use.

12.3 Adaptation of the software to on-site conditions

An appropriate procedure and/or locking device shall be established to prevent the operator inadvertently or incorrectly changing parameters that can affect the set points or other modifiable data items of the class 1 system.

12.4 Operator training

12.4.1 Training programme

In order to achieve safe plant operation, operator behaviour is as important as equipment reliability.

12.4.1.1 Therefore an operator training programme for the safety systems and its software use shall be provided both for plant operators and instrumentation and control specialists consistent with the complexity of the protective functions implemented.

12.4.1.2 The training programme shall address operations under normal and abnormal plant conditions.

12.4.1.3 The training programme shall address all relevant computer-based system operator interface devices.

12.4.1.4 Specific training in the recognition of hardware and software abnormalities should also be included in the programme.

12.4.2 Training plan

12.4.2.1 A training plan consistent with the principles of the training programme shall be established.

12.4.2.2 A user manual for the I&C system shall be provided for use by the operations and maintenance staff.

12.4.2.3 The user manual should define each operator interface device. Each function of each device shall be explained and illustrated in accordance with its complexity.

12.4.3 Training system

12.4.3.1 Operator training shall be conducted on a training system which is equivalent to the actual hardware/software system.

12.4.3.2 The plant stimulus to this system shall be provided by a test system capable of simulating normal and abnormal reactor conditions.

13 Defences against common cause failure due to software

This clause provides requirements for defences against software design and coding faults which can lead to common cause failures (CCF) of functions classified as category A according to IEC 61226.

13.1 General

CCF may occur in the I&C systems and equipment implementing different lines of defence against the same PIE (see 5.3.1 of IEC 61513). Software by itself does not have a CCF mode. CCF is related to system failures arising from faults in the functional requirements, system design, or in the software.

Defence in depth is required by the IAEA (see 2.9 of IAEA Requirements NS-R-1) to be applied to all safety activities, whether organisational, behavioural or design related, to ensure that there are overlapping defences so that if a failure should occur in a subsystem, it would be compensated for or corrected in the integral system.

The single-failure criterion (see 5.34 to 5.39 of IAEA Requirements NS-R-1) requires that the assembly of safety systems has the ability to meet its purpose despite a single random failure assumed to occur anywhere in the assembly.

13.1.1 Software faults are systematic not random faults and therefore the single-failure criterion cannot be applied to the software design of a system in the same manner as it has been applied for hardware. When the defence-in-depth concept is applied, possible effects of CCF due to software inside each defence layer and between redundant layers should be considered and appropriate countermeasures should be adopted throughout the development process and in the evaluation processes (if software CCF is a potential failure mode), for example

1) in the design and implementation, verification and validation of each individual defence layer; and
2) in the evaluation of the independence and diversity of redundant defence layers.

A means of enhancing the reliability of some systems and reducing the potential for certain CCFs is the use of diversity (see 4.23 to 4.31 of IAEA Safety Guide NS-G-1.3).

The rationale for defence against software faults is that any software fault will remain in the system or channel concerned until detected and corrected, and can cause failure if a specific signal trajectory challenges it. If two or more systems or channels implementing different lines of defence for the same PIE (see 5.3.1.5 of IEC 61513) contain the fault, and are exposed to specific signal trajectories within a sensitive time period, both (or all) systems or channels can fail, which is called a CCF. A more detailed description of these conditions is given in Clause G.1.

13.1.2 The potential for CCF due to software should therefore be considered during design. If postulated conditions of CCF can be foreseen, design changes and defence features, including software diversity, may be needed for protection against CCF due to software.

The degree of improvement of defence against CCF and improvement in reliability that can be achieved by diversity cannot be quantified. Judgement is required based on an evaluation of the qualitative reliability which the software can achieve.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

If human errors are made before software design starts, they may lead to faults of requirements and potential system failures against which software engineering alone cannot provide a defence. Defence against such CCF is discussed at the system level in 5.3.1.5 of IEC 61513.

If human errors are made during the software engineering process, they may lead to software faults and potential system failures. Where such faults lead to the failure of more than one line of protection the failures are considered to be CCFs due to software.

13.2 Design of software against CCF

The basic, and most important, defence against CCF due to software is to produce software of the highest quality, i.e. as error free as possible. The extent of coverage of self-monitoring features, such as for data plausibility, parameter range checking, and loop timing, etc. as addressed by 6.2, 7.1 and A.2.2 is a further important factor in limiting the potential for CCF due to software.

The use of well-developed software engineering methods with software tool support for software development and verification can help to reduce the number of human design decisions and so potentially reduce the number of faults in the developed software.

13.3 Sources and effects of CCF due to software

13.3.1 An analysis of the potential for CCF due to software shall be performed and documented at the system level and/or at the level of the total I&C architecture of the I&C systems important to safety of the NPP.

NOTE 1 Requirements on the I&C architecture are given in 5.3.1 of IEC 61513.

NOTE 2 Requirements on the architecture of the individual I&C systems are given in 6.1.2 of IEC 61513.

13.3.2 The analysis should include the following steps:

1) identification of the software components used in the system or I&C architecture;
2) analysis of the potential CCF due to these components within the system or the I&C architecture;
3) analysis of the possible effects of these CCF.

NOTE Performing an analysis of the potential effects of faults does not obviate the need to perform verification and validation activities as required by this standard in Clause 8. The purpose of such an analysis is to reveal any weaknesses in the design, and hence initiate changes to the design, and/or to improve confidence in the software design.

13.3.3 If common modules are used in more than one system, they shall be identified and assurance of the reliability of such common modules shall be assessed. Methods supporting the demonstration of correctness are given in Clause G.4.

13.3.4 Data transmitted inside a computer-based system or between computer-based systems shall be identified. An analysis shall be performed to determine if faulty data can lead to a CCF in receiving computers or systems.

13.3.5 The potential for plant conditions to subject the same software running in different hardware to identical and simultaneous signal trajectories and hence reveal the same software fault in several channels or functional paths shall be assessed.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

NOTE Failures may be caused by signal trajectories which were not considered during the design, verification and validation of the software of the individual channels or systems.

13.3.6 Software modification activities (see Clause 11) have the potential to cause CCF and the processes used for software or data change assessment should provide assurance that such faults are not introduced.

13.3.7 The analysis of potential CCF due to software shall be performed and documented as part of the assessment of the defence against CCF of the design of the I&C architecture (see 5.3.3 of IEC 61513).

13.3.8 If the analysis identifies an unacceptable threat arising from CCF due to software then the design of the software or of the I&C architecture shall be improved. Methods supporting the implementation of CCF defences are given in Clause G.3 and methods supporting the implementation of diversity features are given in Clause G.5.

13.4 Implementation of diversity

13.4.1 Implementation of diversity should use independent systems with functional diversity. If functional diversity is not appropriate or possible, the use of system diversity, diverse software features and diverse design approaches should be considered. Features of importance are given in Clause G.5. The techniques chosen for defence against CCF shall be documented and justified according to the analysis made.

13.4.2 At the software level, defences against CCF should be based on an appropriate selection of techniques, such as

1) guarantee of diversified operational conditions of the software,
2) defences against error and failure propagation,
3) reduction of the negative effects of CCF,
4) use of software diversified by different specifications for different implementations of the same functional requirement.

NOTE 1 Differences in design and implementation methods should be considered for inclusion but are not required.

NOTE 2 N-version programming is not recommended.

13.5 Balance of drawbacks and benefits connected with the use of diversity

If diverse software is used and claimed, the drawbacks and benefits on the overall reliability of the software should be justified on the basis of the above analysis, and documented (see 4.27 of IAEA Safety Guide NS-G-1.3, and 3.81 to 3.85 of IAEA Safety Guide NS-G-1.2). Potential benefits, drawbacks and justification aspects are given in Clause G.6.

14 Software tools for the development of software

14.1 General

This subclause expands on existing requirements of this standard for software tools used in the development of software for computers in safety systems of nuclear power plants.

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

The use of appropriate software tools can increase the integrity of the software development process, and hence software product reliability, by reducing the risk of introducing faults in the process. The use of tools can also have economic benefits as they can reduce the time and human effort required to produce software. Tools can be used to automatically check for adherence to rules of construction and standards, to generate proper records and consistent documentation in standard formats, and to support change control. Tools can also reduce the effort required for testing and to maintain automated logs. Tools can also be necessary because a specific development methodology requires their use.

14.1.1 Tools are most powerful when they are defined to work co-operatively with each other. Care should be taken not to require tools to undertake tasks beyond their capability, for example, they cannot replace humans when judgement is involved. In some cases, tool support is more appropriate than complete automation of the process. When selecting a tool, the benefits and risk of using a tool must be balanced against the benefits and risk of not using a tool. The important principle is to choose tools that limit the opportunity for making errors and introducing faults, but maximise the opportunity for detecting faults.

Tools within the scope of this standard include those used to support the capture of requirements and those used to support the transformation of requirements into the final system code and data (there may be many intermediate steps). This standard also includes those tools used to directly support the performance of verification, validation and testing, tools for the preparation and control of application data (see 14.3.5), and tools for the management and control of the processes and products involved in the software development.

Off-line tools, used to calculate important variables used during the design and analysis of equipment important to safety, are considered beyond the scope of this standard. Word processors, project management tools, and other office administration tools used to support tasks not directly concerned with software development are also not within its scope.

14.2 Selection of tools

14.2.1 The tools used to develop software for class 1 systems shall be selected to support the software engineering process. The criteria and process of tool selection to be followed are described in 14.3.1. The limits of applicability of all tools shall be identified and documented. The tools and their output shall not be used outside their declared limits of application without prior justification.

14.2.2 The tools used in the development of software in class 1 systems in nuclear power plants shall be verified and assessed to a level consistent with the tool reliability requirements, the type of tool (see items 1) to 5) of 14.2.3 and the potential of the tool to introduce faults.

14.2.3 Tools shall have sufficient reliability to ensure that they do not jeopardise the reliability of the end product. For example, a tool could adversely affect the development of software by introducing errors, by producing a corrupted output, by failing to detect a fault that is already present.

Principles of defence in depth and diversity adopted for I&C architecture may be considered when selecting tools, in order to reduce the reliability requirements on individual tools.

The level of verification and assessment required for a tool also depends on the type of tool and whether the output of the tool can be fully verified or validated. Types of tools are:

1) transformation tools such as code generators, compilers, and those that transform text or a diagram at one level of abstraction into another, usually lower, level of abstraction;
2) verification and validation tools such as static code analysers, test coverage monitors, theorem proving assistants, and simulators;
3) diagnostic tools used to maintain and monitor the software under operating conditions;
4) infrastructure tools such as development support systems;
5) configuration control tools such as version control tools.

14.3 Requirements for tools

Requirements for tools are presented by topic:

a) software engineering environment;
b) tool qualification;
c) tool configuration management;
d) translators/compilers;
e) application data tools;
f) automation of testing.

14.3.1 Software engineering environment

14.3.1.1 Tools should be used to support all aspects of the software life cycle where benefits result through their use and where tools are available. Analysis of the software engineering environment and development processes shall be performed to determine the strategy for providing tool support. The results of the analysis should be documented. If tools are not available, the development of new tools may need to be considered.

Examples of processes and operations that can benefit from tool support are:

1) production and checking of specification, design and implementation (see Annex H);
2) tools operating on the language or a subset of the language (see 14.3.4);
3) preparation, verification and validation, and management of application data (see 14.3.5);
4) automation of testing (see 14.3.6).

14.3.1.2 Criteria for the selection and evaluation of tools for the software engineering environment should be developed and prioritised to allow trade-offs prior to use. The criteria should be structured by software quality characteristics as defined in ISO/IEC 9126: functionality, reliability, usability, efficiency, modifiability, and portability. The criteria may include other items like licensing effort and resources required to use a tool, the rigour of the quality plan under which the tool was developed, vendor tool history, and alternatives to tool use.

14.3.1.3 The tool support for the software engineering environment shall be analysed and documented to address:

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

1) how each process is, or is not, supported by tools;
2) the precise identification of the tools (for example, name, version number) and possibly their configuration;
3) how each tool is to be used within the project;
4) how the output of each tool is to be verified and/or validated against its input;
5) how other tools or processes mitigate the consequences of a fault in the tool, including mitigation of potential errors during production and preparation of data for on-line use;
6) how tools interface with other tools, i.e. tools may be required to use, process, and deliver information shared by other tools or part of a repository;
7) how tools provide a consistent interface to users and to the remainder of the software engineering environment;
8) how tools are suitable for the software engineering methods selected;
9) the error detection and handling capability of tools;
10) how tools satisfy the context of use including users, equipment, environment, and user's tasks, to maximise user effectiveness and minimise the impact of user errors;
11) how tools prevent unauthorised use/misuse or modification.

14.3.1.4 The modification, upgrade or replacement strategy for tools shall be documented and justified. This strategy is a part of the operational software modification strategy which shall ensure that the operational software can be adapted or corrected throughout its use in the NPP. It shall also ensure that moving to a new version of a tool is justified and the new version of the tool is suitably qualified, i.e. assessed against the requirement of this standard.

14.3.1.5 Tools used to provide diversity, i.e. compilers used for the development of multiple-version dissimilar software systems, should be demonstrated to be dissimilar. This may be achieved by showing that:

1) each tool was obtained from a different supplier (for example, one tool could be developed and the other tool could be purchased off the shelf); or
2) the tools have different input and/or output languages; or
3) the tools have dissimilar requirements and design processes.

14.3.2 Tool qualification

14.3.2.1 A tool qualification strategy shall be produced and the tools shall be qualified in accordance with that strategy. The strategy shall consider the reliability requirements of the tool and the type of the tool.

14.3.2.2 The qualitative reliability requirements of a tool shall be determined considering:

1) the consequences of a fault in the tool;
2) the probability that a tool causes or induces faults in the software implementing the safety function;
3) what other tools or processes mitigate the consequences of a fault in the tool.

NOTE Principles of defence in depth and diversity can reduce the reliability requirements on tools.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

14.3.2.3 Tool qualification strategy shall consider:

1) analysis of tool development process and vendor tool history;
2) adequacy of tool documentation to allow verification of tool output and ease of learning;
3) testing or validation of the tool;
4) evaluation of the tool over a period of use;
5) feedback of experience with tool use.

NOTE Clause 15 contains qualification requirements for the use of pre-developed software that should also be considered for tool qualification strategy.

14.3.2.4 Tool outputs should be systematically verified (for example, by test, analysis, or comparison with the output of functionally similar tools), if the output is to be included in the final software.

14.3.2.5 Tools shall be subject to evaluation and assessment as described in Clause 15, or developed according to QA requirements of Clauses 1 to 12 of this standard, unless:

– the tool cannot introduce faults into the software (for example a word processor tool for documentation), or;
– there is mitigation of any potential tool faults (e.g. by process diversity or system design, see item 5) of 14.3.1.3, or;
– the tool output is always systematically verified (see item 4) of 14.3.1.3. The qualification process may take into account experience of prior use of the tools where the tool has previously been demonstrated adequately through use in a relevant safety related application.

14.3.3 Tool configuration management

14.3.3.1 All tools shall be under configuration management to ensure the complete identification of selected tools (including name, version, variant, and possibly configuration) and the tool parameters used to generate baselined software (see 5.6).

NOTE This is useful not only for the final software consistency, it also helps in assessing the origin of a fault, which may lie in the source code, in the tool or in the tool parameters. It may also be necessary in the assessment of the potential for CCF due to software tools.

14.3.3.2 Records documenting the error history and limitations of tools shall be maintained throughout the life of any tool whose output can introduce a fault into the final software.

14.3.3.3 Any modification of a tool shall be verified and assessed.

14.3.4 Translators/compilers

This subclause presents requirements specifically related to translators/compilers. The size and complexity of many compilers can make it extremely difficult to demonstrate that a compiler works correctly. However, extensive experience of use can increase confidence that the compiler works correctly.

14.3.4.1 Translators/compilers should be selected on the basis of guidance criteria relevant to translators/compilers in this subclause (which complement the recommendations in Annex D).

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

14.3.4.2 Translators/compilers shall not remove without warning defensive programming or error-checking features introduced by the programmer.

14.3.4.3 The use of compiler optimisation should be avoided. It shall not be used if it produces object code that is excessively difficult to understand, debug, test and validate.

NOTE Code optimisation can be used to meet performance requirements due to constraints in hardware speed and storage limits. In exceptional cases, the alternative use of assembly code can be considered in addition to changing the hardware platform.

14.3.4.4 Where optimisation is used, tests, verification and/or validation shall be performed on the optimised code.

14.3.4.5 Libraries which are used in the target system shall be considered as sets of pre-developed software components. Those components of the library used shall be evaluated, qualified and used in accordance with the requirements in Clause 15 on qualification of pre-developed software.

14.3.4.6 Tests, verification and/or validation shall be carried out to ensure that any additional code (assembly instructions) introduced by the translator which is not directly traceable to source line statements (for example, error checking code, error and exception handling code, initialisation code) is correct.

14.3.5 Application data tools

Computer-based safety systems usually require application data to define signals, addresses and function parameters of the application functions and service functions. The data can be extensive and normally consists of information such as:

– signal tag references, signal descriptions, source locations and cable numbers, measurement types, electrical ranges or states, engineering units, alarm state definitions, alarm and trip levels;
– signal termination points, data base addresses and pointers, information addresses and pointers, hardware addresses and characteristics, display layouts, display symbol and colour information, display signal content identification, log and internal message formats and details of contents;
– protection action codes, alarm priority or logic, outputs for action, identification of logic operations and timers, output states to be adopted at failure.

The data may be taken from design drawings, lists and specifications of plant operations and process instrumentation. It will be translated for loading to target processors of the system, and then used to control the action of the on-line software.

Requirements related to the preparation, verification and validation and management of data for on-line use are presented below.

14.3.5.1 The design of the software system application data and its method of derivation from the plant application data shall be defined and documented.

14.3.5.2 Application parameters that can be changed during operation by the operator shall be identified, together with the methods to be used to control changes of such parameters.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

14.3.5.3 Changes made to modifiable application data should not corrupt other data and code on the runtime system.

14.3.5.4 The formality of procedures for data verification shall be similar to the formality of procedures for software verification and validation, including identification and clearance of errors. End-to-end verification checks shall be performed, and these shall include each stage of data transformation starting from data extraction from plant design information through to incorporation of data structures in the on-line software, including the use of transfer media.

14.3.5.5 The data to be loaded to the on-line software should be in a form, which can be printed and verified, or a tool shall be used to take that data and restore it to an intelligible form for verification.

14.3.5.6 A facility shall be provided to allow verification of all loaded configuration data on site.

14.3.5.7 Where data define the interface between two systems, then the data provided for each system should be automatically generated from the same data base (see 5.3.1.4 of IEC 61513).

14.3.5.8 In some cases, where software functionality including processes, data flow, and input and output connections are controlled or modified by configuration data, specific documented justification shall confirm an adequate level of assessment and subsequent testing has been applied to the data. Extensive system re-testing may be required following changes to such data.

14.3.6 Automation of testing

Automation increases the amount of testing that can be performed in a given period. This can be achieved by meeting the following requirements.

14.3.6.1 Automated validation tools that generate test data, transport or transform test data and test results and evaluate test results should record a complete test log. This is applicable to module tests as well as to plant simulations.

14.3.6.2 Appropriate tools should be used to test and/or simulate the behaviour of the executable code loaded in the target system.

14.3.6.3 Appropriate tools shall be used to ensure or verify that the right executable code is loaded correctly in the target system.

14.3.6.4 The use of the following additional tools should be considered:

1) test generators, test coverage analysers and test drivers;
2) on-line diagnostic programs with dump inspect and trace facilities;
3) debuggers with debugging facilities at the source code level; and
4) automated test suites to facilitate regression testing.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

15 Qualification of pre-developed software

15.1 General

This Clause gives requirements for the use of pre-developed software (PDS) in I&C computer-based systems. These requirements are established as part of the qualification requirements of the system in which the PDS is integrated (see 6.4 of IEC 61513).

PDS for I&C systems may range from small software components (for example, an application function library module), to large and complex software products (for example, parts of an operating system, or communication drivers). PDS may be divided into two types with respect to hardware:

a) general-purpose PDS that has not been specifically developed for a specific hardware environment; and
b) PDS integrated in hardware components that has to be used in association with this hardware.

PDS components are called reusable when they can be used in different computer programs or systems, for example, as part of an equipment family (equipment platform). Those components which are independent of specific plant application details may have already been qualified for use in systems performing category A functions.

The specifications of new safety systems often identify the use of pre-developed equipment including PDS to implement part or the whole of a “new system” (see 6.1.2.1 of IEC 61513). Use of pre-developed equipment can be beneficial to productivity and the reliability of the system when these items are of suitable quality and introduced in a proper manner. When PDS items have been used in many applications similar to the intended use, a benefit from this operating experience can be claimed in their evaluation. In particular, the reuse of validated PDS can increase confidence in the reliability of the system.

15.2 General requirements

15.2.1 A PDS that is a candidate for use as part of a system shall comply with all the requirements developed in this standard, as any software component performing category A functions.

15.2.2 The evaluation of the PDS shall:

1) determine the capability of the PDS to meet the functional, performance and architectural requirements of the system requirements specification (see 6.1.1 of IEC 61513,), and its resulting suitability;
2) identify any modifications needed to correct or adapt the PDS;
3) assess the quality of the PDS; and
4) evaluate the operating experience of the PDS, when required for the above evaluations.

15.2.3 The conclusions of this evaluation process shall be documented.

NOTE In this standard, assessment is used to describe an action made by the organisation in charge of the computer-based system development (or on behalf of this organisation); it is neither implied nor required that assessment be made by licensing authorities, although they may also choose to do so.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

15.3 Evaluation and assessment process

The PDS evaluation and assessment process shall include:

a) an evaluation of the functional and performance features of the PDS and existing qualification documentation (see 15.3.1);

   NOTE For PDS integrated in a product these features may be expressed as product properties according to IEC 61069-2.

b) a quality evaluation of the software development process (see 15.3.2);

   NOTE For reusable pre-assessed software, only the evaluation of suitability needs to be made; the evaluation of quality is implied by its validation.

c) an evaluation of operating experience if needed to compensate for weaknesses in demonstration gained from both a) and b) (see 15.3.3); and

d) a comprehensive documented assessment of the evidence from the above evaluations, and associated complementary work, which will enable the PDS to be accepted for use in the system.

Figure 4 shows the relations between the different stages of the evaluation and assessment process of the PDS.

Figure 5 shows the relationship between this process and the qualification of the system.

NOTE The process described in this subclause is a simplified view of reality and, as such, does not show all the iterations or overlap between the evaluation activities as well as between these processes and the computer-based system development activities.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

1 Suitability evaluation (15.3.1)

Required input documentation (15.3.1.1)	
System specification documentation	PDS specification and user's documentation

Evaluation requirements (15.3.1.2)	
Comparison of the system and PDS specifications	Identification of modifications and missing points

	Conclusions	
The PDS is suitable	Complementary work is needed	Ought to be rejected

2 Quality evaluation (15.3.2)

	Req. Input documentation (15.3.2.1)	
Design documentation	Life cycle documentation	(Operating history documentation)

	Evaluation requirements (15.3.2.2)	
Analysis of design	Analysis of the QA	Identification of missing points

	Conclusions	
The quality of the PDS lifecycle is appropriate or the needed modifications of the PDS are feasible	Additional test and documentation is required or operating experience evaluation required	The PDS ought to be rejected

3 Evaluation of operating experience (15.3.3)

	Req. input documentation (15.3.3.1)	
Collection of data	Operating time	History of defects

Evaluation requirements (15.3.3.2)

	Conclusions	
Sufficient operating experience	Operating experience not sufficient yet	The PDS ought to be rejected

4 Comprehensive assessment (15.3.4)

The quality of the PDS is appropriate	The needed modifications are done

5 Integration in the system and maintenance (15.4)

IEC 718/06

Figure 4 – Outline of the qualification process of pre-developed software

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

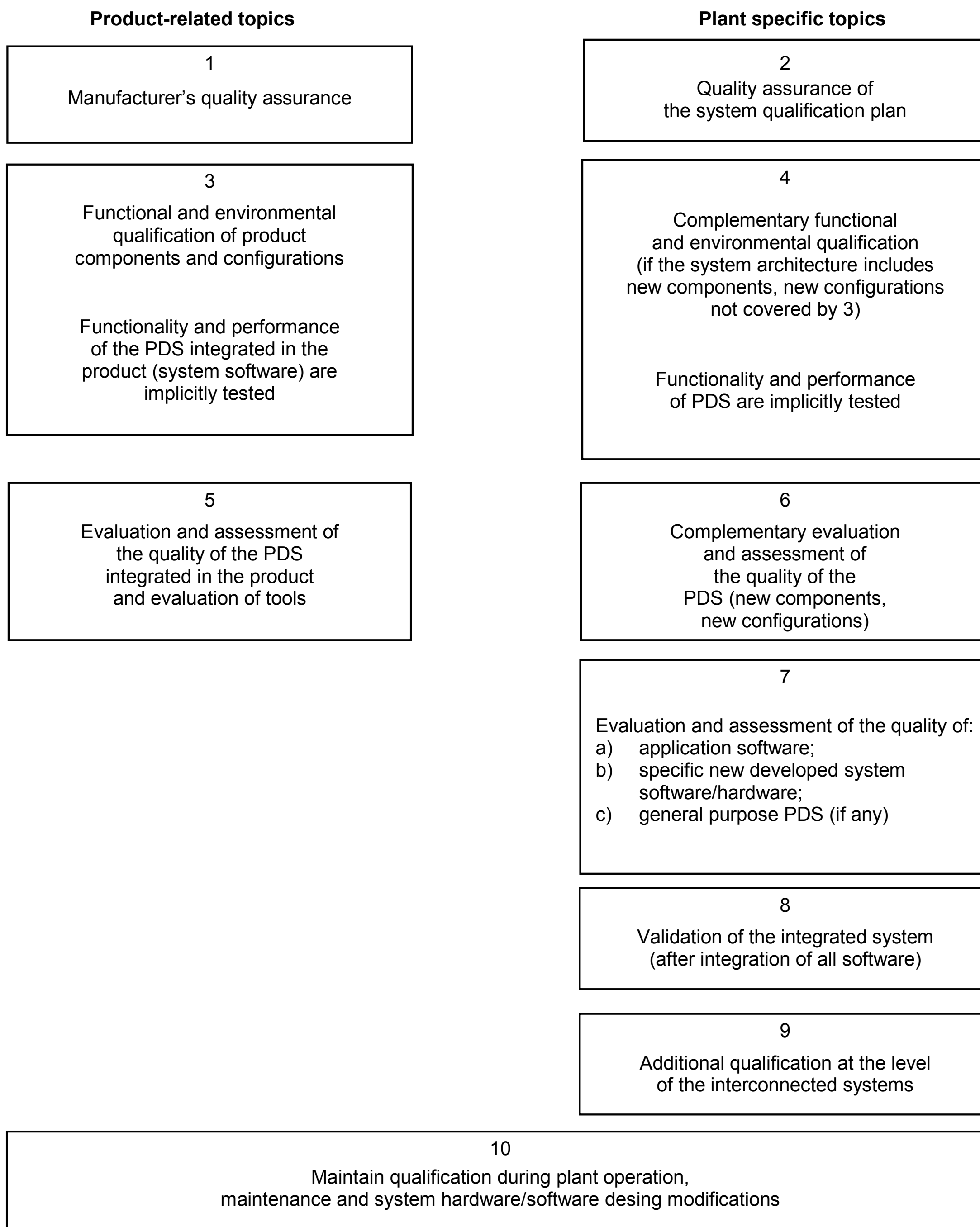


Figure 5 – Relation of PDS evaluation and assessment with the qualification plan of the system in which it is integrated

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

15.3.1 Evaluation of suitability

The objective of this process is to confirm that the functional, performance and architectural specifications of the PDS comply with the requirement specification. This process identifies components directly suitable for use in the plant system and also identifies those that will require modification.

NOTE The evaluation is based on the analysis of the specifications and functional documentation.

The evaluation of the suitability of the PDS should be initiated in an early stage of the system specification (see 6.2 of IEC 61513) and it shall be completed in order to

– assist the designers in the architectural design of the system;
– gain auditable evidence that the functionality and performance of the PDS meets the requirements of the system.

15.3.1.1 Required input documentation

15.3.1.1.1 The following documentation shall be available:

– system specification documentation which identifies the functional, interface and performance requirements, to be fulfilled by the PDS in the framework of the system architecture (see 6.1.1 and 6.1.2 of IEC 61513);
– the PDS description and user documentation. These documents should explicitly define all the characteristics that are relevant in fulfilling the system functional and performance specifications. Analysis or test shall be performed to make the implemented characteristics explicit if they are not explicitly defined.

15.3.1.1.2 The PDS shall be under configuration management; the version and configuration of the PDS shall be known precisely.

15.3.1.2 Evaluation requirements for suitability

15.3.1.2.1 The specifications of the PDS shall be evaluated with respect to the system requirements specification (see 6.1.1 of IEC 61513). If discrepancies exist, the PDS shall either be rejected or modified or the requirements specification shall be adapted to resolve them, provided that the functions important to safety are preserved.

15.3.1.2.2 If it is necessary to modify the PDS an evaluation shall be completed, based on the PDS design documentation, to determine if the change can be performed in a manner compliant with this standard. If the change cannot be performed in a compliant manner the use of the PDS shall be rejected.

NOTE The quality evaluation indicates if these modifications are feasible (see 15.3.2.2). The corresponding implementation is handled in the frame of the system lifecycle (see Clause 6 of IEC 61513).

15.3.1.2.3 For a PDS that is contained in a library, except when the whole library has to be assessed, it should be possible to tailor the library to build a restricted library meeting the software needs and to link the program with this restricted library which shall be composed of assessed components.

15.3.1.2.4 The suitability evaluation shall identify the functions that are included in the PDS that are unintended and unneeded by the system and the measures to ensure that these functions do not interfere with safety functions.

15.3.1.2.5 When the evaluation is concluded, a document shall be produced to record:

1) whether the functional and performance specifications of the PDS comply with the software requirements specification of the system; and
2) where the PDS is not adequate, the grounds for rejection.

15.3.2 Quality evaluation

The objective of this evaluation is to provide evidence that the features of the PDS design are appropriate for a system performing a category A function, and that adequate QA has been exercised throughout the lifecycle of the PDS. This evaluation is based on the design and software quality plan documentation of the PDS but may also require analysis of its operating history.

15.3.2.1 Input documentation

15.3.2.1.1 The following documentation shall be available, in addition to that required in 15.3.1.1:

– the system specification documentation which identifies the importance to safety of the functions implemented with the PDS in the architectural design of the system (see 6.1.2 of IEC 61513 and Annex A of this standard);

NOTE The level of assurance to be achieved by the quality evaluation that the PDS will perform as specified will be different for the three categories, with category A requiring the highest assurance (see 8.2.1 of IEC 61226).

– the qualification documentation of the PDS, including the documentation on previous certifications or independent assessments of the PDS, if it is to be used for the assessment.

15.3.2.1.2 The following documentation related to the PDS shall be provided or the use of alternative information shall be justified:

– the software quality plan (division into elementary tasks and associated activities) used in the software life cycle of the PDS (see Clause 5), and the corresponding quality assurance tasks and procedures records (notably verification planning);
– the specification, design, implementation and modification documents, and the corresponding verification documents;
– the software/hardware integration plan and associated verification;
– the validation plan and tests performed on the product by the vendor or customer.

NOTE For the latter two points: this documentation is needed only if the PDS is integrated in hardware components.

15.3.2.1.3 Documentation of operating experience of the PDS shall be available if it is to be used in the evaluation to compensate for lack of the above documentation or to justify use of practices differing from those of this standard.

15.3.2.1.4 The documentation should provide information on industrial factors such as the distribution of the PDS and the customer support.

15.3.2.2 Evaluation requirements for quality

15.3.2.2.1 The requirements of the PDS software quality plan and the corresponding verification and documentation shall be evaluated for conformance with the requirements of this standard. This analysis of conformance needs interpretation to identify the requirements which are applicable in the context of the use of the PDS in the system.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

15.3.2.2.2 The PDS design shall be consistent with the constraints on the architecture and deterministic internal behaviour of the system.

15.3.2.2.3 If practices differing from those of the annexes of this standard have been used for the development of the PDS, their adequacy shall be analysed and justified according to 5.5 of this standard. Their importance in the assurance of the software quality characteristics shall be evaluated in conjunction with the system requirements. The results of the evaluation and analysis shall be recorded for independent review.

15.3.2.2.4 Non-conformities against requirements of this standard, properties that cannot be verified, weakness or missing steps in the verification or documentation process shall be identified. Each shall be ranked according to its importance in the assurance of the software quality characteristics, and the importance to safety of the functions implemented in the system. Clause I.1 provides guidance for the ranking of non-conformities.

15.3.2.2.5 If systems implementing category A functions include functions of a lower category that are to be performed by a PDS, and the system architectural design is such that this PDS could potentially jeopardise the category A functions (see 6.2 of IEC 61513), then the evaluation criteria for PDS software implementing category A functions shall be applied to such PDS.

15.3.2.2.6 The qualification documentation shall provide evidence that PDS integrated in hardware components, has been validated to demonstrate that it meets its functional and performance specifications.

NOTE The functional and performance behaviour of the PDS components may be implicitly qualified by the functional qualification of the individual equipment in which they are integrated (see 2 of Figure 5). However, there are properties that may only be qualified using configurations of equipment.

15.3.2.2.7 Where PDS components contain features that cannot be validated other than in the final system configuration, then validation of these features shall be performed in the final system configuration.

15.3.2.2.8 The quality and degree of coverage of the validation tests performed on the PDS shall be evaluated with reference to the requirements of Clauses 9 and 10 of this standard and additional validation tests performed if necessary.

15.3.2.2.9 When the evaluation of the design and of the lifecycle is concluded, a document shall be produced to record that at least one of the following conclusions is true:

1) the PDS quality has been proved and no additional tests or analysis of operating experience is required;
2) complementary qualification shall be performed when the system configuration is available;
3) lack of information has been detected during the evaluation, but this can be compensated by the completion of additional verification and validation, testing or code analysis and documentation;
4) lack of information has been detected during the evaluation, which can be compensated for by use of operating experience;
5) the PDS (or part of the PDS) requires modification for the intended use in the system (see 15.3.3), and that it has the appropriate level of quality so it is desirable, but not essential, that the modifications be performed in accordance with this standard;

NOTE After satisfactory completion of the defined modifications, a complementary evaluation and assessment is needed in the frame of the system qualification (see 5 of Figure 5).

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

6) significant problems can be expected because of the transfer of the PDS to new hardware;
7) the PDS quality is not adequate and the PDS shall be rejected on grounds that the weaknesses are too great or the information inadequate for effective compensation; and
8) the independence of the qualified functions/properties of the PDS from those not qualified has/has not been established.

15.3.3 Evaluation of operating experience

The objective of this evaluation is to provide evidence that suitable operating experience of the PDS may complement the confidence in the PDS in case of deficiencies detected in the quality evaluation.

Those functions/properties of the PDS whose feedback of experience has to be evaluated, shall be identified and the following shall be evaluated:

1) the methods for collection of data on operating experience;
2) the methods for recording the PDS version operating time and for producing the operating history;
3) the operational history of findings, defects and error reports; and
4) the operational history of modifications made for defects or other reasons.

15.3.3.1 Validation of input data and methods for producing the operating history

15.3.3.1.1 The evaluation of operating experience of the PDS is based on data available from the vendor and, if possible, from users of systems running the PDS. In order to consider the operating experience suitable for the evaluation of the PDS, the methods for collection of data and producing the operating history shall be assessed.

15.3.3.1.2 Only information that is derived from a well-defined and controlled data collection process shall be used.

15.3.3.1.3 Collection procedures shall be assessed to validate the data for completeness and credibility. Clause I.2 gives guidance for collection and validation of data.

15.3.3.1.4 Operating experience shall be considered valid only if it is observed under conditions similar to the conditions during intended operation.

15.3.3.1.5 The accumulated operating time for the PDS under evaluation shall be established. It may be calculated by adding together the operating time of each installation for which operational experience has been collected and validated. Time in which the PDS was not operating in a representative manner should be excluded.

15.3.3.1.6 The operating times shall be for the same version of the PDS that is intended to be used. When operating time of other versions is included, an analysis of the differences and history of these versions shall be made. This analysis shall identify the parts and functions of the PDS which differ, and the parts and functions which are not affected by the modifications to demonstrate that the history is valid.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

15.3.3.1.7 Problems, failures and their correction in the different versions of the PDS shall be analysed and classified according to their severity. Their impact on the intended functions shall be evaluated.

15.3.3.1.8 No qualified function shall be affected by errors found or modifications made to any other functions of the same PDS.

15.3.3.1.9 Evaluation of communication functions should take account of operating experience. The service limits should be identified and compared with the forecasts for normal, peak load and equipment failure conditions.

15.3.3.1.10 Operating experience should be considered as suitable when the following criteria are met:

1) the PDS has achieved a sufficient accumulated operating time (see Annex I);

   NOTE The sufficient operating time should be determined on a case-by-case decision based on engineering judgement. This judgement should take into account notably the anticipated reliability level required at system level for the functions in which the PDS is used.

2) no significant modifications have been made and no errors are detected over a significant operating time on several sites or applications; and
3) the PDS has preferably operated on several installations.

15.3.3.1.11 When the evaluation of the operating experience is completed, a document shall be produced to record either:

1) that the operating experience is suitable for the identified functions/properties of the PDS and the justifications why the operating experience may be used to support assessment of the quality; or
2) that the operating experience is not suitable or is not sufficiently proved.

15.3.3.2 Acceptance criteria to be applied when using operating experience as a compensating factor

Suitable operating experience may be used as a compensating factor for acceptance of the PDS, when the following criteria are met.

15.3.3.2.1 The evaluation of the feedback of operating experience shall never completely replace the evaluation of the design of the product itself and of its documentation (see 15.3.2.2).

15.3.3.2.2 Suitable operating experience shall be accepted as part of the justification only if it is used to compensate for weaknesses in the assessment of a PDS against the design recommendations of item c) of Clause B.2 of this standard on operating systems and standardised programs.

15.3.3.2.3 The operational history shall be accepted as part of the justification only if it demonstrates that no suspected or known defects are able to prevent operation of a category A function or cause it to act incorrectly in the system.

15.3.3.2.4 The rigour of the analysis on feedback of experience shall be consistent with the safety category of the system functions, and the evidence of technical correctness provided by this analysis should be consistent with that achievable when applying Clauses 1 to 12 of this standard.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

15.3.3.2.5 An evaluation document shall be produced to record either:

1) how satisfactory feedback of experience compensates for any weakness identified in the evaluation of the design and life cycle of the PDS; or
2) that the use of the PDS is rejected because the results of the evaluation are negative, or because the operating experience is not yet sufficient to compensate the weaknesses identified in the development.

15.3.4 Comprehensive assessment

15.3.4.1 When the evaluations and all the necessary complementary work (modifications of the PDS, complementary tests, complementary documentation) have been completed, a comprehensive justification document for the use of the PDS in the implementation of the system shall be prepared.

15.3.4.2 This document, based on the conclusions derived from the evaluations described in 15.3.1, 15.3.2 and 15.3.3, shall record the assessment that demonstrates that the PDS (or part of the PDS) is suitable and has the level of quality appropriate for its intended use in the system and no further modification is needed.

15.4 Requirements for integration in the system and modification of PDS

15.4.1 After the comprehensive assessment, the decision to use the PDS shall be formally made and documented within the project following a formal design review.

15.4.2 The procedures for integration of the PDS shall be described in the system quality assurance plan and system integration plan (see 6.2.1 and 6.2.3 of IEC 61513).

15.4.3 After acceptance, the PDS shall be placed under the system configuration management (see 6.2.1.2 of IEC 61513) and only the release subjected to the qualification described, and any necessary modifications identified by the process shall be used.

15.4.4 The quality plan of the system shall provide procedures for upgrading the PDS when the use of a new release becomes necessary.

15.4.5 Information on errors and failures due to the PDS on other sites and applications, and on corresponding modifications of the PDS, should continue to be accessed and formally analysed during the subsequent life.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Annex A
(normative)

Software safety life cycle and details of software requirements

A.1 Software safety life cycle

The software development process is illustrated by Figure 3, showing the relationship of the life cycle activities from specification through design and implementation to verification and validation.

The details of software requirements are described in Clause A.2.

A.2 Details of software requirements

A.2.1 Description of constraints between hardware and software

A.2.1.1 The following items shall be described:

- operating characteristics in general (word length, exchange types, speed, etc.); in many cases a reference to the equipment manufacturer manuals is sufficient;
- special equipment operating characteristics (particular drivers, data-communications equipment, etc.);
- arithmetic constraints;
- requirements of standard software packages;
- requirements of hardware self-supervision;
- at least a reference to the hardware requirements document shall be made.

A.2.1.2 The reciprocal requirements between hardware and software shall be determined with respect to failure detection and reaction on failure indications.

The criteria established in the IAEA guide NS-G-1.3 require no additional amplification for computer-based system hardware. However, documentation of all hardware requirements which impact software is necessary to provide the basis for hardware/software integration and computer-based system validation.

A.2.1.3 The interface between functions of different levels of safety significance (e.g. category A and non-category A functions) performed by the software shall be described.

A.2.2 Self-supervision

A.2.2.1 Appropriate automated actions should be taken at failures considering the following factors:

- failures shall be identified to a reasonable degree of detail;
- fail-safe output shall be guaranteed as far as possible;
- if such a guarantee cannot be given, system output shall violate only less essential safety requirements;

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

– the consequences of failures shall be minimised;
– remedial routines, such as fall back, system recovery, should be considered for inclusion;
– sufficiently detailed information on failures shall be provided for the operating staff.

A.2.2.2 The system shall be designed in such a way that appropriate self-supervision is feasible. Design principles used to assist this include:

– modularisation;
– intermediate plausibility checks;
– use of redundancy and diversity; diversity may be implemented as functional diversity or software diversity;
– provision of sufficient execution time and memory space for self-checking purposes;
– failure simulation may be used to confirm the adequacy of self-supervision features.

A.2.2.3 Self-supervision shall not prevent timely system reactions in any circumstance.

A.2.3 Presentation of software specifications

A.2.3.1 The software specifications shall be presented in a manner which is easy to understand for all user groups.

A.2.3.2 The presentation shall be detailed sufficiently, free from contradiction, and without duplication as far as possible.

A.2.3.3 The document shall be free of implementation details, complete, consistent and up-to-date.

A.2.3.4 The software specifications document shall distinguish clearly between essential requirements and less stringent targets.

The software specifications document is intended to be used by

– its authors;
– the customer, client or final user;
– the software development team;
– the software verification team;
– assessors and licensing personnel.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Annex B
(normative)

Detailed requirements and recommendations for design and implementation

Annex B provides a set of tables, each table starting with a general requirement (shall) such as B1.a, followed by a list of associated recommendations (should), such as B1.aa.

For each requirement, projects should select the recommendations to be met by the design and implementation. Recommendations not chosen should be justified (e.g not applicable, or covered by another measure, etc.).

The requirements and recommendations are listed in the following tables.

B.1 Design process

B1.a Modifiability

Item	Recommendation	Good against/Good for
B1.*a*	The design process shall make the software easily modifiable	Risk of introducing faults when implementing changes /
B1.*aa*	Characteristics of the software to be developed and its functional requirements, which are likely to change during its life cycle, should be identified at an early design stage	/ Achieving safe and cost effective flexibility
B1.*ab*	Modules should be chosen so that impact on the software of anticipated changes is limited for further design stages	/ Minimise the risk of faults due to changes
B1.*ac*	Modifiability should be carefully counter- balanced with resulting overhead in complexity, run time and memory space	Systems being too complex/

B1.b Top-down approach

Item	Recommendation	Good against/Good for
B1.*b*	A top-down approach shall be used in software design	Design errors/Completeness of the design
B1.*ba*	General aspects should precede specific ones	Risks of inconsistency/Allows developers to work logically from requirements through to final design
B1.*bb*	At each level of design, the whole system should be completely described and verified	/ To ensure consistency and completeness of the design, to find areas of difficulty early

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Item	Recommendation	Good against/Good for
B1.*bc*	Areas of difficulty should be identified as far as possible at an early stage in the design process	/ Potential issues to be an input to design decisions
B1.*bd*	Key design decisions should be documented and reviewed at an early stage of the design process	/ To assess he feasibility of the design as early as possible. To reduce the possibility of changes being required later in the software development.
B1.*be*	In case of a major decision affecting other system parts, the alternatives should be considered and their risks documented	Duplicating work/Careful design
B1.*bf*	Consequences of individual decisions for other system parts should be identified.	Duplicating work/Careful design
B1.*bg*	The gap between software design levels should be such that reviewers can understand each design level in reference to the former level.	Designs difficult to understand/To ensure software design clarity and allows the consistency of the design to be verified.
B1.*bh*	The software design should proceed using one or more higher level descriptive formalisations (where appropriate and effective), such as used in mathematical logic, set theory, pseudo code, decision tables, logic diagrams or other graphic aids or application oriented languages	Misinterpretation or inaccuracy /
B1.*bi*	Automated development aids should be used	/ To reduce the scope for human error
B1.*bj*	Documentation should be such that the specifier is able to understand and check the implementation of the functions in the design.	/ Modification and consistency with specifications

B1.c Verification of intermediate design products

Item	Recommendation	Good against/Good for
B1.*c*	The intermediate design products shall be verified	/ Finding errors as soon as possible, completeness of the design
B1.*ca*	It should be shown that each level of design is complete and consistent in itself	Necessity for later changes /
B1.*cb*	It should be shown that each level of design is consistent with the previous level and with the software requirements specification that is relevant for that design level	Forgetting aspects/To ensure that no requirement has been omitted
B1.*cc*	Consistency checks should be made by personnel not involved in the design process	
B1.*cd*	These personnel should only mark possible faults and not recommend any action	Personal identification with the program/Maintaining critical attitude.
B1.*ce*	They should provide explicit rationale when relevant to help designers correctly understand their findings	/ Increase interest in verification activities and global team efficiency

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

B1.d Modification control during the development

Item	Recommendation	Good against/Good for
B1.*d*	Changes which are necessary during program development shall begin at the earliest design stage which is still relevant to the change	Introducing new faults due to changes/Avoiding hidden far distant effects
B1.*da*	Changes should proceed from the general stage to the more specific stages	/ Recognizing all effects of the change
B1.*db*	If any module is changed, it should be re-tested according to the principles described earlier (see subclause 11.2), before it is reintegrated into the system	Hidden faults in module due to change /
B1.*dc*	In addition, related calling and called modules should also be re-tested, as far as they are affected by the change.	Hidden faults in module environment due to change /
B1.*dd*	Documentation of major changes should include the requirements, code parts, data areas, control flow characteristics and time aspect that were affected	/ Traceability of change effects
B1.*de*	Changes affecting already tested parts or the work of other persons should be evaluated and reviewed prior to incorporation	/ Hidden far distant effects
B1.*de.1*	NOTE This procedure is valid for changes that affect the work of only one person and for modifications that affect the whole system. In the latter case the recommendations of Clause 11 apply additionally.	

B.2 Software structure

B2.a Control and access structures

Item	Recommendation	Good against/Good for
B2.*a*	Programs and program parts shall be grouped systematically	/ Aiding assessment and modification
B2.*aa*	Specific system operations should be performed by specific parts	/ Testability

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Item	Recommendation	Good against/Good for
B2.*ab*	The software should be partitioned so that aspects which handle such functions as: – computer external interfaces (e.g. devices driving, interrupt handling); – real time signals (e.g. clock); – parallel processing (e.g. scheduler); – store allocation; – special functions (e.g. utilities); – mapping standard "functions" onto the particular computer hardware; are separated from application programs with well-defined interfaces between them	/ Aiding testability, clarity of design
B2.*ac*	The program structure should permit the implementation of anticipated changes with a minimum of effort (see also Table B1.a)	/ System adaptability
B2.*ad*	The structuring methods being used should be clearly stated	/ Understandability
B2.*ae*	In one processor, the computer program should work sequentially, as far as possible	Confusion due to timing problems or different interrupt sequences/
B2.*af*	Computer programs should have an explicit and clear modular structure	/ Understandability, testability

B2.b Modules

Item	Recommendation	Good against/Good for
B2.*b*	Modules shall be clear and intelligible	/ Understandability
B2.*ba*	Each module should correspond to a specific function	/ Testability
B2.*bb*	A module should have only one entry. Although multiple exits may be sometimes necessary, single exits are recommended	/ Ease of verification
B2.*bc*	The size of modules should not exceed a limit of executable statements. This limit is to be specified for the particular system and longer modules should be allowed only under special circumstances.	Bulky modules /
B2.*bd*	The interfaces between modules should be as simple as possible, uniform throughout the system and fully documented	Interface faults /
B2.*be*	The number of module interface parameters should be minimized	Interface faults /
B2.*bf*	The status of module interface parameters (i.e. input, output or input/output) should be clearly stated.	Interface faults /

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

B2.c Operational system software

Item	Recommendation	Good against/Good for
B2.*c*	Operational system software use shall be restricted	Failures due to operating system errors/Ease of system verification
B2.*ca*	Only thoroughly verified operational system software with its associated verification documentation should be used; if operational system software is a PDS, refer to Clause 15	Hidden errors /
B2.*cb*	The use of general purpose operational system software (operating system) should be avoided	Use of overly complex software products /
B2.*cc*	If a general purpose operational system software is considered necessary, its use should be restricted to a few simple functions	/ Demonstrating use of a thoroughly tested operational system software
B2.*cd*	It should contain only the necessary functions	Dead code /
B2.*ce*	Operational system software functions should be called always in a similar way	/ Ease of verification
B2.*cf*	Functions used to control hardware should be taken from the operational system software or developed and verified within it	Additional programming and test effort /
B2.*cg*	Operational system software functions should be rigorously defined and should have well-defined interfaces	Errors in using the functions /
B2.*ch*	The conditions of use and the interdependences of operational system software functions should be known and checked.	Errors in using the functions /
B2.*ci*	If a dedicated operational system software or a dedicated part is developed for a special safety application, the whole standard shall apply	/ Ease of verification

B2.d Execution time

Item	Recommendation	Good against/Good for
B2.*d*	Physical process behaviour influence on execution time shall be kept low	Unintelligible timing problems /
B2.*da*	The execution time of any system or part of a system under peak load conditions should be short compared to the execution time beyond which the system safety requirements are violated	Necessity of late design changes /
B2.*db*	The results produced by the computer program shall not be dependent on either: – the time taken to execute the program, or – the time (referenced to an independent "clock") at which execution of the programs is initiated	Unintelligible timing problems/Determinism
B2.*dc*	The computer programs should be designed so that operations are performed in a correct sequence independent of their speed of execution	Synchronization problems and run time problems/Analysis

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Item	Recommendation	Good against/Good for
B2.*dd*	Runtime should not vary significantly in response to changes in input data	/ Ease of run time estimation and run time verification
B2.*de*	The extent of runtime variation which may be caused by the inputs should be documented.	/ Ease of run time estimation and run time verification
B2.*df*	Code parts for which execution time depends on input data should be short	/ Reaching B2.*de*
B2.*dg*	The amount of data read during one computation cycle should be constant	/ Keeping run time differences short
B2.*dh*	Programme execution time should not be coupled to receipt of data	Synchronization problems and run time problems/Analysis

B2.e Interrupts

Item	Recommendation	Good against/Good for
B2.*e*	The use of interrupts shall be restricted	Synchronization problems and run time problems/Analysis
B2.*ea*	Interrupts may be used if they simplify the design of the software and do not make the verification overly complex.	/ Ease of understanding of special configurations
B2.*eb*	Critical software parts (e.g. time critical, critical to data changes) should be identified and protected.	Synchronization problems and run time problems/Analysis
B2.*ec*	The software may inhibit the handling of interrupts during critical parts. Such inhibitions should be justified.	Synchronization problems and run time problems/Analysis
B2.*ed*	If interrupts are used, parts not interruptible should have a defined maximum computation time, so that the maximum time for which an interrupt is inhibited can be calculated	Run time problems /
B2.*ee*	Interrupts usage and masking shall be thoroughly documented	/ System validation

B2.f Arithmetic expressions

Item	Recommendation	Good against/Good for
B2.*f*	Where possible simple arithmetic expressions should be used instead of complex ones.	Side effects/Easy testing
B2.*fa*	Decisions should not depend on voluminous arithmetic calculations	/ Finding reverse function calculating path expressions
B2.*fb*	As far as possible, simplified previously verified arithmetic expressions should be used	/ Showing relationship between intended function and code
B2.*fc*	If voluminous arithmetic expressions are used, they should be coded so that their consistency with the specified arithmetic expression can be shown easily from the code.	/ Program analysis

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

B.3 Self-supervision

B3.a Plausibility checks

Item	Recommendation	Good against/Good for
B3.*a*	Plausibility checks shall be performed (defensive programming)	Potential residual faults/
B3.*aa*	The correctness or plausibility for intermediate results should be checked periodically.	Potential residual faults /
B3.*ab*	The ranges of: - input variables; - output variables; - intermediate parameters should be checked, including array bound checking	Potential residual faults /
B3.*ac*	Failed plausibility checks should be reported.	/ Diagnostic of potential residual faults

B3.b Safe output

Item	Recommendation	Good against/Good for
B3.*b*	If a failure is detected, the system shall produce a well-defined output	Fault propagation/Failsafe behaviour
B3.*ba*	If possible, complete and correct error recovery techniques should be used	System breakdown due to minor failures /
B3.*bb*	If error recovery techniques are used the occurrence of any error shall be reported	Accumulation of faults/Early fault removal
B3.*bc*	The occurrence of a persistent error affecting the system function shall be recorded	Accumulation of faults/Early fault removal

B3.c Memory contents

Item	Recommendation	Good against/Good for
B3.*c*	Memory contents shall be protected or monitored NOTE Protection may be viewed as being either prevention (prevent untimely modification from occurring) or detection (detect an inappropriate state – memory corruption) with appropriate and quick reaction (error confinement or correction). Detection presupposes monitoring (or its synonym: supervision)	Unauthorized changes, potential residual faults /
B3.*ca*	Memory space for constants and instructions shall be write protected or supervised against changes	Propagation of addressing faults or hardware faults, including intermittent faults /
B3.*cb*	Unauthorised reading and writing should be prevented	
B3.*cc*	The system should be secure against code or unauthorised data changes by the plant operator	/Maintaining the software in its licensed form

B3.d Error checking

Item	Recommendation	Good against/Good for
B3.*d*	Error checking shall be performed by the code	Failure propagation /
B3.*da*	Counters and reasonableness traps should ensure that the program structure has been run through correctly	Specific control flow, intermittent hardware faults /
B3.*db*	The correctness of any kind of parameter transfer should be checked, including parameters type verification	Faults in the design of interface and data flow/
B3.*dc*	When addressing an array its bounds should be checked	Data flow faults, too high loop repetition numbers /
B3.*dd*	The run time of critical parts should be monitored (e.g. by a watchdog timer)	Faults in the design of control flow, too high loop repetition numbers /
B3.*de*	Assertions should be used (e.g. in a triangle, if NOT (a+b > c) then Error)	/ Plausibility of intermediate results

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

B.4 Detailed design and coding

B4.a Branches and loops

Item	Recommendation	Good against/Good for
B4.*a*	Branches and loops should be handled cautiously	/ Understandable and verifiable control flow
B4.*aa*	Backward going branches should be avoided, loop statements should be used instead (only for higher level languages)	Difficulties with control flow analysis/readability
B4.*ab*	Branches into loops, modules or subroutines should be barred	Difficulties with control flow analysis/readability
B4.*ac*	Branches out of loops should be avoided, if they do not lead exactly to the end of the loop. Exception: failure exit	Difficulties with control flow analysis/readability
B4.*ad*	In modules with complex structure, macros, or subroutines should be used so that structure stands out clearly	Difficulties with control flow analysis/readability
B4.*ae*	As a special measure to support program proving and verification computed GOTO statements as well as label variables should be avoided.	
B4.*af*	Where a list of alternative branches or case controlled statements are used, the list of branch or case conditions should be an exhaustive list of possibilities. The concept of a "default branch" should be reserved for failure handling	/ Clarifying exclusive "or"
B4.*ag*	Loops should only be used with constant maximum loop variable ranges	Run time problems, violating array boundaries/Surveyable path number

B4.b Subroutines

Item	Recommendation	Good against/Good for
B4.*b*	Subroutines should be organised as simply as possible	Unnecessary complexity /
B4.*ba*	They should have only a predefined maximum number of parameters	/ Keeping routines and interfaces short and simple
B4.*bb*	They should communicate exclusively via their parameters to their environment	/ Understandability of data flow, data flow analysis
B4.*bc*	Subroutines should have only one entry point	/ Understandability of control flow, control flow analysis
B4.*bd*	Subroutines should return to only one point for each subroutine call. Exception: default exit	/ Understandability of control flow, control flow analysis
B4.*be*	The return point should immediately follow the point of call	/ Understandability of control flow, control flow analysis

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

B4.c Nested structures

Item	Recommendation	Good against/Good for
B4.*c*	Nested structures shall be handled with care	/ Intelligibility
B4.*ca*	Nested macros should be avoided	Producing overly complex code /
B4.*cb*	Joining of different types of program action by means of nested loop statement or nested subroutines should be avoided, if they obscure the relationship between problem structure and program structure	/ Favouring sequential structures
B4.*cc*	Hierarchies of subroutines and loops should be used, if they clarify the program structure	/ Indicating different levels of abstraction during top-down design

B4.d Addressing and arrays

Item	Recommendation	Good against/Good for
B4.*d*	Simple addressing techniques shall be used	/ Ease of data flow analysis
B4.*da*	Only one addressing technique should be used for each data type	/ Uniform interface to data base
B4.*db*	Bulky computations of indexes should be avoided	/ Understandable data flow
B4.*dc*	Arrays should have a fixed, predefined length	Run time difficulties, complex control flow/Ease of data flow analysis
B4.*dd*	The number of dimensions in every array reference should equal the number of dimension in its corresponding declaration	/ Understanding addressing process

B4.e Data structures

Item	Recommendation	Good against/Good for
B4.*e*	Data structures and naming conventions shall be used uniformly throughout the system	/ Data flow analysis, intuitive comprehension of the significance of data elements
B4.*ea*	Variables, arrays and memory cells should have a single purpose and structure. The use of equivalence techniques should be avoided	Errors in data use, artificial timing problems/Traceability of data flow
B4.*eb*	Each variable's name should reflect its use	/ Intuitive comprehension of data element
B4.*ec*	Constants and variables should be located in different parts of the memory	Poisoning of data and code/Hardware self-+ supervision
B4.*ed*	When a universally accessible “data base” or similar resource is used as global data structures, they should be accessed via standard resource handling subroutines or via communication with standard resource manipulating tasks.	
B4.*ee*	Programs which receive or transmit data from/to other programs should exchange consistent data sets	Incoherence of data /

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

B4.f Dynamic changes

Item	Recommendation	Good against/Good for
B4.*f*	Dynamic changes to executable code shall be avoided	/ Control flow analysis

B4.g Unit and integration tests

Item	Recommendation	Good against/Good for
B4.*g*	Unit and integration tests shall be performed during the program development	/ Finding faults as soon as possible
B4.*ga*	The approach to testing should follow the approach to design (e.g. during top down design testing should be made by using simulation of not yet existing system parts – stubs –; after completion of system design and implementation this should be followed by bottom up integration testing)	/ Finding faults as soon as possible
B4.*gb*	Each module should be tested thoroughly before it is integrated into the system and the test results documented	Difficulties after integration/Change management
B4.*gc*	A formal description of the test inputs and results (test protocol) should be produced	Duplication of work/Speeding up licensing
B4.*gd*	Faults which are detected during program testing should be recorded and analysed	/ Detection of some design faults
B4.*ge*	Incomplete testing should be recorded	/ Clarity
B4.*gf*	In order to facilitate the use of unit and integration test results during final validation, the former degree of testing achieved should be recorded (e.g. all paths through the module tested)	Duplication of work /

B.5 Language dependent recommendations

B5.a Sequences and arrangements

Item	Recommendation	Good against/Good for
B5.*a*	Detailed rules shall be elaborated for the arrangement of various language constructs	/ Getting a uniform and intelligible shape of program listings
B5.*aa*	The recommendations should include sequence of declarations, including parameter types	
B5.*ab*	Sequence of initialisations	
B5.*ac*	Sequence of non-executable code/executable code	
B5.*ad*	Sequence of formats declaration (e.g for languages such as FORTRAN)	

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

B5.b Comments

Item	Recommendation	Good against/Good for
B5.*b*	Relations between comments and the code shall be fixed by detailed rules	Difficulties with both writing and understanding comments/Getting meaningful comments
B5.*ba*	It should be made clear what has to be commented	
B5.*bb*	The position of comments should be uniform	
B5.*bc*	Form and style of comments should be uniform	

B5.c Assembler

Item	Recommendation	Good against/Good for
B5.*c*	If an assembly language is used, extended and documented coding rules shall be followed	Difficulties of assembler programming, tricks/Simplicity and understandability
B5.*ca*	Branching instructions using address substitution should not be used. Branch table contents should be constant	Branches whose goal cannot be identified from the code at the branching point/Improve testability and clarity
B5.*cb*	All indirect addressing should follow the same scheme	/ Clarity of ultimately addressed memory locations
B5.*cc*	Indirect shifting should be avoided	/ Seeing immediately how far shift goes
B5.*cd*	Multiple substitutions or multiple indexing within a single machine instruction should be avoided	/ Understanding addressed location
B5.*ce*	The same macro should always be called with the same number of parameters	/ Understanding the macro's function
B5.*cf*	Labels (named locations) should be used to make references within the code. Numerical values (either absolute addresses or relative offsets) should be avoided	/ Associating a meaning with the branching goal
B5.*cg*	Subroutine call conventions should be uniform throughout the coding and specified by further rules	Arbitrary parameters types and locations /

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

B5.d Coding rules

Item	Recommendation	Good against/Good for
B5.*d*	Detailed coding rules shall be issued	/ Improve clarity and consistency of code
B5.*da*	It should be made clear, where and how code lines are to be indented	/ Identification of blocks
B5.*db*	Module layout should be fixed uniformly	/ Understanding module structure
B5.*dc*	Further details should be regulated according to need	

B5.e Application oriented languages

Item	Recommendation	Good against/Good for
B5.*e*	Application oriented languages should be used rather than machine-oriented languages	/ Understandability, simplicity
B5.*ea*	Functions or part of functions not achievable with application oriented languages should be developed as independent modules	
B5.*eb*	Application oriented languages with a graphical syntax should offer an associated literal language	/ Use of automated analysis tools
B5.*ec*	Any features of application oriented languages not appropriate for a class 1 system development should be identified and documented and finally avoided	

B5.f Automated code generation

Item	Recommendation	Good against/Good for
B5.*fa*	The output of the code generator should be traceable to the input	/ Ability to verify the code generator output during qualification or use
B5.*fb*	The generated code should be readable	/ Finding faults due to the code generator
B5.*fc*	No changes shall be made at generated code level. If changes are necessary, they shall be introduced and documented at input level	/ Keeping consistency of output with input /Assurance of software quality of the generated code
B5.*fd*	The language used for the generated code should comply with the recommendations of Annex D, where applicable	/ Availability of software tools for compilation and V&V, readability

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Annex C
(informative)

Example of application oriented software engineering (software development with application-oriented language)

Application oriented software engineering has taken a rapid development since the first edition of IEC 60880 in 1986.

Equipment families (system platforms) for industrial automation tasks are now widely available on the market.

Powerful engineering tools, which are developed for economical reasons as well as for quality assurance, are an integral part of these platforms.

A typical aspect of the application oriented software is that the most challenging parts of the traditional software development process are performed automatically.

This annex gives an example of how the requirements of this standard may be implemented in the context of a typical software development with an application-oriented language.

C.1 Principle of application of requirements

Software quality is an essential part to ensure the overall quality and safety of computer-based I&C systems important to safety.

Basic factors to ensure compliance with the requirements of this standard are:

a) a software development process structured into phases with clear input and output definitions (see 5.4);

b) an understandable software structure developed during the software design phase which forms the basis for the coding as well as for software assessment (see 7.1);

c) coding rules and practices in accordance with the requirements and recommendations given in Annex B (see 7.3.2);

d) traceability of input requirements down to the final software code (see 7.4).

In the case of this example, the software development process is strongly determined by software tools. Nevertheless, it has been structured into phases with clear input and output definitions.

The code generator has been designed to comply with engineering requirements related to the software design. In consequence, the generated code has a structure which ensures clear assignment of the code to the specified functions.

In the same way, coding rules have been implemented in the code generator so that the generated code meet the requirements and recommendations given in Annex B of this standard.

The tool set applied supports traceability of input requirements during all phases of the development process up to the executable code integrated into the target system.

标准分享网 www.bzfxw.com 免费下载

C.2 Application of requirements for software life cycle

The use of application-oriented languages supported by tools for automated code generation influences significantly the software life cycle.

Figure C.1 below shows an example of life cycle for application oriented software development. In comparison to the classical software life cycle, the phases for detailed software design, software coding, module integration and testing are integrated into automated processes.

During project execution the activities related to specification and parts of the system validation are typically performed by process engineers whereas the activities for I&C system design, functional specification and system testing are performed by I&C engineers.

Application software can be generated immediately after the specification of I&C system and related functions has been set up.

Hence, the possibility exists to assess the specified functions by means of the generated code against a simulator or specific input data trajectories.

This functional assessment which can be performed even before the hardware of the target system is manufactured may improve the design quality by allowing early detection of design faults.

The software tool set supports the whole software life cycle i.e. specification, design including verification and validation, system operation and modifications in case of later changes of the input requirements.

The formats to document the specification of the system design and the related functions support effective verification. This feature of the application-oriented software engineering complies with the requirements of Clauses 8 to 10 of this standard.

C.3 Application of requirements for automated code generation

Automated code generation is a typical feature of state of the art equipment families (system platforms) to ensure efficient design of I&C systems as well as software production on the highest quality level.

Such equipment families include a tool set to support the software engineering process.

The methodology of automated code generation provides software of high quality when complying with this standard and reduces respectably the potential for the introduction of human error.

C.3.1 Tools for design and project management

In the example denoted here, the software-based tools generate the application source code from a formalised specification (e.g. graphical specification) in a language which has been carefully chosen:

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

a) the use of application-oriented notations for specification of I&C systems architecture and associated functions help solving interface between system designers and I&C designers;
b) the input notations have been based on specified graphical representations for the intended I&C functions comparable to the classical proven function diagrams and enable comprehensive checking of the functions specified;
c) the documentation produced allows strict correlation to the development results.

The set of software tools provides assistance for the following software engineering activities:

- possibility to specify the hardware architecture in accordance with the required fault tolerance which leads to the necessary redundancy structure;
- demonstration of syntactic correctness for the software specification by the use of integrated check tools;
- assistance for an effective configuration management by unmistakable identification of all software components (e. g. by CRC check sum);
- assistance for verification to ensure correct engineering related to each design phase;
- assistance for verification of the design by means of simulation;
- assistance for diagnosis and inspection of the software processed on the target system by means of service tools;
- assistance for I&C systems design by means of tools to evaluate worst case processor load and worst case bus load;
- assistance for the administration of design data by a database for all relevant design data.

C.3.2 Tools for automated generation

The software tools for automated code generation of the example include the following features:

a) the automated code generation covers the full scope of application functions, application data and the communication between all the processing units of the class 1 system;
b) the input notations for the automated code generation have been specified (syntax and semantics);
c) the application functions are specified by functional diagrams;
d) the tools allow the assignment of the designed functions to dedicated processing units;
e) proven software design rules integrated in the tool have been implemented to ensure:
   - clear structure of the generated source code to meet the general requirements stated in this standard;
   - program control flow is independent from the sequence of inputs to the graphical specification;
   - consequent checks on algebraic performability to avoid software exceptions have been put in;
   - static allocation of system resources is used.
f) the code is generated in a high level language which enables the application of standard compilers to produce the executable object code for the target system. It is produced in a traceable way directly linked to plant simulation codes for transient or disturbance analysis.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

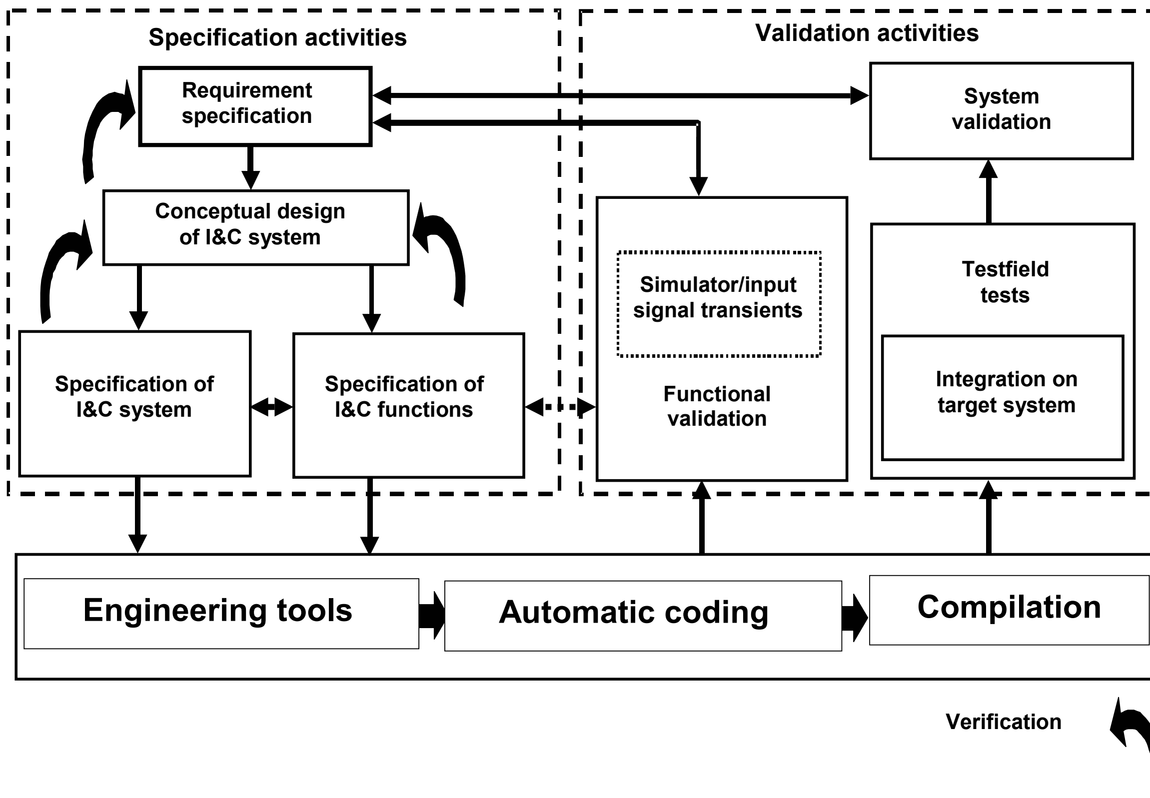


Figure C.1 – Life cycle for application oriented software engineering

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Annex D
(informative)

Language, translator, linkage editor

For a safe application of a language, its translator and linkage editor, the following more detailed recommendations are given in addition to those from the main part of this standard. These recommendations also apply to any other auxiliary system program. Recommendations for translators apply likewise to interpreters, cross compilers and emulators. Similarly, the aspects that apply to translators and linkage editors should be recognised during the selection and formal specification of design tools and their use. A project should select recommended criteria according to the priority indicated.

D.1 General

Item	Recommendations	Priority
a	Translator, linkage editor and loaders should be thoroughly tested prior to use; operation is considered very important	1
b	Reliability data of sufficient quality about translator, linkage editor and loader should be available	2
c	In the case where auxiliary system programs are used such as aids, documentation systems and the like, they should be appropriately tested before employed	1
d	Language syntax should be completely and unambiguously defined	2
e	Semantics should be well and completely specified and understandable	1
f	High level languages should be used rather than machine-oriented ones	2
g	Both the spread of a language and its problems adequacy are considered important	2
h	The recommendations of Annex B should be supported in general and as far as possible	1
i	Readability of produced code is more important than writeability during programming	2
j	Syntactic notations should be uniform; for the same concept not more than one notation should be allowed	2
k	The language should avoid error prone features	2
l	Produced programs should be easy to modify	2
m	Input parameters, output parameters and transient parameters should be syntactically distinct	2
n	An additional output that is reviewable should be provided at all stages of the translation process	3

标准分享网 www.bzfxw.com 免费下载

D.2 Error handling

Item	Recommendations	Priority
a	Language translator and linkage editor should provide for detection of as many programming errors as possible during translation time or on-line execution	2
b	During on-line execution, exception handling should be possible	2
c	The language may provide assertions	3
d	Errors likely to cause an exception during execution time should include:	
	- exceeding of array boundaries;	1
	- exceeding of value ranges;	1
	- access to variables that are not initialised;	3
	- failing to satisfy assertions;	2
	- truncation of significant digits of numerical values;	2
	- passing of parameters of wrong type.	1
e	If any error is detected during translation or linkage time, it should be reported without any attempt at correction	2
f	If it is not clear whether any rules have been violated, a warning should be issued	3
g	During translation time, parameter types should be checked	3

D.3 Data and variable handling

Item	Recommendations	Priority
a	The range of each variable should be determinable at translation time	1
b	The precision of each floating point variable and expression should be determinable at translation time	2
c	No implicit conversion between types should take place	2
d	The type of each variable, array, record component, expression, function and parameter should be determinable at translation time	2
e	Variables, arrays, parameters, etc. should be explicitly declared, including their types	1
f	Variable types should distinguish between input, output, transient and sub-routine parameters	2
g	Variable names of arbitrary length should be allowed	2
h	As far as possible, type checking should take place during translation time rather than execution time	3
i	At translation time, it should be checked whether an assignment is allowed to any particular data item	2

D.4 On-line aspects

Item	Recommendations	Priority
a	During expression evaluation, external assignment should not be allowed to any variable that is accessible in the scope of the expression	1
b	Used computing time should be examinable on-line	3
c	On-line error capture should be provided (D.2*d*)	1

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Annex E
(informative)

Software verification and testing

E.1 Software verification and testing activities

This clause is intended to provide guidance for software verification and testing activities. Depending on the program, the different methods for this purpose are more or less effective in finding program fault. The following main methods for software verification and testing are complementary: tool-based static analysis of the code (e.g. model-checking), code inspection (walkthrough) and dynamic execution (e. g. simulation or actual software execution). Most of these methods are recommended systematic approaches. A set of complementary testing approaches may be used, including statistical testing that could be applied to search for cases not exercised using the systematic approach.

A review of possible methods is given in E.4.1. When necessary, different approaches and different criteria for test data should be chosen for different program parts in order to determine whether a particular part is free from fault or determine a level of confidence. The selection depends on the internal structure of a program part, the level of reliability required, the demands imposed on it during plant operation and the available testing tools.

Software testing gives consideration to different levels of software design (e.g. module, subsystem and system levels).

E.2 Systematic approaches

Each module should be verified and tested in a systematic way according to objectives and their associated coverage criteria. In addition to manual verification and testing activities, automated verifier and testing tools should be used as much as possible. Test results should be checked with expected results derived from the program module specifications. Input to and output from the module should be handled in the same manner as in the safety system.

Subclause E.4.2 is a checklist, which can be interpreted according to the needs of each special case. Due to the enormous variety of possible practical cases, it is not possible to recommend any combination of the tests indicated for a particular class of applications. It is, however, indicated which tests should be performed under all circumstances. On the other hand, it is evident that neither all combinations of all tests nor all individual tests can be executed in a particular application.

At the subsystem level, the software is partially integrated into the system. The subsystem level tests assure the proper integration of the software modules. A distinction should be made between the case in which data flow dependency exists among the software modules and the case in which it does not. Testing should be done to provide assurance that all independent branches in the subsystem are correctly followed. Test cases at the borders of input domains and at the limits of module operational region should be used.

标准分享网 www.bzfxw.com 免费下载

The same test data set utilised at module level should be used. Results should be checked with pre-calculated values. Parameters should be input and output from the subsystem in the same manner as in the safety system. Where practical, the subsystem level testing should use a hardware configuration identical to the target hardware.

At the system level, the software is completely integrated in the hardware. The system test assures proper integration of subsystems. Testing should be performed by running programs together with a realistic simulator, or with a realistic version of the system to be monitored or controlled by the class 1 systems.

This test of the complete system should be performed in accordance with the performance statements given in the software requirements specification. Beyond the testing activities described above, a systematic timing check should be performed.

E.3 Statistical approaches

Statistical approaches may be utilised to complement systematic approaches.

Statistical testing has the following characteristics:

- tests are independent samples from a probability distribution representing operational use of the system;
- sequence and number of tests do not influence the outcome of a single test-run;
- each occurring failure is detected;
- the number of tests is large (cf. formulae below);
- failures are rare.

Typically, statistical testing may be performed to supplement the confidence in correct operation of class 1 systems that has been gained through an extensive programme of conventional functional-based testing.

The formulae developed for a probabilistic software verification can be used for the estimation of a system's probability of failure on demand under verified assumptions. The formulae provides an estimate of the upper bound of the probability of failure on demand as follows.

Assuming that n statistical tests are run on a system and 0 failures are observed. Then, for the probability of failure detection (pfd) to be smaller or equal to some target value pfd with probability α, the following condition needs to be fulfilled:

$$pfd \le -\frac{\ln(1-\alpha)}{n}.$$

Thus for α = 0,95, we obtain after n failure-free statistical tests approximately:

$pfd \le \frac{2{,}99}{n}$ with probability 0,95; for example to obtain a pfd of 10^{-4} to a probability of 95 %, a total of 29 900 tests should be performed without failure.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

For α = 0,99, we obtain approximately:

$pfd \leq \frac{4,6}{n}$ with probability 0,99; for example to obtain a pfd of 10^{-4} to a probability of 99 %, a total of 46 000 tests should be performed without failure.

The validity of the calculated pfd depends upon the similarity of the profile of the test inputs to the profile of the actual inputs experienced by the system in operation. If the above equation is used on an unrealistic operational profile, it will provide an estimate for the pfd under an imaginary profile of use, i.e. a pfd will be estimated that may be very different to the actual system availability that would be obtained in active use. This is a fundamental weakness of the statistical testing approach as it is generally very difficult to accurately determine the operational profile that a system will experience in use, and this is particularly true for systems with large numbers of inputs.

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

E.4 Verification and testing methods

E.4.1 Selected verification methods

Overview

No.	Method	Aim	Major advantages	Problems during application, major disadvantages	Cost and effort compared with development cost	Result	Relationship to other methods	Assessment
1	Supervision of testing procedure	To derive a reliability figure without additional effort	- Small additional effort for application - No detailed knowledge from test object required	- Reliability figures to be gained not sufficient - Practical cases behave only roughly according to theory	Small	Probabilistic figure, e.g. MTBF	Uses probability theory as No.2	Difficult to use for safety applications
2	Statistical testing							
2.1	According to demand	To derive a reliability figure without looking into the code's details	No detailed know-ledge from test object required	- To provide a test profile that represents input states with the same probability as the on-line operation	- Number of required test cases very high, if meaningful results are to be obtained	Probabilistic figure, e.g. availability per demand or risk per program life time	Same reasoning as for No.1	Can be reasonably applied in a short form complementary to program analysis
2.2	According to correctness			- To provide a test profile that hits any program property with equal probability	- Cost for test can be much higher	Probabilistic figure, e.g. to place a limit on the probability of failure due to possibly remaining software errors		

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Overview

No.	Method	Aim	Major advantages	Problems during application, major disadvantages	Cost and effort compared with development cost	Result	Relationship to other methods	Assessment
3	Program proving	Provide a basis for comparison with specifications	Rigorous statement on correctness achievable	No formalism available for finding loop assertions or loop invariants; error prone	In the same order of magnitude or higher	Correct/not correct, as far as proof if correct	Some aspects to be used with analysis	Possibly the only reasonable way for general “WHILE” loops
4	Program analysis		Brings good results with simple programs	Some program constructs are difficult to analyse		Free from some specific error types as far as analysis was deep enough	Same kind of thinking from program proving is used	
4.1	Manual analysis	To provide a basis for a meaningful test or for a comparison with specifications	Can be performed without additional tools	Error prone	Slightly less than development cost			Should be used if no automated tool available
4.2	Automated analysis	As above sometimes restricted to some vaguer quality measurement	- Only limited hand work - Results quickly obtained	- Many tools requires additional hand work - Some results of such tools are difficult to interpret	Depending on the available tool, much lower than development cost			Should be used as widely as possible *Most promising approach*

标准分享网 www.bzfxw.com 免费下载

E.4.2 Testing methods

E.4.2.1 General

No	Kind of test	Kind of detected errors	To be performed	Remarks
1	Cases representative for program behaviour in general, its arithmetics, timing	All, but with no guarantee for being exhaustive	Always	It is assumed that the system works correctly, if the cases are executed correctly
2	All individually and explicitly specified requirements	Forgotten functions detected completely	Always primarily, if required functions are specified in detail	- Can be an exhaustive test, if functions are kept completely separate - Does not say much on timing problems
3	All input variables in extreme positions (crash test)	Timing errors, but with no guarantee. Overflow, underflow	Always	
4	Operation of all external devices	Hardware and software interface design errors	Always	
5	Static cases and dynamic paths which are representative for the behaviour of the technical process	All, but with no guarantee	Always	Especially valuable if simulator of technical process is available
6	Correct operation shown by turning off and on each redundant subsystem/external device (some combinations should be also tested where relevant)	Hardware interface handling errors	Always	Ensures system robustness

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

E.4.2.2 Path testing

No	Kind of test	Kind of detected errors	To be performed	Remarks
7	Every statement executed at least once	Inaccessible code	Always	
8	Every outcome of every branch executed at least once	Errors in control flow, with no guarantee of completeness	Always	Contains 5; can be exhaustive, if no loops no timing problem Purely combinatorial problem mapped on program structure
9	Every predicate term exercised to each branch	Errors in control flow and data flow	If combination of tests 8 and 12 is not applied	Contains 8; included in combination of 9 and 14
10	Each loop executed with minimum, maximum and at least one intermediate number of repetitions	Errors in loop control and array data handling	Always, if program contains loops	Not applicable to 'loop forever' constructs
11	Every path executed at least once	All the erroneous control flow	To validate that design and coding choices do not make the verification overly complex	Achievable only for modules. Contains 8, note that any new loop running implies at least a new path.

E.4.2.3 Data movement testing

No	Kind of test	Kind of detected errors	To be performed	Remarks
12	Every assignment to each memory place executed at least once	Errors in data flow, but with no guarantee	Arrays used	
13	Every reference to each memory place executed at least once	Errors in data flow, exhaustive in special cases	Arrays used	Contains 12 in most cases Only reasonable in connection with 12
14	All mappings from input to output executed at least once each	All data flow errors	Always for individual segments	Only feasible for modules Possibly covered by 7, 8 or 11

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

E.4.2.4 Timing testing

No	Kind of test	Kind of detected errors	To be performed	Remarks
15	Checking of all time constraints	Timing errors, computing time too long	Always	
16	Maximum possible combinations of interrupt sequences	Organisational errors	If number not too large	
17	All significant combinations of interrupt sequences	Organisational errors	Always	

E.4.2.5 Miscellaneous

No	Kind of test	Kind of detected errors	To be performed	Remarks
18	Check for correct position of boundaries of data inputs	Erroneous subdivision of input data space	If analogue input is used	Number and kind of input data subareas to be found by analysis
19	Check for sufficient accuracy of arithmetical calculations at all critical points	Numerical errors, errors in algorithms, rounding errors	If word length of computer is short; complicated arithmetic used	
20	Only for programs; test of module interfaces and module interaction	Incorrect data transfer between modules	Always	Test aid welcome
21	Every module invoked at least once	Incorrect control flow and incorrect data flow between modules, with no guarantee	Always	
22	Every invocation to a module exercised at least once		Always	
23	Operation at high load	System timing and response errors	Always	Ensures system robustness

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Annex F
(informative)

Typical list of software documentation

References to the subclauses of this standard	Main reference	Other references
Documents relating to software production		
System requirements specification	15.2	15.3
System specification	5.3	6.1, 8.1, 9.3, 15.3
Software requirements specification	6.1	3.33 (3.35 and 3.37), 5.3, 6.4, 7, 7.1.3, 8.1, 8.2, 8.2.3.2, 11.2, 11.3, 15.3.1.2
Software quality assurance plan	5.5	
Detailed recommendations (program development)	7.3	Annex D
Software verification plan	8.2.1	
Software aspects of system integration plan	9.1	9.3
Software design specification	7.4	7.1.3, 8.1, 8.2.2, 8.2.3
Software design verification report	8.2.2	7
Software test specification	8.2.3.1.2	8.2.3.1.3
Software code verification report	8.2.3.1.1	
Software test report	8.2.3.1.3	
Software aspects of integrated system verification report	9.5	9.1, 9.2, 9.3, 9.4
Software aspects of the system validation plan	10.1	10.2, 10.4
Software aspects of the system validation report	10.3	
Software user manual	12.4.2	
Software aspects of the commissioning test plan		
Software aspects of the commissioning test report		
Documents relating to software modifications		
Anomaly report	11.1	11.3
Software modification request	11.1	11.2, 11.3
Software modification report	11.2	
Software modification control history	11.2	11.3

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Annex G
(informative)

Considerations of CCF and diversity

CCF can occur for example:

- if a latent fault is implemented in two or more components or systems and, all these components or systems are operated under the same or similar conditions so that a failure can be triggered in a timely correlated way or,
- if failure conditions propagate via data communication.

G.1 CCF due to software

For a software induced CCF, signal trajectories must trigger a software fault causing a failure that affects two or more systems or channels (for example two protection channels, two closed loop controllers, or two logic control subsystems). The reduction of the likelihood of CCF of different systems due to software can be achieved by reducing the potential that the software of these systems contains common faults and/or by operating these systems with different signal trajectories. Software faults may originate from the requirement specification for the I&C system or may be introduced during software development.

For the CCF to be of a safety concern, these failures must disable a safety function and happen within a time period in which a safety challenge could result, or it must itself cause a safety challenge such as loss of protection or control.

If the same or similar software, implementation methods or algorithms are used in redundant or in different systems, then a significant common element exists.

No agreed method of estimation currently exists for estimating the probability of failure or for failure rate arising from software faults.

G.2 Potential CCF causes and effects

G.2.1 CCF potential

A potential for a software induced CCF of different systems or between different channels in one system exists if common software or software modules are used. Potential sources of latent faults are design errors from the I&C system requirement specification, the architecture, algorithms, development methods, tools, implementation methods, or maintenance.

Requirements that are not properly understood or not correctly transformed can result in faults in the software specification resulting in risks of CCF due to exercising the resulting software fault. Deficiencies in software can be due to incorrect, incomplete, inaccurate or misunderstood software requirements and software specifications. Design errors leading to software faults can be introduced into diverse programs, due to common human factors such as training, organisation, thinking processes and design approaches.

Another potential cause of CCF could result from connection of systems to ones with lower quality software.

G.2.2 Triggering of CCF

A specific transient of input signals may trigger a CCF if it affects:

- two or more redundant channels of a system using common software;
- two systems whose functions are diverse but which use some common software modules.

A software fault may result in a software failure when a specific transient of input signals appears. If this transient is identical for two or more channels or systems, this may result in a CCF which will jeopardise one or more defence layers when sufficient quality, independence and diversity are not provided.

G.2.3 Abnormal conditions and events

Abnormal hardware failures, plant conditions and events can cause unforeseen signal trajectories, unexpected software states, transients or overload conditions that were not covered by the initial requirements or by the software design.

Potential events which may cause CCF include:

- maintenance activities;
- common timing signal failure, causing loss of timed actions;
- power supply transients causing software stop or auto-restart;
- plant trips causing communication channels to overload;
- saturation of operator capacity, causing an incorrect action;
- operator demands saturating system capacity during plant trips and transients;
- all automated controller functions engaged and operating; and
- abnormal conditions during outages and commissioning.

G.3 CCF defences

Possible defence features include:

- methods used throughout the life-cycle which aim to produce fault-free software (see 13.1);
- demonstration and enhancement of the quality of common software (see 13.2);
- use of common software in a very narrow and guaranteed set of operating conditions;
- limitations of the effects of software failures (see 13.3);
- design of the channels or systems so that a coincident failure of two channels or systems is very unlikely due to there being a demonstrable difference in the signal trajectories for the systems;
- design of the channels or systems using asynchronous operation; this may be used to show defence against the same processors in different channels being subjected to identical trajectories at the same time; and
- diverse features for some or all functions and enhancement of independence concepts during the whole life-cycle (see 13.4).

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

G.4 Demonstration of correctness

Methods of supporting demonstration of correctness include:

- reuse of proven software standard modules with a clear and proven interface (see Clause 15); typical functions include for example modules for device driving, process monitoring and input gathering, basic control algorithms (such as proportional-integral-derivative, deadband, hysteresis);
- use of tools and procedures independent of the design processes for decoding of the code loaded in memory and demonstration that the loaded code matches the specification;
- use of dynamic analysis for testing the correct software behaviour in a simulation environment which represents relevant parts of the plant (see 8.2.3.2.3);
- the use of static analysis of code to identify control and data flow, and to demonstrate correct decision and logic processes;
- the use of two software versions tested back-to-back, exercising the software by random signal trajectories. This method can be used in addition to systematic testing for detection of design and coding faults;
- performing a comprehensive bottom-up testing programme, where the correct operation of each system component is thoroughly validated before being integrated into the system.

G.5 Diversity features

a) Software diversity features of importance include:
   - functional diversity;
   - different design specifications for the same functional requirements.
b) Diversity at the system level can include:
   - use of independent systems for different actuation criteria;
   - use of different basic technology, such as computers versus hardwired design;
   - use of different types of computers, hardware modules and major design concepts;
   - use of different classes of computer technique such as PLCs, microprocessors or mini-computers.
c) The design approach features and problem solutions which enhance diversity include differences of:
   - processing algorithms;
   - data for configuration, calibration and functionality;
   - signal input hardware;
   - hardware interfaces and communications;
   - input sampling processes;
   - time sequences of operations;
   - timing processes;
   - use of historical information, latches and rates of change.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

d) Differences in design and implementation methods include:
   - languages;
   - compilation systems;
   - support libraries;
   - software tools;
   - programming techniques;
   - system and application software;
   - software structures;
   - different use of the same software modules;
   - data and data structures.
e) Diversity during tests (back-to-back testing)
f) Diverse aspects of management approach include:
   - two designs following deliberately dissimilar development methods (forced);
   - separation of the design teams;
   - restriction of communication between the teams;
   - formal communication of resolution of ambiguities in requirements or specifications;
   - use of different logic definition processes;
   - differences in documentation methods;
   - use of different staff.

G.6 Drawbacks, benefits and justification of diversity

G.6.1 Drawbacks

The disadvantages introduced by diversity may include:

- greater overall complexity;
- increased risk of spurious actuation;
- more complex specifications and design;
- control of two suppliers;
- modification problems, for example ensuring diversity is not lost during modification;
- increased documentation;
- increased space, supplies, environmental control requirements;
- the cost of several versions of the software can reduce its commercial potential, except as a testing method;
- each version produced may be of lower quality.

G.6.2 Benefits

When functional or software diversity are used, adequately diverse versions give increased protection against CCF due to software.

G.6.3 Justification

The justification can consider the improvement in reliability of the safety functions achieved by use of diversity.

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Annex H
(informative)

Tools for production and checking of specification, design and implementation

Tools now form an essential part of the development environment for software performing category A functions. Methods which are applied purely manually are highly error-prone and require the involvement of very well-trained humans. Therefore, they should be supported by tools which use techniques to reveal the structure and internal functional relationships of the software and to check for internal consistency, consistency with some prior model, desirable/undesirable properties, etc.

Final demonstration that the code meets its specification can be performed by means of a compliance analyser. Where a proven code generator ensures that the executable code is fully consistent with its design description, then static and dynamic analyses of the code provide a diverse check on the correctness of that description.

Tools for formal specification and design methods can be classified as constructive or analytical tools.

H.1 Constructive tools

Constructive tools are used to support the specification, design and implementation phases of development, and may include:

H.1.1 Text editor

Because formal methods based on set theory and on predicate and propositional calculus require special mathematical symbols, it is important that a suitable text editor is available capable of both displaying these on a high definition screen and printing them out legibly.

H.1.2 Graphical interface

A suitable graphics capability is required where a formal method involves the use of graphics.

H.1.3 Automated code generator

Once a formal specification has been proved, the integrity of the design process can be greatly enhanced by the use of a validated automated code generator. Such a code generator will transform the specification into executable code and thus reduce the likelihood of introducing errors. Additionally, a safe sub-set of a language may be enforced through the code generator design.

Certified software modules should be used for standard functions.

Automatically generated code should be readable. Comments should support the identification of the associated parts of the specification. The structure of automatically generated code should support automated verification.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

H.1.4 Proof obligation generator

Formal methods based on logical reasoning require a proof obligation generator that automatically records the proof obligations arising during the design steps.

H.2 Analytical tools

Analytical tools enable checking of the specification, design and implementation. They may include the following.

H.2.1 Syntax checker

A syntax checker provides information on the program structure, the use of program data, the dependency of output variables on input variables, and the control flow through the program allowing:

a) identifying structure defects like multiple starts, multiple ends, unreachable code, redundant code, non-usage of function results;
b) identifying module/subroutine hierarchy;
c) identifying violation of standards and programming conventions, including checks for unconditional branches into loops;
d) identifying data that is read before written, data written before read, data written twice without an intervening read;
e) checking information flow against specification;
f) assisting the design of a dynamic test plan;
g) test data management and possible test data generation.

H.2.2 Semantic checker

A semantic checker describes the mathematical relationships between output and input variables for every semantically feasible path through the branch free regions of the program. This allows checking what the program will do under all circumstances and detection of faults such as: unexpected output values affected by input values, incorrect response to unexpected input values, incorrect polarity of functions and operators, etc.

H.2.3 Formal proof generator

Formal proofs of a design require the use of an interactive program which carries out the necessary symbol manipulations under the guidance of a human operator in order to discharge the proof obligations. Such a program is known as a theorem proving assistant (TPA). This usually means the application of a proof checker whose input is the output of the TPA. TPA are large programs for which one cannot prove the absence of faults. Therefore, a diversified verification is needed and should be performed. This usually means the application of a proof checker whose input is the output of the TPA. The proof checker should be based on a formalised proof theory and it should be verified against this proof theory.

H.2.4 Animator

Where possible the specifications should be animated so that the end user of the system can examine aspects of the specification or design in order to validate (as far as is practicable) the requirements specification and proposed design. The animation should be as representative as possible of the design and may require the use of prototypes to demonstrate the non-

标准分享网 www.bzfxw.com 免费下载

functional aspects. This evaluation is done against the user's criteria, and the system requirements may be modified in light of this evaluation.

H.2.5 Compliance analyser

A compliance analyser can demonstrate that the code correctly implements the specification. Such a tool uses pre- and post-conditions plus loop invariables in such a demonstration. The tool confirms systematically that each condition is fulfilled in the code.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Annex I
(informative)

Requirements concerning pre-developed software (PDS)

I.1 Guidance for graduating non-conformities and compensating factors

Weakness and non-conformities with respect to IEC 60880 requirements, may be of different types, for example:

– requirements concerning the understandability and the completeness of the software documentation;

– requirements concerning the readability of the programs;

– requirements related to software design that enhance the deterministic behaviour of real time performance;

– requirements related to software that provide self-supervision of the computer-based hardware; or

– requirements related to the completeness of documented validation reports.

Weakness and non-conformities are rarely a "black and white" issue and should be graduated in terms of degree of fulfilment and according to the impact on the final quality of the computer-based system. For instance:

– IEC 60880 requirements concerning understandability are critical for development and for modifications and less significant for a stable and proven product.

The possibility of taking credit of operating experience or additional testing as compensating factors ought to be judged according to the type of the PDS and its role in the computer-based system. For instance:

– operational system software, normally available as micro-code or binary-code, may be difficult to analyse and there may be limited access to the documentation of the development process. In some cases, a large amount of operating experience may be available and may be an important acceptance factor particularly for software performing defined repetitive functions, for example communication drivers or parts of operating systems. In other cases, for example software that implements surveillance and recovery functions of the computer-based system, significant operational experience may be not available because these functions are seldom activated during the operation of the computer-based systems;

– application software libraries may be thoroughly documented on the basis of accessible information on module design, development process and validation tests. Feedback of operating experience from similar plant applications or additional testing may also be used to support the qualification so that a high degree of confidence may be reasonably achieved in the correctness of the execution of this type of software.

标准分享网 www.bzfxw.com 免费下载

I.2 Collection and validation of data on the operational history

I.2.1 Data collection

Data to be collected should include:

– site information: including configuration and operational conditions of the PDS in the computer-based system, functions used, number of PDS running;

– site operating time: including elapsed time since first start-up, elapsed time since last release of the PDS, elapsed time since last severe error (if any), elapsed time since last error report (if any);

– error report: including date of error, severity, fixes; and

– release history: including date and identification of releases and associated configuration, errors fixed, functional modifications or extensions, pending problems.

I.2.2 Data verification

The statistical relevance of collected data should be verified keeping in mind different factors:

– data on operating time are acceptable only when a minimum pre-established time has elapsed since the commissioning of the PDS;

– the PDS operating time may be taken into account only when it refers to features of the PDS which are actually used during operation of the computer-based system where the PDS is installed;

NOTE For instance, some modules of the application libraries, e.g. software related to auxiliary system functions, can be installed but never or seldom used on a particular site.

– completeness of data describing operation history (e.g. are all malfunctions really documented correctly?) should be estimated depending on the site organisation of the modification staff;

– only data from genuine and unbiased information sources are acceptable.

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Annex J
(informative)

Correspondence between IEC 61513 and this standard

J.1 General

IEC 60880 (1986) has been developed several years before IEC 61513 and includes system level clauses in addition to software level clauses.

During the revision process, the clauses relevant to system level and now adequately addressed in IEC 61513 have been removed from the revised text. Clauses explicitly referenced by IEC 61513 or not adequately addressed have been kept in the revised text and are provided in this annex, with IEC 60880 (1986) and IEC 60880 revision clause numbers.

J.2 IEC 60880 (1986) clauses referenced by IEC 61513

IEC 61513 is a system level document covering all categories of safety I&C functions.

It states in the Introduction:

"When class 1 computer-based systems are addressed, this standard has to be used in conjunction with IEC 60880, IEC 60880-2 and IEC 60987 to assure the completeness of the system requirements for software and hardware aspects."

IEC 61513 makes explicit references to IEC 60880 requirements for several subjects (integration, validation, and modification).

The related clauses have been kept in this revision of IEC 60880, and revised where appropriate.

This table gives the correspondence between the clauses referenced by IEC 61513 in IEC 60880 (1986), IEC 60880-2 here designated by 'Suppl' and the revised clauses in this document.

Table J.1 – Correspondence between clauses of IEC 61513 and this standard

IEC 61513 clauses referring to	IEC 60880 (1986)/Suppl. clause	Clause in this standard
5.3.1.5.4 Defence against CCF due to systematic faults NOTE 2 This subclause is intended to give an overview. Detailed requirements for defences against CCFs due to software faults for category A functions are given in 4.1 IEC 60880-2.	4.1 Suppl	13
5.3.3.1 Assessment of reliability and defences against CCF NOTE 2 Requirements for the analysis of CCF due to software are given in 4.1.1 of IEC 60880-2.	4.1.3 Suppl	13.3

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

IEC 61513 clauses referring to	IEC 60880 (1986)/Suppl. clause	Clause in this standard
5.5.1 Architectural design documentation NOTE Requirements on software engineering methods and tools for class 1 systems are given in 4.2 and 4.3 of IEC 60880-2	4.2 Suppl 4.3 Suppl	14 15
6. System safety life cycle NOTE 1 This requirement differs from that stated in 6.1 of IEC 60880. For software it is desirable, but not considered necessary, to complete each phase of a development before starting the next phase providing the requirements defined above have been addressed NOTE 2 For class 1 systems, software requirements on this activity are defined in IEC 60880. NOTE 3 For class 1 systems, requirements for predeveloped software are defined in IEC 60880-2.	6.1 None None	8.1
6.1.1.1.1 Application function … An estimate of the quantitative reliability of the application functions may be required for verification of the system design and of the plant design basis (see A.2.2 of IEC 60880). This estimation is normally performed for the system hardware design where well-established practices exist but there is no generally recognised method available for the quantitative evaluation of software reliability (see 6.1.3.1.2). …	A.2.2	None
6.1.1.2.1 System architecture NOTE Failures due to software are systematic and not random failures. Therefore, the single-failure criterion cannot be applied to the software design of a system in the same manner as it can be applied for hardware design. Possible effects of CCF due to software inside each defence layer and between redundant layers have to be considered at the level of each system and of the I&C architecture (see 4.1.1 of IEC 60880-2)	4.1.1 Suppl	13.1
6.1.1.2.2 Internal behaviour of the system c) (Specific requirement) In order to provide a high degree of assurance on deterministic behaviour, class 1 systems should be developed by techniques such as those of appendix B of IEC 60880 (notably B2.*d* on execution time and B2.*e* on interrupts). … NOTE 3 See Clause 1 of IEC 60880 concerning the role of the appendices to the standard and what is required if practices differing from those of the appendices are used.	B2.d B2.e 1	B2.d B2.e 5.5.3
6.1.1.2.3 Self-monitoring and tolerance to failure a) Systems should be designed so that errors and failures are detected sufficiently early to maintain the required system availability. The detection of failures by self-test facilities should be balanced with the additional complexity that is introduced. The requirements of 4.8 and A.2.8 of IEC 60880 on self-supervision should be supported in general and as far as possible for each class of system	4.8 A.2.8	6.2 A.2.2
6.1.2 System specification In accordance with Clause A.1 of IEC 60880, this phase includes the activities necessary to produce the software requirements, the hardware requirements and the integration requirements of the system	A.1	5.3
6.1.2.1 Selection of pre-existing component NOTE 2 Subclause 4.3 of IEC 60880-2 addresses acceptance criteria for reusable pre-existing software for category A functions.	4.3	15
6.1.2.2.3 Defence against propagation and side-effects of failure NOTE Detailed requirements for avoidance of error-prone software structures and for verification and tests of software modules are given in IEC 60880 and IEC 60880-2.	None	None

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

IEC 61513 clauses referring to	IEC 60880 (1986)/Suppl. clause	Clause in this standard
6.1.2.3 Software specification NOTE 1 The software architecture defines the major components and subsystems of the software, how they are interconnected, and how the required attributes will be achieved. The requirements for software architecture are outwith the scope of this standard. (For class 1 systems refer to IEC 60880 and IEC 60880-2.) a) In order to ease the specification, verification and validation of the application functions, the software architecture should provide a clear separation between the application software and the system software (see B2.a of IEC 60880). In such cases, the verification and validation of the application software may be performed independently.	None B2.a	None B2.a
6.1.3 System detailed design and implementation For class 1 systems, requirements on software development are established in IEC 60880 and IEC 60880-2, and hardware requirements in IEC 60987.	None	None
6.1.4 System integration The objective of this phase is to assemble hardware and software modules and verify compatibility of the software loaded into the hardware (see Clause 7 of IEC 60880).	7	9
6.1.5 System validation The objective of this phase is to test the integrated system to ensure compliance with the functional, performance and interface specifications (see Clause 8 of IEC 60880). b) (Specific requirement) The requirements of Clause 8 of IEC 60880 shall apply to validation of category A functions.	8	10
6.1.7 System design modification e) (Specific requirement) For class 1 systems, the modification process of software shall be in accordance with Clause 9 of IEC 60880 and the modification process of hardware in accordance with Clause 11 of IEC 60987.	9	11
6.2.1 System quality assurance plan NOTE The requirements for software quality assurance plan of safety systems are defined in Clause 3 of IEC 60880. e) The quality assurance plan shall be established at an early stage of the system life cycle and shall be planned within the general schedule of the other activities of the I&C safety life cycle. The plan may be either a part of the system specification or a companion document (see 3.2 of IEC 60880).	3 3.2	5.5 5.5
6.2.1.1 System verification plan h) (Specific requirement) For class 1 systems, the verification plan shall be executed by individuals independent of the designers of the system (according to 6.2.1 of IEC 60880).	6.2.1	8.2.1
6.2.3 System integration plan System integration covers the activities to integrate subsystems into the system and to integrate hardware and software for CB systems. It covers notably the activities described in 7.4 of IEC 60880.	7.4	9.1
6.2.4 System validation plan b) (Specific requirement) For category A functions, the system validation plan shall be developed and validation activities performed by teams independent from the ones who designed, implemented, and or modified the system (see Clause 8 of IEC 60880).	8	10
6.2.5 System installation plan b) (Specific requirement) For class 1 systems, the installation plan shall meet the requirements of Clause 9 of IEC 60987 and 10.1.1 of IEC 60880.	10.1.1	None

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

标准分享网 www.bzfxw.com 免费下载

IEC 61513 clauses referring to	IEC 60880 (1986)/Suppl. clause	Clause in this standard
6.3.1.2 Characteristics NOTE 3 Detailed requirements regarding software tools for class 1 systems are given in 4.2 of IEC 60880-2.	4.2 Suppl	14
6.3.3 System detailed design documentation NOTE For class 1 systems, requirements on software documentation are established in IEC 60880 and IEC 60880-2, and requirements on hardware documentation in IEC 60987.	None	None
6.3.4.2 Characteristics b) (Specific requirement) For class 1 systems, the requirements of 7.7 of IEC 60880 apply.	7.7	9.5
6.3.5.2 Characteristics (Specific requirement) For class 1 systems, the requirements of 8.1 of IEC 60880 apply.	8.1	10.3
6.4.1.2 Software evaluation and assessment b) (Specific requirement) For systems of class 1, newly developed software shall be evaluated and assessed in accordance with the requirements of IEC 60880. c) (Specific requirement) Pre-existing equipment selected for class 1 systems should have been developed according to recognized guides and standards appropriate to the high level of quality required for class A functions (see 8.1.2 of IEC 61226). In particular, the requirements of IEC 60880-2 on pre-developed software and tools and the requirements of IEC 60987 shall be met.	None None Suppl	
Table 5 Requirements for the specification and implementation of the FSE Validation: IEC 60880 (6.2.4)	Clause not defined	

References from IEC 61513 to IEC 60880 should be updated during the next revision of IEC 61513.

J.3 IEC 60880 (1986) clauses not referenced by IEC 61513 but relevant to the system level

These clauses have been kept in this revision of IEC 60880 and revised where appropriate. They should be considered as input during the next revision of IEC 61513 and then of this standard.

Table J.2 – IEC 60880 clauses to be considered during the next revision of IEC 61513

6.3 Periodic testing
9.2 System integration
10.2 System validation
12.4 Operator training

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

This page deliberately left blank

标准分享网 www.bzfxw.com 免费下载

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

This page deliberately left blank

Licensed Copy: athen reading, Reading University Library, 23/01/2010 05:46, Uncontrolled Copy, (c) BSI

British Standards Institution (BSI)

BSI is the independent national body responsible for preparing British Standards and other standards-related publications, information and services. It presents the UK view on standards in Europe and at the international level.

It is incorporated by Royal Charter.

Revisions

British Standards are updated by amendment or revision. Users of British Standards should make sure that they possess the latest amendments or editions.

It is the constant aim of BSI to improve the quality of our products and services. We would be grateful if anyone finding an inaccuracy or ambiguity while using this British Standard would inform the Secretary of the technical committee responsible, the identity of which can be found on the inside front cover.

Tel: +44 (0)20 8996 9001 Fax: +44 (0)20 8996 7001

BSI offers Members an individual updating service called PLUS which ensures that subscribers automatically receive the latest editions of standards.

Tel: +44 (0)20 8996 7669 Fax: +44 (0)20 8996 7001
Email: plus@bsigroup.com

Buying standards

You may buy PDF and hard copy versions of standards directly using a credit card from the BSI Shop on the website **www.bsigroup.com/shop.** In addition all orders for BSI, international and foreign standards publications can be addressed to BSI Customer Services.

Tel: +44 (0)20 8996 9001 Fax: +44 (0)20 8996 7001
Email: orders@bsigroup.com

In response to orders for international standards, it is BSI policy to supply the BSI implementation of those that have been published as British Standards, unless otherwise requested.

Information on standards

BSI provides a wide range of information on national, European and international standards through its Knowledge Centre.

Tel: +44 (0)20 8996 7004 Fax: +44 (0)20 8996 7005
Email: knowledgecentre@bsigroup.com

Various BSI electronic information services are also available which give details on all its products and services.

Tel: +44 (0)20 8996 7111 Fax: +44 (0)20 8996 7048
Email: info@bsigroup.com

BSI Subscribing Members are kept up to date with standards developments and receive substantial discounts on the purchase price of standards. For details of these and other benefits contact Membership Administration.

Tel: +44 (0)20 8996 7002 Fax: +44 (0)20 8996 7001
Email: membership@bsigroup.com

Information regarding online access to British Standards via British Standards Online can be found at **www.bsigroup.com/BSOL**

Further information about BSI is available on the BSI website at **www.bsigroup.com/standards**

Copyright

Copyright subsists in all BSI publications. BSI also holds the copyright, in the UK, of the publications of the international standardization bodies. Except as permitted under the Copyright, Designs and Patents Act 1988 no extract may be reproduced, stored in a retrieval system or transmitted in any form or by any means – electronic, photocopying, recording or otherwise – without prior written permission from BSI. This does not preclude the free use, in the course of implementing the standard of necessary details such as symbols, and size, type or grade designations. If these details are to be used for any other purpose than implementation then the prior written permission of BSI must be obtained. Details and advice can be obtained from the Copyright & Licensing Manager.

Tel: +44 (0)20 8996 7070
Email: copyright@bsigroup.com

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

Tel +44 (0)20 8996 9001
Fax +44 (0)20 8996 7001
www.bsigroup.com/standards

raising standards worldwide™

标准分享网 www.bzfxw.com 免费下载