IEC 62628

Edition 1.0 2012-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Guidance on software aspects of dependability

Lignes directrices concernant la sûreté de fonctionnement du logiciel

IEC 62628:2012

About the IEC

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

About IEC publications

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

Useful links:

IEC publications search - www.iec.ch/searchpub

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It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

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

A propos de la CEI

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

A propos des publications CEI

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

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d’études,…).
Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC®

IEC 62628

Edition 1.0 2012-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Guidance on software aspects of dependability

Lignes directrices concernant la sûreté de fonctionnement du logiciel

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XB

ICS 03.120.01

ISBN 978-2-83220-303-3

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

GUIDANCE ON SOFTWARE ASPECTS OF DEPENDABILITY

FOREWORD

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

International Standard IEC 62628 has been prepared by IEC technical committee 56: Dependability.

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

FDIS	Report on voting
56/1469/FDIS	56/1480/RVD

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

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

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

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

INTRODUCTION

Software has widespread applications in today's products and systems. Examples include software applications in programmable control equipment, computer systems and communication networks. Over the years, many standards have been developed for software engineering, software process management, software quality and reliability assurance, but only a few standards have addressed the software issues from a dependability perspective.

Dependability is the ability of a system to perform as and when required to meet specific objectives under given conditions of use. The dependability of a system infers that the system is trustworthy and capable of performing the desired service upon demand to satisfy user needs. The increasing trends in software applications in the service industry have permeated in the rapid growth of Internet services and Web development. Standardized interfaces and protocols have enabled the use of third-party software functionality over the Internet to permit cross-platform, cross-provider, and cross-domain applications. Software has become a driving mechanism to realize complex system operations and enable the achievement of viable e-businesses for seamless integration and enterprise process management. Software design has assumed the primary function in data processing, safety monitoring, security protection and communication links in network services. This paradigm shift has put the global business communities in trust of a situation relying heavily on the software systems to sustain business operations. Software dependability plays a dominant role to influence the success in system performance and data integrity.

This International Standard provides current industry best practices and presents relevant methodology to facilitate the achievement of software dependability. It identifies the influence of management on software aspects of dependability and provides relevant technical processes to engineer software dependability into systems. The evolution of software technology and rapid adaptation of software applications in industry practices have created the need for practical software dependability standard for the global business environment. A structured approach is provided for guidance on the use of this standard.

The generic software dependability requirements and processes are presented in this standard. They form the basis for dependability applications for most software product development and software system implementation. Additional requirements are needed for mission critical, safety and security applications. Industry specific software qualification issues for reliability and quality conformance are not addressed in this standard.

This standard can also serve as guidance for dependability design of firmware. It does not however, address the implementation aspects of firmware with software contained or embedded in the hardware chips to realize their dedicated functions. Examples include application specific integrated circuit (ASIC) chips and microprocessor driven controller devices. These products are often designed and integrated as part of the physical hardware features to minimize their size and weight and facilitate real time applications such as those used in cell phones. Although the general dependability principles and practices described in this standard can be used to guide design and application of firmware, specific requirements are needed for their physical construction, device fabrication and embedded software product implementation. The physics of failure of application specific devices behaves differently as compared to software system failures.

This International Standard is not intended for conformity assessment or certification purposes.

GUIDANCE ON SOFTWARE ASPECTS OF DEPENDABILITY

1 Scope

This International Standard addresses the issues concerning software aspects of dependability and gives guidance on achievement of dependability in software performance influenced by management disciplines, design processes and application environments. It establishes a generic framework on software dependability requirements, provides a software dependability process for system life cycle applications, presents assurance criteria and methodology for software dependability design and implementation and provides practical approaches for performance evaluation and measurement of dependability characteristics in software systems.

This standard is applicable for guidance to software system developers and suppliers, system integrators, operators and maintainers and users of software systems who are concerned with practical approaches and application engineering to achieve dependability of software products and systems.

2 Normative references

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

IEC 60050-191, *International Electrotechnical Vocabulary – Chapter 191: Dependability and quality of service*

IEC 60300-3-15, *Dependability management – Part 3-15: Application guide – Engineering of system dependability*

3 Terms, definitions and abbreviations

For the purposes of this document, the terms and definitions given in IEC 60050-191, as well as the following apply.

3.1 Terms and definitions

3.1.1
software
programs, procedures, rules, documentation and data of an information processing system

Note 1 to entry: Software is an intellectual creation that is independent of the medium upon which it is recorded.

Note 2 to entry: Software requires hardware devices to execute programs and to store and transmit data.

Note 3 to entry: Types of software include firmware, system software and application software.

Note 4 to entry: Documentation includes: requirements specifications, design specifications, source code listings, comments in source code, “help” text and messages for display at the computer/human interface, installation instructions, operating instructions, user manuals and support guides used in software maintenance.

3.1.2
firmware
software contained in a read-only memory device, and not intended for modification

EXAMPLE Basic input/output system (BIOS) of a personal computer.

Note 1 to entry: Software modification requires the hardware device containing it to be replaced or re-programmed.

3.1.3
embedded software
software within a system whose primary purpose is not computational

EXAMPLES Software used in the engine management system or brake control systems of motor vehicles.

3.1.4
software unit
software module
software element that can be separately compiled in programming codes to perform a task or activity to achieve a desired outcome of a software function or functions

Note 1 to entry: The terms "module" and "unit" are often used interchangeably or defined to be sub-elements of one another in different ways depending upon the context. The relationship of these terms is not yet standardized.

Note 2 to entry: In an ideal situation, a software unit can be designed and programmed to perform exactly a specific function. In some applications, it may require two or more software units combined to achieve the specified software function. In such cases, these software units are tested as a single software function.

3.1.5
software configuration item
software item that has been configured and treated as a single item in the configuration management process

Note 1 to entry: A software configuration item can consist of one or more software units to perform a software function.

3.1.6
software function
elementary operation performed by the software module or unit as specified or defined as per stated requirements

3.1.7
software system
defined set of software items that, when integrated, behave collectively to satisfy a requirement

EXAMPLES Application software (software for accounting and information management); programming software (software for performance analysis and CASE tools) and system software (software for control and management of computer hardware system such as operating systems).

3.1.8
software dependability
ability of the software item to perform as and when required when integrated in system operation

3.1.9
software fault
bug
state of a software item that may prevent it from performing as required

Note 1 to entry: Software faults are either specification faults, design faults, programming faults, compiler-inserted faults or faults introduced during software maintenance.

Note 2 to entry: A software fault is dormant until activated by a specific trigger, and usually reverts to being dormant when the trigger is removed.

Note 3 to entry: In the context of this standard, a bug is a special case of software fault also known as latent software fault.

3.1.10
software failure
failure that is a manifestation of a software fault

Note 1 to entry: A single software fault will continue to manifest itself as a failure until it is removed.

3.1.11
code
character or bit pattern that is assigned a particular meaning to express a computer program in a programming language

Note 1 to entry: Source codes are coded instructions and data definitions expressed in a form suitable for input to an assembler, compiler, or other translator.

Note 2 to entry: Coding is the process of transforming of logic and data from design specifications or descriptions into a programming language.

Note 3 to entry: A programming language is a language used to express computer programs.

3.1.12
(computer) program
set of coded instructions executed to perform specified logical and mathematical operations on data

Note 1 to entry: Programming is the general activity of software development in which the programmer or computer user states a specific set of instructions that the computer must perform.

Note 2 to entry: A program consists of a combination of coded instructions and data definitions that enable computer hardware to perform computational or control functions.

3.2 Abbreviations

ASIC	Application specific integrated circuit
CASE	Computer-aided software engineering
CMM	Capability maturity model
CMMI	Capability maturity model integration
COTS	Commercial-off-the-shelf
FMEA	Failure mode and effects analysis
FTA	Fault tree analysis
IP	Internet protocol
IT	Information technology
KSLOC	Kilo-(thousand) source lines of code
ODC	Orthogonal defect classification
RBD	Reliability block diagram
USB	Universal serial bus

4 Overview of software aspects of dependability

4.1 Software and software systems

Software is a virtual entity. In the context of this standard, software refers to procedures, programs, codes, data and instructions for system control and information processing. A software system consists of an integrated collection of software items such as computer programs, procedures, and executable codes, and incorporated into physical host of the processing and control hardware to realize system operation and deliver performance functions. The hierarchy of the software system can be viewed as a structure representing the system architecture and consisting of subsystem software programs and lower-level software units. A software unit can be tested as specified in the design of a program. In some cases, two or more software units are required to construct a software function. The system encompasses both hardware and software elements interacting to provide useful functions in rendering the required performance services.

In a combined hardware/software system, the software elements of the system contribute in two major roles: a) operating software to run continuously to sustain hardware elements in system operation; and b) application software to run as and when required upon user demands for provision of specific customer services. Dependability analysis of the software sub-systems has to consider the software application time factors in the system operational profile and those software elements required for full-time system operation. Software modelling is needed for reliability allocation and dependability assessment of software-based systems.

Human aspects of dependability [1][1] play a pivotal role in guiding effective software design and implementation. The human-machine interface and operating environment influence the outcome of software and hardware interaction and affect the dependability of system performance. This leads to a strategic need for software dependability design and perfective maintenance efforts in the software life cycle process [2].

4.2 Software dependability and software organizations

Software dependability is achieved by proper design and appropriate incorporation into system operation. This standard presents an approach where existing dependability techniques and established industry best practices can be identified and used for software dependability design and implementation. The dependability management systems [3, 4] describe where relevant dependability activities can be effectively implemented in the life cycle process. The achievement of software dependability is influenced by

- management policy and technical direction;
- design and implementation processes;
- project specific needs and application environments.

Software organizations are organized and managed groups that have people and facilities with responsibilities, authorities and relationships involving software as part of their routine activities. They exist in governments, public and private corporations, companies, associations and institutions. Software organizations are structured according to specific business needs and application environments for various combinations of development, operation and service provision.

Typical software organizations include those that

a) develop software as their primary product,
b) develop hardware products with embedded software,
c) provide software service support to clients,
d) operate and maintain software networks and systems.

Annex A describes the categorization of software and software applications provided by typical software organizations.

4.3 Relationship between software and hardware dependability

Software behaviour and performance characteristics are different than those experienced in hardware from a dependability perspective. Software codes are created by humans. They are susceptible to human errors, which are influenced by the design environment and organizational culture. Whereas most hardware component failure data are well documented and experienced in use environment, the nature of software faults and their traceability of

[1] Reference in square brackets refers to the bibliography.

cause and effects are not easy to determine in system operation. In most cases the software faults leading to system failures cannot be consistently duplicated. Corrective actions on system failures due to software faults do not guarantee total elimination of the root causes of the software problem.

A bug, after being triggered, results in a software failure (event) and exhibits as a software fault (state). All software faults that cause the inability of the software to accomplish its intended functions are noticed by the software user. Faults and bugs cause problems in the software to perform as designed. Software containing bugs could still accomplish its intended function that is not noticeable to the user. Bugs could cause failures, but could also create nuisance issues that are not affecting a certain function. A software fault can cause system failure, which may exhibit systematic failure symptom.

Software systems and hardware products also have many similarities. They both are managed throughout their design and development stages, and followed by integration and test and production. The discovery of failures and latent faults occur through rigorous analysis, test and verification process with high-levels of test or fault coverage. The high-levels of coverage of the verification process are determined by the assessment of its percentage of fault detection, or fault detection probability. While the management techniques are similar, there are also differences [5, 6]. The following are some examples:

- Software has no physical properties, while hardware does. Software does not wear-out. Failures attributable to software faults appear without advance warning and often provide no indication that they have occurred. Hardware often provides a period of gradual wear-out and possibly graceful degradation until reaching a failed condition.
- Changes to software are flexible and much less time consuming or costly as compared to hardware design changes. Changes to hardware designs require a series of time-consuming adjustments to capital equipment, material procurement, fabrication, assembly, and documentation. However, regression testing of large and complex software programs could be constrained by time and cost limitations.
- Hardware verification and testing is simplified since it is possible to conduct limited testing through knowledge of the physics of the device to analyse and predict behaviour. Software testing can also become simplified through regression testing and analysis to verify minor changes to software due to an identified failure cause. However, minor changes to correct probabilistic failure causes of software, such as race conditions, could lead to very elaborate test and verification cycles to demonstrate adequate correction of the problem.
- Repair and maintenance actions would restore hardware to its operational state generally without design changes. Software repair and maintenance would involve design changes with new service packs or software releases to correct or rectify software faults.

4.4 Software and hardware interaction

Software and hardware interaction occurs in system operation. Dependability issues exist in the interface between the hardware and the operating system. The issues are generally resolved by incorporation of error detection and correction techniques, and exception handling of the hardware and the operating system to mitigate physical faults, and information and timing errors that exist in the interaction. The advent of multi-core processors has enabled redundant multi-threading to enhance dependability in system performance. This enables the user, programmer, or system architect to influence and exploit the redundancy inherent in the multi-core processors to enhance detection and recovery from errors. This also provides opportunity for recovery from soft errors or transient errors that affect either hardware or software or both. The exploitation of increased complexity in multi-core redundancy should be taken into consideration in such applications.

In any control system, the system is controlling some physical processes of actual hardware devices such as sensors and actuators that can fail in system operation. Many of these devices contain embedded software not accessible to the system designer or architect. Examples include smart sensors that contain error detection, redundancy and some error correction features, which are driven by the embedded software. It is important to review the software control algorithms. This is to ensure that the control algorithms are resilient to bad sensor data and missing sensor values, and that they can detect failed actuation and are capable to compensate or revert to fail-safe condition. Sensor feedback is essential to confirm successful actuation. The feedback mechanism should contain some independent checking of the effects of the commanded actuation. The control system behaviour, assumptions and failure modes should be considered in the design of the software control system.

Intentional and malicious injection of hardware faults to thwart or foil the software algorithms could happen when the system is exposed to deliberate cyber attack. For example, one can inject hardware faults into a cryptographic system to extract the key, or inject a virus into the USB device that is used to initialize a voting machine. The software and hardware interaction could create serious problems to the system operation and affect the dependability in system performance.

Interoperability problems associated with software and hardware interaction could also exist when the software is inappropriately reused in a different environment or for a different application.

The solution to dependability problems related to software and hardware interaction is to increase better understanding of how the new technological system works, and to exercise caution in conducting dependability assessment and testing to fully consider the effects of hardware failures on the software system.

5 Software dependability engineering and application

5.1 System life cycle framework

A system life cycle framework should be established to guide product development and system implementation. The framework is used for defining the system life cycle and governing the performance of the system life cycle processes. IEC 60300-3-15 describes the engineering of system dependability and life cycle implementation, which is based on the technical processes of ISO/IEC 15288 [7]. This applies to any system, whether composed of hardware, software or both.

5.2 Software dependability project implementation

Software engineering activities during the design cycle and useful life period of the system life cycle should be planned, coordinated and managed accordingly along with their hardware counterparts. Engineering activities during the useful life period would involve design changes that could be caused by high failure rates in the customer application, or hardware obsolescence while supplying spares for sustainment of operations. As the hardware changes over the product life cycle, the software would need to change as well. Changes to the software are necessary, as the system design requires forward and backward compatibility between different versions and configurations of the system design.

Dependability activities should be integrated in the respective project plans and incorporated in the system engineering tasks for effective system design, realization, implementation, operation and maintenance. The guidance to engineering dependability into systems per IEC 60300-3-15 applies to this standard. The guidance on software aspects of dependability consists of the following recommended procedures for software dependability achievement in software project implementation:

a) identify the software application objectives and requirements relevant to the software life cycle (see 5.3) and application environment (see Clause A.2);
b) identify the applicable software dependability attributes (see 5.4) relevant to the software project;
c) review the adequacy of dependability management processes and available resources to support software project development and implementation (see 5.5);
d) establish software requirements and dependability objectives (see 5.6, Annex B);

e) classify software faults (see 5.7) and identify relevant software metrics (see 6.2, Annex E) for software dependability strategy implementation (see 5.8);

f) apply relevant dependability methodology for software design and realization (see 6.1, 6.3);

g) initiate dependability improvement where needed taking into consideration of various constraints and limitations for project tailoring (see 6.4, 7.2);

h) monitor development and implementation process for control and feedback to sustain software operability and assure dependability in system operation (see Clause 7).

5.3 Software life cycle activities

The software life cycle encompasses the following activities:

- *requirements definition* identifies the system requirements for combined hardware and software elements in response to the users' needs and constraints of system applications;
- *requirements analysis* determines the feasible design options and transforms the system requirements for service applications into a technical view for hardware and software subsystem design and system development;
- *architectural design* provides a solution to meet system requirements by allocation of system elements into subsystem building blocks to establish a baseline structure for software subsystem decomposition and identify relevant software functions to meet the specified requirements;
- *detailed design* provides a design for each identified function in the system architecture and creates the needed software units and interfaces for the function which can be apportioned to software, hardware, or both. The functions apportioned to software are defined with sufficient details to permit coding and testing. The software function can be labelled as software subsystem and identified as a software configuration item for design control;

- *realization* produces the executable software units that meet verification criteria and design requirements including lower level activities in
  - *coding* of the software units;
  - *unit test* for verification of software unit to meet design requirements;
  - *subsystem test* for verification of software program functions to meet design requirements;
- *integration* assembles the software units and subsystems consistent with the architectural design configuration and installs the complete software system in the host hardware system for testing;
- *acceptance* establishes the system capability and validates the software applications to provide the required performance service for specified system operations in the target environment; software acceptance tests include lower level activities in
  - *reliability growth testing* to increase the reliability of the software system; the testing is conducted after the software system is fully integrated and executed in simulated field operational conditions representing the target environment;
  - *qualification testing* to validate acceptance of the software system for customer release;
- *software operation and maintenance* engages the software in system operation, sustains the system operational capability and responds to application service demands to deliver specific operational services;
- *software update/enhancement* improves the software performance with added features;
- *software disposal* terminates the support of specific software service.

Annex B presents typical software system requirements and related dependability activities for the software life cycle stages.