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

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2012 IEC, Geneva, Switzerland

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

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de la CEI ou du Comité national de la CEI du pays du demandeur.
Si vous avez des questions sur le copyright de la CEI ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de la CEI de votre pays de résidence.

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

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

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

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a 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It also gives information on projects, replaced and withdrawn publications.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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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

Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.

® Registered trademark of the International Electrotechnical Commission
Marque déposée de la Commission Electrotechnique Internationale

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

CONTENTS

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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,

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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);

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

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.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Figure 1 shows the key dependability activities important to the software life cycle identified for project implementation.

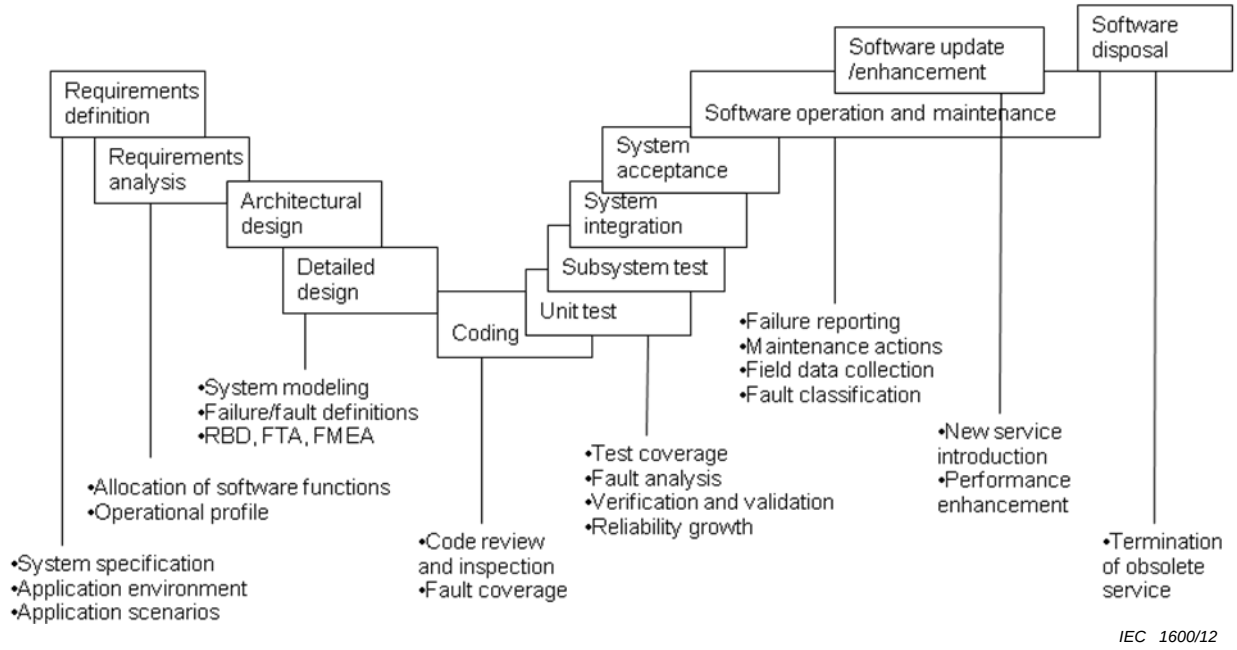


Figure 1 – Software life cycle activities

5.4 Software dependability attributes

Software dependability attributes are those characteristics inherent in the software by design. Specific application related performance attributes should be taken into consideration for incorporation in system design and construction to achieve combined hardware/software system dependability objectives.

The main software dependability attributes or inherent software dependability characteristics contributing to system dependability objectives include:

- *availability*: for readiness of software operation;
- *reliability*: for continuity of software service;
- *maintainability*: for ease of software modification, upgrade and enhancement;
- *recoverability*: for software restoration, following a failure, with or without external actions;
- *integrity*: for correctness of software data.

The specific application related performance attributes contributing to system dependability objectives include, but not limited to the following:

- *security*: for protection from intrusion in software application and use;
- *safety*: for prevention of harm in software application and use;
- *operability*: for robust, fault tolerant, and non-disruptive operation;
- *reusability*: for using an existing software for other applications;
- *supportability*: for sustaining system performance with logistic and maintenance resources;
- *portability*: for cross platform applications.

These inherent software dependability characteristics and specific application related performance attributes form the basis for software system design and application.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.5 Software design environment

The dependability management objective is to provide a well-balanced design environment for creativity within project budget resources, time schedule and delivery targets. Organizations associated with software development and provision of software services are user application oriented. Tailoring for software projects is needed to manage the allocation of available resources and seek out appropriate design options for effective implementation. The selection and adoption of applicable processes for engineering dependability into a specific software system is accomplished through the project tailoring process for effective dependability management. The recommended tasks for implementation of tailoring process are provided in 7.2. The opportunities for outsourcing design construction, software reuse, and application of commercial-off-the-shelf (COTS) software products for system integration should be explored.

The software design environment relies on an organized process to promote good design practices for error-free code generation, minimize mistakes in requirements definition, and assure test validation for software release. The cultural aspects in software management approach often adopt a capability maturity model concept for infrastructure development [8]. This is similar to the formal implementation of *Capability maturity model integration* [9] described in Annex C for software process management. Software development is a technical process following established software engineering disciplines and application guidelines. The software design environment and practice principles should be included in the organization's policy to establish mission and goals for dependability achievement.

Software design often engages the applications of CASE (Computer-aided software engineering) tools. An effective automated system provides the computational accuracy, traceability of data, configuration management, and a means for collecting the required measurements or input metrics to the models automated. Most data collection systems for field failure reporting, analysis and corrective actions are automated for the same reasons. Historical experience data on software products and services is an indispensable and valuable asset.

5.6 Establishing software requirements and dependability objectives

Software requirements should be established for the software life cycle stages. Applicable dependability activities should be identified for implementation relevant to each stage. Timing for implementation of relevant dependability activities is important. Dependability applications are time dependent and have extensive impact on system life cycle cost [10]. Project tailoring is essential for design trade-offs and constraints resolution. The software requirements and dependability objectives should form part of the overall software product specifications. The strategy for software dependability implementation is described in 5.8. The methodology for engineering dependability into software modules or units and for building software system architecture is addressed in Clause 6. Tailoring process is described in 7.2.

The dependability activities associated with the software requirements are application specific. They reflect the software design and implementation needs to deliver the required system functions for service performance applications. Systematic approaches for implementing relevant dependability activities throughout the software life cycle would ensure the achievement of dependability objectives. Specific dependability objectives are derived from the selection of key dependability attributes and the relevant quantitative metrics. Software dependability requirements can be formulated for specific projects by using the baseline information contained in Annex B.

The influencing conditions on combined hardware/software system dependability specifications are described in the system dependability specifications [11]. The following influencing factors affecting dependability achievement in software development should be considered:

- the organization's design culture, the capability maturity process, and the experience in software design, development and implementation (see Annex C);

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- understanding the application environments, user needs, and changing market dynamics for new platform or feature development for practical implementation;
- documentation processes such as failure reporting, data collection, software configuration management for control of software versions and maintenance of experience data records;
- application of software design rules for fault avoidance by controlling the design processes to optimize software performance in software complexity, program complexity, and functional complexity;
- effective use of applicable software methods and tools such as structured design, fault tolerance, design review [12], and software fault management to enhance reliability growth;
- selection of appropriate higher order of programming languages more suitable for specific software structured development;
- established requirements for qualification and measurement of software dependability characteristics.

5.7 Classification of software faults

Software faults could be classified as specification faults, design faults, programming faults, compiler-inserted faults, or faults introduced during software maintenance.

Classification of software faults provides a means for capturing and grouping relevant software fault information. The classification process helps software designers to discover unusual fault patterns for corrective actions. The objective is to eliminate the recurrence of the class of similar faults.

The *orthogonal defect classification* (ODC) [13] is a method used in software engineering for analysis of software fault (defect) data. The ODC addresses the causal effects of quality issues concerning software design and code in a procedural language environment. A defect is a non-fulfilment of a requirement related to an intended or specified use of the software. In this context, a fault due to the inability of the software to perform its required functions exhibits the characteristics of defect attributes in the ODC scheme. Defect attributes are the signature of a defect containing relevant information related to the software fault. The ODC method captures the software fault information of the defect attributes for analysis and modelling. The analysis of ODC data provides a valuable diagnostic method for evaluating the maturity of the software product at various stages of the software life cycle. The ODC can also be used to evaluate the process by analysing the types of triggers to identify specific technical needs to stimulate the missing triggers. The causal analysis of fault (defect) data presents a means for software fault reduction and reliability improvement.

The ODC defect attributes are classified as *Activity*, *Trigger*, *Target*, *Defect Type*, *Defect Qualifier*, *Source*, *Impact*, and *Age*. They are normally collected and analysed during software development to benefit design improvement. The information on defects is available at two specific points in time. When a fault is found, the circumstances leading to the fault exposure and the likely impact to the user are generally known. When a fault is closed after the fix is made, the exact nature of the fault and the scope of the fix are known. ODC categories capture the semantics of a fault (defect) from these two perspectives. By defining the *Activities* during the development process and their mapping to the ODC *Triggers*, the ODC provides valuable insights and customized the fault (defect) information for the software development organization.

The set of defect attributes is summarized: a) when a fault is found which is known as *opener section*, and b) when a fault is fixed which is known as *closer section*. ODC is most useful in mature software organizations where extensive data are normally collected and analysed for software product improvement.

Annex D presents a summary of classification of software defect attributes.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.8 Strategy for software dependability implementation

5.8.1 Software fault avoidance

Software codes are generated to produce a software product. A mistake made during software design and coding could manifest itself when triggered, to become a software fault leading to a system failure. Since faults are the main cause of system failures, preventing faults from being introduced during design, such as code review and removing the residual faults that had escaped detection by testing, are common approaches to lessen the existence of fault problems for the software life cycle. The recommended software fault avoidance strategy includes fault prevention and fault removal.

a) Fault prevention

- Establish fault prevention objectives in software engineering disciplines.
- Initiate requirement specifications review.
- Conduct early user interaction and refinement of the software requirements.
- Introduce formal methods where applicable and practicable.
- Implement systematic techniques for software reuse and assurance for application.

b) Fault removal

- Detect and eliminate the existence of software faults by testing.
- Conduct formal inspection on finding faults, correcting faults, and verifying the corrections.
- Perform corrective and perfective maintenance actions during software in-service operation.

5.8.2 Software fault control

Software faults are difficult to detect and fault removal can be achieved by various means including rigorous software testing and inspection. Exhaustive testing of software is often limited by time and cost constraints in project management. Dependability assurance based on testing alone does not guarantee complete fault elimination. Software fault control employs fault tolerance and forecasting methods to minimize the manifestation of latent software faults or bugs that can still exist after the software release for use. The recommended software fault control strategy includes fault tolerance and fault/failure forecasting.

a) Fault tolerance

- Establish methodology for fault confinement, fault detection and fault recovery.
- Implement software design diversity and fall-back schemes.
- Introduce multi-version programming techniques.
- Implement self-checking programming techniques.

b) Fault/failure forecasting

- Establish fault/failure relationships in operational environment.
- Establish data collection system to capture relevant data.
- Conduct reliability growth testing where applicable.
- Develop and implement relevant reliability models for fault/failure estimation.
- Refine forecasting techniques for time projection of software version release.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

6 Methodology for software dependability applications

6.1 Software development practices for dependability achievement

The capability maturity in software development reflects an organization's ability to develop software with consistency and dependable products for intended applications. The following fault avoidance and fault control techniques are recommended for incorporation where applicable in software development:

a) standardize methods for high-level architectural design, detailed design, coding and testing, and documentation to facilitate communications and fault avoidance;

b) develop modular designs for software units and subsystems with well-defined software functions and interfaces by building simple, separate and independent software units to facilitate design interaction, maintenance, error traceability, fault mitigation and bug removal;

c) use design patterns, which are general reusable solutions of well-tested software, as templates for solving software design problems to speed up the development process;

d) institute formal design methods where appropriate for control and documentation of software design and development process;

e) utilize *software reliability engineering* [14] techniques for software reliability assessment and enhancement [15];

f) reuse software available from software library on well-tested software units and subsystems for similar application and operational profile to reduce development cost and time, and minimize new design fault introduction;

g) develop regression testing methods to ensure functionality of existing software as new functionality is introduced or fault removal is performed;

h) testing software units and subsystems to verify low-level design functions and validate integrated high-level design architectural system performance for progressive bug removal to prevent fault propagation;

i) conduct inspections and reviews of software design requirements, software codes, user manuals, training materials, and test documents to detect and eliminate as much as possible mistakes; different review teams for comparison of results should be considered and employed where practicable;

j) control change to reduce fault occurrences such as version and change control process in software configuration management;

k) analyse root cause problems and implement appropriate corrective actions for continuous software improvement;

l) establish data collection system for knowledge base capture of software faults and performance data history.

6.2 Software dependability metrics and data collection

Software dependability metrics are measures of dependability characteristics of a software system. The measurement of metrics provides a quantitative scale and method to determine the value of a specific characteristic associated with the software system. These industry standard metrics are obtained either by direct measurement or by deduction. They are used for software system performance measurements. The following software metrics are de facto industry standards for application. They should be considered where appropriate for software system dependability assessment.

a) *Availability*: provides a measure of up time over the duration of system operation.

b) *Failure frequency*: provides a measure of the number of failure over the duration of system operation.

c) *Time-to-failure*: provides a measure of the failure-free time period.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

d) *Restoration time*: provides a measure of the time for restoration of a system from a failed condition (down state) back to normal operation (up state).

e) *Fault density*: provides a measure of the number of faults contained per kilo source lines of code (KSLOC) or per function point and is used for software reliability assessment.

f) *Function point*: provides a measure of the functional size of application software for software project planning by means of *function point analysis* method [16].

g) *Code coverage*: provides a measure of the degree to which the source code and the logical branches of a software program that has been systematically tested; *code coverage* is an indicator on the thoroughness of software testing, it is used to represent *fault coverage* that indicates the percentage of faults detected during the test in code execution.

h) *Fault removal rate*: provides a measure of the number of faults detected and corrected in a software product for a defined period of time or software execution duration; *fault removal rate* is used in reliability growth to establish reliability improvement trend.

i) *Residual faults in software*: provide a measure of the estimated number of bugs still remaining in the software product after testing for bug removal.

j) *Time for software release*: provides a measure of the estimated time for software product release schedule based on established criteria on an acceptable level of bugs still remaining in the software product for software project management.

k) *Software complexity*: provides a measure of the degree of difficulty for design and implementation of a software function or a software system; other complexity measures based on the complexity concept include program complexity, functional complexity, operational complexity; complexity-related metrics are used as inputs for reliability assessments and prediction models.

There are numerous metrics used for various reasons during the software life cycle. Software metrics can be grouped into three general categories to facilitate data collection.

1) *Fault data metrics*: capture the software problem reporting data for measuring the impact of the faults and the efficiency of the reporting process to improve software maintenance.

2) *Product data metrics*: capture the software product information by categorizing the size, functionality, complexity, location of use, and other characteristics to facilitate the experienced data as inputs to benefit new product development. The metrics provide performance history and data, and information of various software product groups.

3) *Process data metrics*: capture the software restoration process information and conditions at the time of fault detection and removal for reliability model inputs in reliability prediction.

The data collection process is critical for measuring software dependability attributes and performance characteristics. An effective data collection system should be practical for implementation. The amount and types of data should be relatively simple to collect, easy to interpret for data analysis, and useful for software dependability assessment, improvement and enhancement. The data collected is used to determine system reliability trends, frequency and time duration needed for software maintenance, response time for service calls, degraded performance restoration and maintenance support requirements.

Annex E presents examples of software data metrics obtained from data collection.

6.3 Software dependability assessment

6.3.1 Software dependability assessment process

The objective of software dependability process implementation is to ensure software system maturity in development and dependability achievement. The assessment process is the enabling mechanism to ensure verification of software requirements and validation of software dependability in system performance results. The dependability assessment process incorporates crucial software engineering activities adopted from established industry

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

practices [14]. The following process for conducting software dependability assessment is recommended:

- identify user needs and system performance objective and develop dependability specification;
- establish software operational profile;
- allocate applicable dependability attributes;
- perform dependability analysis and evaluation to determine options and possible solutions;
- conduct software testing and measurements;
- conduct software verification and software system validation;
- perform software reliability growth and forecast improvement trends;
- evaluate assessment results and feedback.

The software dependability assessment activities are described in the following sub-clauses.

6.3.2 System performance and dependability specification

The purpose is to identify the system performance objective for development of a system dependability specification [11] if not provided by the customer or user. The following process is recommended:

- identify the system performance scenario and application environment;
- identify the relevant performance influencing factors;
- identify the system boundary and interfaces with external interacting systems;
- identify the relevant system performance attributes;
- identify the system architecture, hardware/software configuration;
- identify the interoperating hardware and software functions of the system configuration;
- characterize and quantify the dependability attributes of the relevant hardware and software functions; including availability, reliability, and recoverability associated with maintenance support criteria.

Documentation of the system dependability specification should include the following data as part of the system specification:

- system identification;
- system performance objective;
- system operational profile;
- system dependability performance targets;
- system configuration;
- system functions;
- dependability requirements for each function;
- system maintenance support requirements.

For software system, it is essential to consider the operational profile that affects the execution time of software functions for on demand applications. A functional *reliability block diagram* [17] can be constructed to represent the hardware/software system. The functional blocks created would facilitate allocation of the respective software dependability metrics to each software function established according to the software system architecture and software configuration.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

6.3.3 Establishing software operational profile

An operational profile is the sequence of required activities to be performed by the combined hardware/software system to achieve its mission or service objective. The system performance is highly dependent on the environment in which the system operates. The environment can affect hardware physical changes, but cannot affect the software functions delivered by the execution of software programs in system operation.

The development of an operational profile is a quantitative characterization on how the software is being used. Operational data and relevant information are usually gathered through customer surveys and gained by field service experiences. The following are recommended processes for development of an operational profile:

a) determine the customer profile by establishing the needs and types of customers, such as an organization or an individual intended to acquire or purchase the software system;

b) establish the user profile on the different types of users, such as a person or an employee of an organization, or interacting software application systems, operating or using the software system for specific applications;

c) define the system-mode profile on how the system is being operated and in what sequence or order expressed in terms of modes of operation, such as software testing for upgrade maintenance, or normal batch data processing in executing the software system;

d) determine the functional profile by evaluation of each system mode for performance functions and service features, such as create e-mail message or address look-up in meeting the software functional requirements;

e) determine the operational profile based on the functional profiles established for system performance functions;

f) determine the information profile by collecting software application data at software development life cycle.

The functional profile is a user-oriented view of system capabilities. From the developer's perspective, functional profile represents the system operations that actually implement the required functions. From a dependability perspective, the operational profile is a set of different system operating scenarios and their probabilities of occurrence. The operational profile provides the needed inputs for development of test cases to simulate software system field operations and specific applications of the software functional features usage. The execution of test cases for software testing provides valuable information and data capture for estimation of software reliability in field operation and validate the provision of maintenance functions and efficiency of maintenance support performance.

6.3.4 Allocation of dependability attributes

Allocation of dependability attributes and measures for software system is based on the concept of modelling system architectural functions to reflect the requirements of the system dependability objective. The initial value assignment of applicable dependability metrics such as reliability and availability, are most likely based on experience data. These metric values are further refined through iterative analysis and evaluation process. The apportionment of reliability and availability values to the various software subsystems and functional units are assigned according to their complexity, criticality, estimated achievable reliability or availability performance targets, and other influencing factors relevant to the allocation process.

The development of a system model for software differs significantly from hardware due to its inherent operating characteristics. For each mode in system operation that involves the software program functions as configuration items, different set of constituent software units are being executed. Each mode has a unique time of application associated with the software unit execution duration on demand in system operation. This indicates the time duration of each system mode. The software system modelling includes the number of lines of source code in each software unit, the code complexity and other information pertaining to software development resources, such as programming language and design environment. They are

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

used to establish the initial failure rate for reliability or availability prediction of the software configuration items.

6.3.5 Dependability analysis and evaluation

The following dependability analysis and evaluation activities are needed to support software system development. The process is iterative for optimization of dependability design requirements to meet system performance objective. Availability/reliability modelling is used for analysis and evaluation of the software time-dependent performance functions.

a) Modelling availability/reliability functions

A simple approach to analyzing the availability or reliability of a system comprised of hardware and software is to form a structural model of the system. A functional availability/reliability model for the combined hardware/software system consisting of functional blocks can be constructed using the *reliability block diagram* (RBD) technique [17]. The model is decomposed into separate subsystem models representing the constituent hardware and software elements of the system. fault tree analysis (FTA) [18], Markov chains [19] and Petri nets [20] are also useful for system availability/reliability model development. For example, FTA can be effectively used to model system reliability with dynamic gates to determine hardware and software availability/reliability functions for trade-offs and improvement [21]. It should be noted that RBD and FTA are logically equivalent. RBD focuses on success; FTA on failure.

The hardware subsystem availability/reliability model consists of all hardware elements of the system with the availability/reliability functional blocks constructed as appropriate to represent the relevant hardware subsystem structure and redundancy configuration. This is to facilitate prediction of failure rates of individual hardware components and for hardware subsystem availability/reliability determination according to prediction techniques. There can be one or more hardware subsystems servicing different functions in the system configuration.

The software subsystem reliability model is constructed using software units as building blocks to deliver software program functions. A software unit is the lowest level of a configurable software item. Software units do not fail independently as with the hardware components. Software codes are virtual entities not subject to physical changes. Software units fail in association with the system operational profile, which affects the configuration scheme of the software reliability model structure. Modelling software reliability needs to incorporate the operational profile information in developing the software configuration structure. The software subsystem program can consist of one or more constituent software units to deliver the required functions. A software subsystem program residing in a host hardware subsystem is configured to form a software configuration item. The interoperation and mutual dependency of the software subsystem and its designated hardware host are needed to deliver specific subsystem software functions for system operation. There can be several combined software and hardware subsystems servicing different functions in the entire system configuration. Annex F presents an example to illustrate the interactions of combined hardware/software reliability functions to derive the system failure rate for reliability assessment.

Availability assessment of combined hardware/software system should first establish the interactions of combined hardware/software reliability functions prior to derivation of system availability functions. The total system downtime, or time for restoration for the duration of system operation, is required for system availability assessment.

b) Determining reliability of software functions

Software failures can occur during system operation. Determining the software failure rates for use in reliability modelling requires that the software be treated as a subsystem, which resides in a host hardware subsystem, configured to form a software configuration item. The software subsystem can perform one or more of its required functions. A function is a capability of the system to deliver a required service from the end user's perspective. The function can be accomplished by a software configuration item, so that the required service

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

function is recognized by version control. A software unit designed to perform exactly one single function can be a configuration item. A software subsystem program requiring multiple software units to perform a single function can also be a configuration item. A software subsystem can consist of several software programs to deliver a set of related functions. Each of these software programs is a configuration item by definition. The concept of software configuration item is viewed from a software design version control perspective. Software configuration item is essential for tracking design changes. Each design change is assigned a version issue for identification. Referencing software version is necessary for tracking effectiveness of software maintenance upgrades. Reliability growth trend is established by performance improvement indication with the system running the new version replacing old version. Delivering the required functions in system operation is the challenge to meet system reliability objective.

The software functions that comprise a system are related in a timing configuration and a reliability topology.

Timing configuration is a concern when the various functions are active or inactive during a specific time period in system operation. The major timing relationships among software functions are concurrent or sequential. Functions are concurrent if they are active simultaneously. The functions are sequential if they are active one after the other. It is also possible for function times to partially overlap, resulting in a hybrid concurrent/sequential timing configuration. Concurrent active software functions are found in systems that are serviced by more than one central processing unit, such as in a multiprocessing system or a distributed system. Run time references are identified as execution time, system operating time, and calendar time.

Reliability topology concerns the number of functions in the system that can fail before the system fails. Reliability topology is the relationship of an individual function failure to the failure of the aggregate system. Software functions are generally related in a series topology. The failure of one function would cause in the failure of the software system. Software fault tolerant design can be used to protect a system in the event of failure of one or more functions.

c) Run time reference of software function

Software failure rate can be expressed with respect to three different time frames of reference.

- *Execution time* is the central processing unit run time, which accumulates when the software program is executing instructions. The execution time is used to determine the execution-time failure rate of the application software subsystem.
- *System operating time* increments whenever the hardware/software system as a whole is operating. This is used to determine the operation-time failure rate of continuous operation software subsystem.
- *Calendar time* is the time period used for project planning and scheduling purpose. Calendar time is always incrementing.

During system operation, software programs do not always run continuously. Some programs can time-share a single central processing unit. Multiple central processing units can also be present, allowing the program executions to overlap. The failure rates of the various programs need to be combined to arrive at an overall software failure rate. They are converted into a common time frame of reference in system operating time. This is the same time frame used to express hardware failure rates to facilitate failure rate determination of combined hardware/software system.

d) Criticality of software function

Software functions are often used for control of a critical system where a failure can cause catastrophic consequences. The criticality of software functions should be identified early in system concept definition and evaluated during software architectural design of the system.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

The criticality of functional failures should be classified in the system specifications, such as critical, major, or minor based on established criteria; and verified by analysis in system reliability performance.

The level of risk associated with the critical software function can be determined and evaluated by means of risk assessment techniques. Project risk management [22] should focus on fault prevention and fault tolerance where the severity of failure consequences can be mitigated.

Fault tree analysis [18] can be used to identify the possible causes of an unwanted top event. It is used to investigate the potential faults and their causes, and quantify their contribution to system unavailability. Fault tree analysis is a top-down technical approach, where the starting point is from the top-level software subsystem program and following it through the software hierarchical structure to the lowest software unit. The potential faults can be individually identified and assessed on their respective probability of failure occurrences. The quantitative assessment provides an indication or magnitude of the criticality of the software function. This is of interest for design optimization and fault avoidance.

Failure mode and effects analysis [23] can be used to determine possible failure modes and faults in the software units and their effects on the next higher-level subsystem of the software hierarchical structure. Failure mode and effects analysis is a bottom-up technical approach. It can be extended and used for criticality analysis of software functions. The criticality analysis combines quantitative value of the likelihood of failure occurrence and qualitative information on failure severity to support design trade-off and fault mitigation.

Other system dependability analysis techniques [24] are used for software decomposition and system simulation. They can be selectively used for detailed reliability and maintainability assessment of software functions in combined hardware/software systems.

Software integrity level is a value representing project-unique characteristics that define the importance of the software to the user. Examples of project-unique characteristics include software complexity, criticality, risk, safety level, security level, desired performance, and reliability. The software integrity level is determined by classification of criticality of the impact of failure consequences and their associated frequency of occurrences [25]. The criticality of software functions is also application specific. For safety-related systems, the safety-integrity level should be defined and incorporated for software system development to meet functional safety requirements [26]. For security-related systems, specific system security requirements [27] should be incorporated.

6.3.6 Software verification and software system validation

Specification of software tends to be much more complex than specifying physical hardware systems such as machinery and electric/electronic systems. The "correctness" of software is of primary concern. The verification process [2] is to determine that the requirements for the software are complete and correct as applicable to the software life cycle stages. The validation process [7] is to determine that the software system performance and services are conformed to the customer/user requirements. Appropriate enabling systems, such as test equipment, facilities and supplementary resources, are required to support the implementation of the verification and validation processes. The enabling system does not contribute directly to the performance functions of the software or the system under test during operation of its life cycle stages.

a) Software verification

The software verification process is to confirm that the specified requirements are fulfilled by the software system. The following verification process activities are recommended:

- define strategy for software verification;
- develop a verification plan based on software system requirements;

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- identify the constraints and limitations associated with the design decisions;
- ensure that the enabling system for verification is available and associated facilities and testing resources are prepared;
- conduct the verification to demonstrate compliance to the specified design requirements;
- document the verification results and data;
- analyse the verification data for initiation of corrective action.

b) Software system validation

The software system validation process is to provide objective evidence that the system performance meets customer/user requirements. The following validation process activities are recommended:

- define strategy for validation of the services in the operational environment and achieving customer/user satisfaction;
- prepare a validation plan;
- ensure that the enabling system for validation is available and associated facilities and testing resources are prepared;
- conduct validation to demonstrate conformance of services to the customer/user requirements;
- document the validation results and data;
- analyse, record and report validation data according to the criteria defined in the validation strategy.

6.3.7 Software testing and measurement

a) General consideration for testing software

Software testing is the process of executing a program or a set of coded instructions with the intent of verifying software functions and finding errors. The software testing objectives vary with project needs, software product availability, software maturity status, and scheduling for testing during the software life cycle. When planning for software testing the following should be taken into consideration.

- Test planning is essential and should be documented to describe the test objectives, test process, procedures and resources.
- Software testing requires knowledge and skills and good testing practice. Although many of the test routines and tools have been automated and widely deployed in industry, good testing techniques demand the skills, experience, intuition and creativity of the tester to achieve dependable results. Maintaining test record is important to provide accuracy and traceability of test data.
- Testing is more than just debugging the software program to locate faults and correct errors. Testing is also used in software verification and validation, availability and reliability measurement.
- Test efficiency and process effectiveness are criteria for coverage-based testing techniques. Test automation can expedite software test time and reduce project cost. The selection of appropriate test tools, the training and support costs associated with the test tool acquisition should be taken into consideration.
- Testing may not be necessarily the most effective means to improve software quality unless appropriate follow-up actions are taken. Alternative methods, such as code inspection and code review should be considered.
- Software testing is only part of the software reliability growth and improvement process. It needs collaboration of other assurance efforts to achieve dependability goals.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- Complete testing may not be feasible or practical, and often time/cost prohibitive. Software complexity influences the extent of test completeness. The complexity problem often limits the tester's ability to detect and remove bugs by the testing process.
- Latent software faults do exist in software after its release for use operation. Software reliability prediction provides a means to estimate the test time required on reducing the residual software bugs to an acceptable number before the software version release.
- Testing beyond unit testing should be performed by testing teams that are separate and independent to the teams developing the software.

b) Types of software tests

The following presents the types of software tests performed during the software life cycle.

- *Unit test*: testing of one software unit that can be compiled before it is integrated into the software program or subsystem. The software unit is tested to verify that the detailed design specifications for the unit has been correctly implemented.
- *Subsystem test*: testing of a subsystem software program consisting of one or more software units as a software configuration item to verify functional performance requirements.
- *Integration test*: testing of a software system in a hardware host as a whole consisting of integrated subsystems to verify functional operation, expose problems in software interfaces, hardware interfaces and interactions between the hardware and software, and validate reliability performance.
- *Reliability growth test*: testing of software in an iterative process to improve reliability through testing until failure, analysing failures, implementing corrective action on the existing software version for upgrade, and continuing the test with the new software version. Termination of reliability growth test is based on when the established software reliability target is met.
- *Qualification test*: testing to demonstrate that the software meets its specifications when integrated in its host hardware system and ready for use in its target environment. Before release of final version for software distribution, alpha and beta testing are often conducted for quality assurance purposes. *Alpha testing* is an in-house trial carried out by software developer before release for external users. *Beta testing* is a field trial carried out by a limited number of users in its intended application to seek user feedback experience information.
- *Acceptance test*: testing of a software system to validate that the customer's requirements are met. For acceptance testing of complex hardware/software systems where no prior information exists on similar systems, reliability growth and stress testing [28] should be considered as part of the acceptance test requirements.
- *Regression test*: testing of software that has been previously tested in an effort to uncover any maintenance errors introduced, new code being developed, improper configuration, or inadequate source control.

c) Testability of software

Testability is the ability of the software to be tested with minimum time and resources. Testability is a design characteristic that allows the software operational status to be determined effectively. The testability design characteristic also permits the process for faults detection, isolation and diagnosis to be performed efficiently. Design for testability should focus on structured design of the software function to enable testing. Modular design approach where each software function is independent of the other functions would facilitate testability in detection and isolation of faults. The approach would enhance maintainability of the software function by simplifying the process for software update or modification.

Self-test programmes, monitoring and control procedures can be designed and incorporated into a software system to perform self-testing of the system. The self-test function can be operated on demand or activated automatically by the programmes to facilitate maintenance and diagnosis of the system indicating its operational performance status. The self-test design

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

features should include the capability of false alarm detection and indication such as operator error and transient condition. False alarm is a warning reported by the self-test diagnostic management function indicating the existence of an operational fault when that fault does not exist. False alarms can be reduced or eliminated by a full and accurate diagnostic analysis and validated by the run-time diagnostic management process.

The structured design approach for testability involves a process for rationalization of the objectives for testing. The process analyses the software attributes and predicts the likelihood if there are any bugs in the software that can be revealed through testing. The analysis is used to optimize the testing process to determine how much testing is enough. It provides the means to manage test resources and determine the value or benefits of a specific testing approach.

d) Test cases

Test cases are developed based on the software specifications. Test cases are used to simulate actual software field operating conditions in which specific interest areas or potential problems could be encountered. A test case is a set of test inputs, execution conditions, and expected results developed for a particular testing objective; such as to exercise a particular software program path or to verify compliance with a specific requirement. A test case specification is the documentation for specifying inputs, identifying expected test results, and establishing execution conditions for the test item. An effective testing process includes both manually and automatically produced test cases. Manual tests cover the depth of finding software faults reflecting the developer's understanding of the problem domain and data structure. Automatic tests cover the breadth of fault investigations by executing the entire range of test values, including those extremes that manual tests might miss. The automatic test process engages the use of a test case generator to accept source code, test criteria, specifications, or data structure definitions as inputs to generate test data and determine expected results. Fault insertion test could be considered as one of the test cases in which a deliberate fault is introduced in one part of the software system to verify that another part reacts appropriately. The test results are used to determine probable fault conditions and facilitate software fault-tolerant design. Fault insertion technique is also used to test the coverage of the test program by counting the fraction of the inserted faults found.

Testing a software program is an attempt to make the software fail. It is important to note that any failed execution must yield a test case for inclusion in the software project's test suite. The most important aspect of a testing strategy is the number of faults that the test has uncovered as a function of time. This provides an indication of test efficiency.

e) Software measurement and metrics for project management

Measurement is the process of determination or estimation of quantitative values or metrics to facilitate effective project management. The metric data are obtained by various methods described as follows from different perspectives for software reliability prediction and system dependability performance improvement.

- *Design structure metrics*: measurement of design approach, complexity, and independence of the software design.
- *Design completion metrics*: measurement of the extent to which the software performs the specified functions completely.
- *Process management metrics*: measurement of the management, cost-effectiveness, and design trade-offs for the software based on the analysis results of the process data metrics and the relevant fault data metrics captured in the data collection process.
- *Product management metrics*: measurement of the characteristics of the software that are specific to the software product developed based on the analysis results of the product data metrics and the relevant fault data metrics captured in the data collection process.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Many of these metrics are used for inputs to software reliability model parameters for prediction and estimation where quantitative values are needed.

Annex G presents a summary of software reliability model metrics commonly used in industry practice.

6.3.8 Software reliability growth and forecasting

Software reliability growth is the condition characterized by a progressive improvement of a reliability performance measure of the software system with time. Software reliability improvement is achieved by design and the progressive reliability attainment is verified by means of reliability growth testing. Software does not fail if it is not executed to expose failures. A software program can only fail when it is executed. The software failures uncover faults, and the removal of these faults results in reliability improvement. Software reliability growth trends are based on the fault removal rates with respect to the cumulative software execution time. For scheduling purposes, execution time can be converted to calendar time to establish the software failure rates for reliability estimation. A reliability growth program [29] can be established for combined hardware/software system. The reliability growth models and estimation methods for assessments, based on failure data captured in the reliability growth program, are described in the statistical methods for reliability growth [30]. Typical software design improvement methods are provided in 6.4. Software reliability growth testing is presented as follows.

a) Software reliability growth testing

Reliability growth testing is performed to assess current reliability, identify and eliminate bugs, and forecast future reliability. The reliability values based on the bug counts uncovered and removed during the execution time period are compared with intermediate software reliability objectives. This is to measure reliability progress trends in the testing process to achieve software reliability targets.

Accelerated testing [31] has been successfully used to shorten product test time. This is accomplished through the application of increased stress levels or by increasing the speed of application of repetitive stresses to assess or demonstrate product reliability growth. For combined hardware/software systems, the stress application to software can involve test excursions through various operational scenarios. The objective is to verify system performance adequacy under simulated operating environments within practical test time and conditions.

b) Software execution environment

The software execution environment includes the hardware platform such as the hardware host system, the operating system software, the system generation parameters, the workload, and the operational profile. The operational profile is described in 6.3.3.

A *run* is the result of execution of a software program. A run has identifiable input and output variables. Software reliability testing is based on selecting a set of input variable values for a particular run. Each input variable has a declared data type representing a range and ordering of permissible values. An operational profile is a function associated with the probability of the input variable, which is used in statistical estimation for reliability growth.

c) Multiple software copies

The time on test during reliability growth testing can be accumulated on more than one copy of the software. The copies can run simultaneously to accelerate testing. This procedure permits multiple copies run time accumulation to speed up the testing process, especially to demonstrate achievement of high reliability targets. In this respect, the total amount of calendar time on test is reduced. The use of multiple copies can provide economic and scheduling advantages.

d) Software reliability forecasting

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Reliability growth for software is the positive improvement of software reliability over time, accomplished through systematic removal of software bugs. The rate of reliability growth depends on how fast the bugs can be uncovered and removed. A software reliability model applicable to growth conditions allows project management to track the software reliability progress through statistical inference to establish trends and forecast future reliability targets. Appropriate management actions can also be taken if the trend indicates negative.

e) Software reliability models

Measuring and projecting software reliability growth requires the use of an appropriate software reliability model that describes the variation of software reliability with time. The parameters of the reliability model can be obtained either from prediction based experience data, or from estimation of test data collected during system test. The selection and use of software reliability model should be validated. The estimation process is based on the times at which the failures occur with sufficient data sample for significant execution time accumulation. This is to establish a reasonable degree of statistical confidence to validate the reliability growth trends. The approach is to forecast software maturity and release targets.

Annex H presents typical examples of software reliability models used in industry practice.

6.3.9 Software dependability information feedback

Software dependability data collection is addressed in 6.2. The data collection activity is conducted for field tracking to assess the dependability of software performance operation in customer premises. This is to ensure and confirm that the accepted level of dependability performance in operation is sustained for the software deployment. The in-service field dependability information is collected together with relevant customer feedback information. The information is used to justify changes for new software requirements and initiate development of software new release.

Often times due to the dynamics of application environments and technology evolution, the decisions on software new releases are influenced by market competitions and driven by business strategies.

6.4 Software dependability improvement

6.4.1 Overview of software dependability improvement

Software dependability improvement can be achieved by improvement in software design, improvement through reliability growth testing, and improvement in software maintenance support performance for customer support services, including software enhancement effort.

Reliability is a key attribute of software dependability. Software reliability improvement through reliability growth testing is described in 6.3.8. The following subclauses provide practical approaches relevant to software reliability design and recommended techniques for software enhancement and implementation. The design objectives are focused on *testability* for ease of verification of software functions, *modularity* for independence of each software function to facilitate fault isolation and containment, and *maintainability* for ease of modification in software life cycle.

6.4.2 Software complexity simplification

a) Structural complexity

Structural complexity describes the logic paths for software module connection of software design. Each module unit could be programmed (by coding) to provide an executable unit of software function in the software structure. Structural complexity is related to the testability of program codes that affect fault detection, hence influencing the reliability and maintainability of the software architecture. The more complex the structure, the harder it is to test the software. The software design rules should establish a level of complexity to facilitate design for dependability.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

b) Functional complexity

Functional complexity describes the required functions that software module or segment of code in the unit must perform. Ideally, one module unit should be designed to perform one function to achieve simplicity with one set of cohesive inputs and outputs to facilitate software fault isolation and removal. In practice, both structural complexity and functional complexity should be considered for software design evaluation. Software design strategy on complexity is directly linked to the number of test cases needed for complete software verification.

6.4.3 Software fault tolerance

Fault tolerance is the software ability to continue functioning and preserve the integrity of data with certain faults present. Software fault tolerance design is to prevent software faults from causing system failure during system operation. Software fault tolerance is constructed to have a low probability of exhibiting common-mode failure from a number of diverse system designs including the following recommended practices.

- *Fault confinement*: software is written in such a way that when a fault occurs, it cannot contaminate portions of the software beyond the local domain where it occurred.
- *Fault detection*: software is written such that it tests for and responds to faults when they arise.
- *Fault recovery*: software is written that after detecting a fault, it takes sufficient steps to allow the software to continue to function successfully.
- *Design diversity*: software and its data are created so that there are fall-back versions available.

Fault tolerance exhibits the graceful degradation property that enables a software system to continue operating properly for a period of time in the event of failures. This is to prevent failures that would otherwise cause abrupt system outage or total breakdown. Fault tolerance is of particular importance for safety critical systems that depend on high availability system performance in the presence of faults or operating under adverse conditions. An example of fault tolerance design is the Transmission Control Protocol for Internet communications. This is a software protocol designed to allow reliable two-way communications in a packet switch network, even in the presence of communication links that are imperfect or overloaded. The fault tolerance design is accomplished by requiring the end points of the communication to expect packet loss, duplication, reordering and corruption, so that these conditions do not damage data integrity, and only reduce throughput by a proportional amount to sustain operation.

Multi-version programming method is a possible approach used for fault tolerance in design of critical systems and for improvement of software reliability in operation. The method engages multiple functionally equivalent programs that are independently generated from the same initial software specifications. The independence of separate programming effort would greatly reduce the probability of identical software faults occurring in two or more versions of the program. Implementation of these programs utilizes different algorithms and programming language. Special mechanisms are built into the software to allow these separate programs to be controlled by a voting scheme in the decision algorithm for program execution in application. The concept is based on the assumption that output from multiple independent versions is more likely to be correct than the output from a single version from a redundancy view-point. In practice, the improvement benefits of multi-programming effort would require justification of the additional time and resource requirements to warrant cost-effective implementation. The effectiveness of the method would also depend on assuring diverse fault characteristics between versions in software design and implementation.

6.4.4 Software interoperability

Software interoperability is the ability of diverse software systems to work together to exchange information and to use the information that has been exchanged. In an open system such as an IP network, it is important to achieve interoperability of diverse software systems to establish communication links. Failure of the communication link would affect dependability

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

in performance operation. One practical approach recommended to enhance interoperability in communication network is to incorporate a specific feature in the software system design to monitor the situation of the established communication. For example the “heartbeat” technology (signal processing and synchronization scheme) incorporating the monitoring feature is to send the “heartbeat” signal to each other when the communication link is established. If the link is broken or interrupted due to changes in the environment or any other causes, the software system will automatically attempt to seek and re-establish the link to maintain continued communication such that dependability in performance operation of the communication network is not degraded.

6.4.5 Software reuse

Software reuse is motivated by various reasons including proven history in performance operation, economy in time and cost savings, and proprietary products in business decisions.

Software reuse is the use of existing software to build new software. Reusable software is a reusable asset. The most well known reusable asset is code. Programming code written at one time can be used in another program written at a later time. The reuse of programming code is a common technique that attempts to save time and effort by reducing the amount of repeated work. Software *reusability* is the degree to which a software asset can be used in a different software system or in building other assets.

From a dependability perspective, the application of software reuse in projects and its *reusability* attributes should be controlled to achieve dependability improvement. Reusability is directly dependent on the software structure and modular design. For a software unit to be reusable it should be confined to perform only one function completely. This restriction is essential because if the intended reuse of the software unit is performing less than one function, or it is able to perform more than one function, it would be difficult to implement or to maintain for its intended reuse purpose. Deviation from such restriction would decrease the usefulness of the reusable software. Reusing software that does not perform exactly one function could have adverse effect on dependability due to the possibility of errors introduced into the software during implementation or maintenance.

The reuse of software should be implemented only if the functional requirements of the new software unit are in line with those of the reusable software for very similar application and operational environment. Otherwise, it would diminish the cost-effectiveness of the software reuse objective and possibly decrease reliability when implemented.

Reusable software should be well documented for traceability to facilitate configuration management of software assets. Commercial off-the-shelf (COTS) software products and systems should be treated as reusable software for varied multiple applications. Qualification testing for assurance purposes should be implemented to validate COTS software product/system performance and suitability to meet project application needs.

6.4.6 Software maintenance and enhancement

Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other software performance attributes, or to adapt the product to a modified environment. There are four main categories of software maintenance.

- *Corrective maintenance*: reactive modification of a software product performed after delivery to correct discovered problems.
- *Adaptive maintenance*: modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.
- *Perfective maintenance*: modification of a software product after delivery to improve performance or maintainability.
- *Preventive maintenance*: modification of a software product after delivery to detect and correct bugs in the software product before allowing further propagation into real failure occurrences.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

The key software maintenance issues are both managerial and technical. The management issues include alignment with customer priorities, maintenance resource planning and allocation, skill training of maintenance personnel, contract maintenance work, and customer satisfaction survey feedback. The technical issues include incident reporting, technical problem resolution, impact analysis, standardization of application procedures and testing practices, software maintainability assessment and test efficiency measurements.

Software enhancement is part of the software evolution process. Software system in field operation is noted for its increasing complexity due to modification and enhancement work done to meet customer needs, continuous changes in maintenance support strategies due to competitive service offerings, and the need to develop the skills and techniques to accommodate the changing business environments. The extent and achievement of software maintenance and enhancement effort should be verified, validated and documented. The specific resources needed for software maintenance should be part of the dependability assurance strategy.

6.4.7 Software documentation

Software documentation is the written text and information of the design and application documents associated with the software product. Documentation is an important part of software engineering. The major categories of software documentation include the following.

a) Architecture and design documentation

The documentation presents an overview of the software product and its relations to application environment and construction principles to be used in the design. The software design document is a comprehensive software design model providing detailed information.

- The *data design* describes the structures of the software. Attributes and relationships between data objects dictate the choice of data structures, which impact the structural complexity of the software affecting dependability in performance operation.
- The *architectural design* uses information flow characteristics, and maps them into the program structure, which impacts the modularity of the software design affecting reliability of software module design. Recommended practice for architectural description [32] should be implemented.
- The *interface design* describes the internal and external program interfaces as well as the design of human interfaces including hardware interfaces and hardware drivers. Internal and external interface designs are based on the information obtained from the design analysis, which affects redundancy schemes and reliability requirements of systems, software, and hardware module designs and configurations.
- The *procedural design* describes the structured programming concepts using graphical, tabular, and textual notations. These design media enable the designer to represent procedural details that facilitate translation to code which affects consistency in programming practices and reduction in coding errors introduction.

b) Technical documentation

Technical documentation includes code, algorithms, interfaces and additional text to describe various aspects of the software products intended operation. The documentation should be comprehensive but concise in writing the source code to facilitate software maintenance and update. In-code commenting is a form of technical documentation, where the commenting includes brief explanatory comments lines added to the code that are recognized by the compiler to be comments only and not part of the code for execution. The purpose of in-code commenting as a software documentation practice is to increase reusability and maintainability. This would facilitate code review and inspection, code update and modification, and enhance software integrity and reliability. Technical documentation can also include where needed and applicable to the project, such as requirements specifications, test plans and procedures, technical reports and relevant data.

c) User documentation

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

User documentation [33] includes manuals for the end users, system administrators and support personnel. It is aimed at assisting the end user application of the software products. The documentation describes how the software can be used in its application environment. User documentation also describes the features of the software product, and assists the user in realizing these features for application; including software release and version control information, trouble-shooting guidelines, safety instructions, warnings and restrictions on the use of the software, and help instructions. Sometimes on-line help is available to promote user-friendly access and service contact to achieve customer satisfaction.

d) Marketing documentation

Marketing documentation includes promotional materials to encourage casual observers to learn more about the software product. Web access and customer care centres are common service provisions in today's competitive market environments for obtaining software products and application information. Customer focus is essential in developing a marketing strategy. Marketing documentation is but one of the many approaches for dissemination of information. Marketing documentation can include information addressing specific dependability values and related issues for appropriate software applications such as software reuse and modification for specific applications.

6.4.8 Automated tools

Automated tools are useful for routine data processing, computational analysis, and comparison of evaluation results. There are broad ranges of automated tools available in the market to meet most application needs for modelling, analysis, and knowledge-base data to support all forms of dependability assessment. The selection and application of appropriate tools for specific project tasks would require the knowledge and experience of the dependability engineer or practitioner. Automated tools are enabling systems that could help routine computational work to improve productivity. The validity and accuracy of these automated tools should be investigated prior to commitment for project application. The supportability of these tools should be determined before tool acquisition. Automated tools are used for software development and testing applicable to dependability enhancement and reliability growth improvement. They form an essential part of the enabling system for application in software verification and software system validation.

6.4.9 Technical support and user training

Technical support is a range of services providing assistance with the software products in use. The objective is to help the user solve specific problems with product operation or application. Technical support takes various forms including telephone query, online service, e-mail, remote access repair and on-site visit for problem solving. There are increasing growth and use of outsourced call centres by technology product development organizations for business, economic and geographical reasons to facilitate real-time response to technical support services. These call centres serve as centralized technical support for a broad range of technology products such as computer systems including software requiring technical assistance around the clock with worldwide toll-free user access. Technical support services form part of the maintenance support to sustain product operability and reliability performance contributing to dependability improvement.

Software user training is an important aspect of software dependability improvement. The objective is to enhance or familiarize the skill level or understanding of the software product applications from the users' perspective. Software user training takes various forms including online access of the product supplier's tutorial database, call centre assistance, dedicated technical expert service to address unusual problems encountered.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

7 Software assurance

7.1 Overview of software assurance

Software assurance is the planned and systematic set of activities that ensure that software life cycle processes and products conform to requirements, standards, and procedures. The capability maturity models [8, 9] are common management tool recommended for implementation of software assurance programs in software development organizations. There are also extensive documented software assurance methodology and procedures for software development and applications [34].

Software assurance generally involves the technical disciplines of quality, reliability, safety and security associated with software product development and system operation. The software assurance process is to plan, develop, maintain and provide grounds for confidence and decision making. The assurance life cycle [35] is conducted for conformity assessment purposes throughout the system life cycle on software products to meet applicable safety, security, dependability and other objectives. The assurance case [36] studies are claim records on process performance and the physical properties and functional characteristics of the software system audited for proof of conformance to system specifications. Software assurance engages in risk assessment, verification and validation testing, documentation and maintenance of audit records as objective evidence. Software assurance utilizes relevant project-based measurement data to monitor the software product and relevant process for possible improvements.

Software dependability emphasizes software reliability as an intrinsic part of software assurance through implementation of software reliability engineering process [37]. Software dependability and quality are pre-requisites for achievement of safety and security in system operation.

7.2 Tailoring process

Tailoring is a project management activity to assure timing and action, and appropriate allocation of resources to meet project needs. The tailoring process can be effectively employed for implementation of software assurance activities. Tailoring is often used in short-term projects to enhance or sustain system operation where the project requirements and constraints are more restrictive than starting a new development project. The following tasks are recommended for implementation of the tailoring process.

- Identify and document the circumstances that influence tailoring, such as operating environment, project size and complexity, project schedule and budget, resource availability, safety, security and integrity issues, legacy issues, and standards conformance requirements.
- Identify input requirements for decision-making.
- Establish project objectives and plan the tailoring process for implementation.
- Select appropriate life cycle stages applicable for tailoring to achieve intended results.
- Document tailoring results to facilitate review of effectiveness and improvement.

7.3 Technology influence on software assurance

Software technology has provided numerous advancements for efficient software development resulting in versatility and economic advantages for software applications. Software assurance has traditionally been focusing on software quality and reliability improvements from a product development perspective. Recent cyber attacks in software operations have become more frequent, more prominent and increasingly sophisticated. They affect not only the software developers using the COTS software but also cause significant time-lost problems to software system operators and users of the end products. The entire situation has become a chain reaction propagated by unknown viruses, stealthy intrusions

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

and cyber attacks creating a complex and dynamic risk environment for IT-based operations that are software dependent.

Software assurance is critical to organizations involved in safety, security and financial transactions in view of the vulnerability in software applications. Software assurance encompasses the development and implementation of methods and processes for ensuring that software functions as intended while mitigating the risks of vulnerabilities, malicious code, faults or errors that could bring harm to the end user. Software assurance is vital to ensuring the security of critical IT resources. With the rapidly changing nature of threat environment, even the highest level of quality software is not impervious from cyber intrusions if the software is improperly configured and maintained. Managing the threats in cyberspace requires a layered approach on security prevention and collaboration. The developers build more secure and robust software, the system integrators ensure that the software is installed correctly, the operators maintain the system properly, and the end users using the software in a safe and secure manner.

This leads to organizations involved with software to redefine software assurance for their operations. For example, *software assurance* can be interpreted as the “level of confidence that software is free from vulnerabilities, either intentionally or unintentionally designed into the software or accidentally inserted at any time during the software life cycle, and that the software functions in the intended manner” [38]. Software assurance should provide a reasonable level of justifiable confidence that the software will function correctly and predictably by a manner consistent with its documented requirements. The assurance objective is to ensure that the software function is not compromised either through direct attack or through sabotage by maliciously implanted code.

Depending on specific applications, the level of confidence in software assurance addresses:

a) *trust-worthiness* – that no exploitable vulnerabilities exist, either maliciously or unintentionally inserted;

b) *predictable execution* – that software functions when executed as intended will provide justifiable confidence;

c) *conformance* – that planned and systematic set of multi-disciplinary activities to ensure software processes and products conforms to requirements, standards and procedures.

The challenges identified for software assurance include:

1) accidental design mistakes or implementation errors that lead to exploitable code vulnerabilities;

2) the changing technological environment which exposes new vulnerabilities and provides the cyber attackers with new tools for exploitation;

3) malicious insiders and outsiders who seek to do harm to the developers or the end users.

The first challenge is accidental and unintentional. The second and third challenges are intentional and deliberate by the cyber attackers. The countermeasure is to manage risks associated with these challenges through software assurance best practices.

7.4 Software assurance best practices

There are software technology and software assurance forums [39] that involve government, industry, academia and user participation in implementation of software assurance best practices. The recommended software development practices are identified in 6.1. The recommended software assurance best practices are presented as follows:

a) establishment of software assurance policy to guide software development and process implementation;

b) training on software product related technology applications and the use of reference resources;

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

c) use of common software architecture design platform to facilitate diverse software product development;

d) implementation of software life cycle processes;

e) initiation of software assurance case studies for risk assessment where warranted and appropriate;

f) established common criteria for verification and validation for software qualification and conformance;

g) configuration management control of software version release;

h) established software performance and fault tracking and data collection system for software design and process improvement;

i) established customer help centre to facilitate users service support and software product application.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annex A
(informative)

Categorization of software and software applications

A.1 Categorization of software

A.1.1 Software categories

The categories of software include relevant software development products and data that are produced by the software engineering process. Software characteristics and application environments are influencing factors that affect the dependability processes in software design and implementation. The categorization scheme presents an orderly combination of views and categories related to software [40].

The category is represented by the grouping of software based on its attributes or characteristics. Emphasis is placed on software dependability related issues to facilitate development and applications. Typical examples of such groupings include as follows.

A.1.2 Characteristics

- *Operation mode* – categories defined by specific processing technique or type adopted by the software system such as real-time, batch, time-shared, parallel and concurrent processing. A real time system should focus on response time. Time-shared software should focus on interface specifications.
- *Scale of software* – categories defined by the size (e.g. KSLOC) or complexity (e.g. data flow) of the software and interpreted as small, medium or large; simple or complex. Complex or large software should be decomposed or broken down to smaller sizes to facilitate project control, stepwise testing and integration.
- *Stability* – categories defined by intrinsic evolutionary aspect or stability in terms of software system characteristics such as continually changing, incremental change, or unlikely to change. Continually or incremental changes of software require interface specifications that allow flexibility and stability after each change. Special development models such as spiral model and the waterfall model are often used.
- *Software function* – categories defined by the type of function such as compiler, business transaction processing, word processing, control systems. Business transactions should emphasize security and availability. Control systems should focus on availability, safety and security.
- *Security* – categories defined by the level of unauthorized access protection, audit trail, program and data protection. Emphasis should be on robustness and availability.
- *Reliability* – categories defined by the level of required reliability such as maturity, fault tolerance, and recoverability. Emphasis should be on reliability growth and configuration control for reliability achievement.
- *Performance* – categories defined by the software performance in terms of capacity, throughput, turnaround or response time. Emphasis should focus on response time that varies with load and capacity.
- *Language* – categories defined by the type of programming language primarily used for the software such as traditional (e.g. COBOL, FORTRAN), procedural (e.g. C), functional (e.g. Lisp), object oriented (e.g. C++). Emphasis should focus on programmer training and user familiarity with the programming language features and limitations.

A.1.3 Environment

- *Application area* – categories defined by the type or class of external system in which the software is used such as e-business, process control and networking system. Security and data integrity are major influencing factors to e-business. Safety and reliability are

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

important concerns in process control. Response time and availability are critical for network systems operation.

- *Computer system* – categories defined by the specific target computer system in which the software operates such as microprocessor controlled, mainframe, and real-time operating system. Limitations of memory size and programming code are important to microprocessor systems. Compatibility of software operability in mainframe hardware configuration and response time for real-time operation should be considered.
- *User class* – categories defined by the skill level or characteristics of its intended user class such as novice, intermediate or expert. User class identification is essential for interface design and user instruction development to facilitate ease of application.
- *Computer resource* – categories defined by the limitations of the computer resource such as memory requirement, disk requirement, and local area network requirement. The limitations of computer resource would affect development of software support needs as well as application capability.
- *Software criticality* – categories defined by the product integrity level requirements such as national security, organizational security, and privacy. Regulatory requirements and societal needs should be taken into consideration.
- *Software product availability* – categories defined by the availability of the software product such as commercial off-the-shelf (COTS), custom or proprietary software. The timing for acquisition and availability of software product is a decision factor for in-house design or outsourcing in project management.

A.1.4 Data

- *Data representation* – categories defined by data item, type and structure such as relational, indexed, formatted file. Data compatibility should be considered.
- *Software data usage* – categories defined by the type of usage of the intended software data such as single user, multiple sequential users. Data usage would affect data file design and data maintenance support criteria.

A.2 Software applications

Software is used in a wide variety of applications. In general, computer software applications can be grouped as follows.

- *System software* provides the infrastructure to control the computer hardware so that application software can perform. Examples are operating systems such as Microsoft Windows, Mac OS and Linux systems.
- *Application software* is the computer software designed to facilitate user in performing a particular task such as word processors, spread sheets and database applications.
- *Firmware* is the software resident in the programmable memory devices of the end-user products for internal control of various electronic devices such as remote controls, calculators, mobile phones, and digital cameras.
- *Middleware* is the computer software that connects software elements for multiple applications or provision of services such as multiprocessing in distributed systems and web services.
- *Testware* is a subset of software with the special purpose for software testing and test automation.
- *Programming software* is software development tool to facilitate software designers to create, debug, maintain, or support other programs and applications. Examples include CASE tools.
- *Malware* is the malicious software designed to infiltrate a computer without the owner's consent.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annex B
(informative)

Software system requirements and related dependability activities

B.1 General

Typical software system requirements and related dependability activities are summarized for each software life cycle stage. The information can be used as baseline reference for tailoring of software dependability projects.

B.2 Requirements definition

Software system requirements	*Related dependability activities*
• Market information on software products • System application requirements and user needs • Operating system domain and platform	• Identify software requirements • Identify performance needs • Identify support needs

B.3 Requirements analysis

Software system requirements	*Related dependability activities*
• Functional and capability performance requirements • Application scenarios • Application specific requirements for safety, security and integrity where applicable • Interface requirements • Qualification requirements • Feasibility of software design and testability • Feasibility of operation and maintenance • Installation and acceptance requirements • Documentation requirements	• Develop operational profile • Develop dependability project plan • Develop dependability assurance plan • Identify software dependability metrics • Determine data integrity requirements • Determine safety and security requirements • Establish human-factors engineering (ergonomics) design rules • Establish software support criteria • Identify constraints affecting dependability design and implementation including application specific requirements for design incorporation • Establish software reuse criteria • Establish software reliability growth and qualification acceptance criteria • Determine reliability test records and documentation requirements

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

B.4 Architectural design

Software system requirements	*Related dependability activities*
• An architecture describing the top-level structure and identifying the constituent software elements • Requirements transformation and allocation to facilitate configuration of the software items • Incorporation of application specific requirements for safety, security and integrity where needed in the system architecture • Internal and external interfaces for system integration and verification • Preliminary documentation for database and test requirements • Recommended design methods and standards to meet project objectives and design specifications • Traceability to the requirements of the software item • Feasibility of detailed design • Operation and maintenance conditions	• Perform application scenario analysis • Determine software structural and functional complexity • Incorporate application specific requirements in modelling system dependability performance • Perform availability/reliability functional model analysis • Perform software availability/reliability allocation • Establish dependability metrics database • Conduct preliminary availability/reliability prediction • Establish software reliability growth and qualification acceptance plan • Establish data records and reporting system • Establish software support plan • Review architectural design for implementation

B.5 Detailed design

Software system requirements	*Related dependability activities*
• A refined lower-level structure for coding of software units for inclusion in software configuration items • Detailed design specifications of software units and descriptions of software configuration items • Consistency and traceability of detailed design and architectural design specifications • Establishment of design methods and standards to meet project requirements • Establishment of special design methods to address safety, security and integrity issues where applicable • All interface requirements for compilation and testing of software units and configuration items • Documentation of database and detailed test requirements and test schedules • Project management reviews to monitor progress and delivery targets • Baseline for software configuration, and communications of design changes	• Implement software design rules • Establish measurement standards and metric evaluation criteria • Incorporate specific designs to meet safety, security and integrity requirements • Foster fault tolerant design • Apply software dependability standards • Conduct software code review and inspection • Refine software availability/reliability allocation • Perform software complexity assessment • Predict software unit reliability • Predict software subsystem availability/reliability • Perform design trade-off analysis • Refine software availability/reliability prediction • Update dependability metrics database • Implement configuration management • Conduct formal design review • Conduct project review

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

B.6 Realization

Software system requirements	*Related dependability activities*
• Software unit design and coding methods and standards • Software configuration item with specific software units • Verification criteria for unit test • Test coverage of software units • Verification of software functions including application specifications for safety, security and integrity requirements • Feasibility of software integration and testing	• Implement measurement standards and metric evaluation criteria • Determine code coverage of software units • Perform unit testing • Determine fault coverage and test completeness • Categorize fault data for classification • Implement dependability assurance process for unit and functional testing • Verify software units and functions in meeting performance and application specifications • Establish failure reporting, analysis and corrective action system • Implement software assurance program including outsourcing and supply chain where needed • Conduct project review

B.7 Integration

Software system requirements	*Related dependability activities*
• Integration strategy for software units and configuration • Verification criteria for software configuration item test • Verification of software subsystems including application specifications for safety, security and integrity requirements • Documentation of integration test results • Documentation of design changes • Regression strategy for re-verification of changed items • Test data collection system	• Implement fault tracking procedure • Implement fault analysis procedure • Initiate reliability growth program • Implement failure reporting, analysis and corrective action system • Implement data collection system • Verify software subsystems for integration • Perform integration testing • Evaluate availability/reliability test data • Identify problem areas • Perform corrective actions • Control design change and version release • Conduct project review

B.8 Acceptance

Software system requirements	*Related dependability activities*
• Criteria for software system acceptance • Demonstration of test compliance • Validation of integration test results met requirements • Validation of software system for customer acceptance • Regression strategy for re-testing of integrated software changes • Documentation of qualification acceptance results	• Perform reliability growth testing and accelerated testing as required • Monitor reliability trend and improvement status • Perform qualification testing • Review test results for acceptance • Initiate customer acceptance • Validate software system in meeting customer requirements including dependability performance demonstration and safety, security and integrity performance features where applicable • Document software version release status

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

B.9 Operation and maintenance

Software system requirements	*Related dependability activities*
• Operation procedures and conditions • Maintenance support strategy • Logistic support • Field data collection • User training • Software assurance program to sustain dependability of system operation	• Monitor field performance trends • Update field performance and maintenance support records • Conduct customer satisfaction surveys • Review field data to identify areas for reliability improvement • Establish field performance operational profile • Maintain system dependability performance history and experience database • Collect appropriate dependability metrics for reliability forecasting • Implement software assurance best practices

B.10 Software update/enhancement

Software system requirements	*Related dependability activities*
• Software upgrades • Perfective maintenance strategy implementation • New service introduction and impact assessment • Effects of enhancement/improvement on software performance	• Monitor software upgrades • Conduct perfective maintenance • Implement design change and configuration control • Assess new service introduction impact • Manage new software version release

B.11 Retirement

Software system requirements	*Related dependability activities*
• Termination of specific service • User advisory of termination of old service and new service replacement	• Identify retired software and support service termination • Advise customer care service of any required dependability actions

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annex C
(informative)

Capability maturity model integration process

Capability maturity model integration (CMMI) is a process improvement maturity model for the development of products and services. It consists of best practices that address development and maintenance activities covering the product life cycle from conception through delivery and maintenance. The *CMMI for development* [9] models contain practices that cover project management, process management, systems engineering, hardware engineering, software engineering, and other supporting processes used in development and maintenance. The CMMI process correlates with the implementation of software system requirements and related dependability activities as shown in Annex B. CMMI is used for benchmarking and appraisal activities, as well as guiding an organization's improvement efforts. The CMMI process is designated by levels.

- Capability levels, which belong to a continuous representation, apply to an organization's process improvement achievement in individual process areas. These levels are means to guide incremental improvement process corresponding to a given process area. There are six capability levels, numbered from 0 through 5.
- Maturity levels, which belong to a staged representation, apply to an organization's process improvement achievement across multiple process areas. These levels are used for predicting the general outcomes of the next project undertaken. There are five maturity levels, numbered from 1 through 5.

Table C.1 aligns the six capability levels to the five maturity levels for comparison.

Table C.1 – Comparison of capability and maturity levels

Level	Continuous representation capability levels	Staged representation maturity levels
0	An *incomplete process* is a process that is either not performed or partially performed. One or more of the specific goals of the process area are not satisfied, and no generic goals exist for this level since there is no reason to institutionalize a partially performed process.	N/A
1	A *performed process* is a process that satisfies the specific goals of the process area. It supports and enables the work needed to produce work products. Although capability level 1 results in important improvements, those improvements can be lost over time if they are not institutionalized. The application of institutionalization helps to ensure that improvements are maintained.	At maturity level 1, processes are usually ad hoc and chaotic. The organization usually does not provide a stable environment to support the processes. Success in these organizations depends on the competence of the people in the organization and not on the use of proven processes. The outcomes of maturity level 1 organizations often produce products and services that work; however, they frequently exceed their budgets and do not meet their schedules. There is a tendency to over commit, abandonment of processes in a time of crisis, and an inability to repeat their successes.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Level	Continuous representation capability levels	Staged representation maturity levels
2	A *managed process* is a performed process that has the basic infrastructure in place to support the process. It is planned and executed in accordance with the policy; employs skilled people who have adequate resources to produce controlled outputs; involves relevant stakeholders; is monitored, controlled, and reviewed; and is evaluated for adherence to its process description. The process discipline reflected by capability level 2 helps to ensure that existing practices are retained during times of stress.	At maturity level 2, the projects of the organization have ensured that processes are planned and executed in accordance with the policy; the projects employ skilled people who have adequate resources to produce controlled outputs; involve relevant stakeholders; are monitored, controlled, and reviewed; and are evaluated for adherence to their process descriptions. The outcomes of maturity level 2 organizations ensure that existing practices are retained during times of stress; projects are performed and managed according to their documented plans; the status of the work products and the delivery of services are visible to management at defined points at major milestones and at the completion of major tasks. Commitments are established among relevant stakeholders and are revised as needed. Work products are appropriately controlled. The work products and services satisfy their specified process descriptions, standards, and procedure.
3	A *defined process* is a managed process that is tailored from the organization's set of standard processes according to the organization's tailoring guidelines, and contributes work products, measures, and other process improvement information to the organizational process assets. The relevant standards, process descriptions, and procedures are consistent and tailored to suit a particular project or organizational unit. A defined process clearly states the purpose, inputs, entry criteria, activities, roles, measures, verification steps, outputs, and exit criteria. The processes are managed proactively by understanding the interrelationships of the process activities and detailed measures of the process, its work products, and its services.	At maturity level 3, the processes of the organization are well characterized and understood, and are described in standards, procedures, tools, and methods. These standard processes are used to establish their consistency in implementation across the organization. Projects establish their defined processes by tailoring the organization's set of standard processes according to tailoring guidelines. The outcomes of maturity level 3 organizations demonstrate consistencies in performance.
4	A *quantitatively managed process* is a defined process that is controlled using statistical and other quantitative techniques. Quantitative objectives for quality and process performance are established and used as criteria in managing the process. Quality and process performance is understood in statistical terms and is managed throughout the life of the process.	At maturity level 4, the organization and projects establish quantitative objectives for quality and process performance and use them as criteria in managing processes. Quantitative objectives are based on the needs of the customer, end users, organization, and process implementers. Quality and process performance is understood in statistical terms and is managed throughout the life of the processes. For selected sub-processes, detailed measures of process performance are collected and statistically analysed. Quality and process performance measures are incorporated into the organization's measurement repository to support fact-based decision-making. Special causes of process variation are identified and, where appropriate, the sources of special causes are corrected to prevent future occurrences. The outcomes of maturity level 4 organizations demonstrate adequate control of performance of processes by using statistical and other quantitative techniques to ensure performance results are quantitatively predictable.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Level	Continuous representation capability levels	Staged representation maturity levels
5	An *optimizing process* is a quantitatively managed process that is improved based on an understanding of the common causes of variation inherent in the process. The focus of an optimizing process is on continually improving the range of process performance through both incremental and innovative improvements.	At maturity level 5, the organization continually improves its processes based on a quantitative understanding of the common causes of variation inherent in processes; focuses on continually improving process performance through incremental and innovative process and technological improvements. Quantitative process improvement objectives for the organization are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement. The effects of deployed process improvements are measured and evaluated against the quantitative process improvement objectives. Both the defined processes and the organization's set of standard processes are targets of measurable improvement activities. The outcomes of maturity level 5 organizations demonstrate continual process improvement to achieve the established quantitative process improvement objectives.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annex D
(informative)

Classification of software defect attributes

D.1 General

The *orthogonal defect classification* (ODC) is a method used for capturing and grouping of software fault information in terms of software defect attributes. The ODC process provides the capability to extract defect signatures and infer the health of the software development process. The classification is based on what is known about the defect. When a fault or defect is opened, the way in which the defect was found and exposed and the impact to the user are normally known. Therefore, the ODC attributes of *Activity*, *Trigger*, and *Impact* can be classified. Similarly, when a fault is diagnosed and fixed, the details of the fix are known. The ODC attributes of *Defect Target*, *Defect Type*, *Qualifier*, *Source*, and *Age* can be classified. Additional non-ODC attributes such as the scheduled project phase-found, severity, and component, that are captured in any fault or defect tracking system can be used in conjunction with ODC-based analysis. ODC does not impose a specific structure. The following clauses summarize the classification of software defect attributes under the headings of *opener section*, *closer section* and *activity to trigger mapping*.

D.2 Opener section

An opener section is to classify attributes when a fault is found.

When a fault is discovered and is open for diagnosis during software development, the information on fault exposure and likely impact and severity can be assessed. Typical attributes of fault information captured when a fault is found can be summarized in Table D.1.

Table D.1 – Classification of software defect attributes when a fault is found

Fault removal activity [a] **(when detected)**	**Trigger** [b] **(how detected)**	**Impact** [c] **(effect and severity)**
• Design review • Code inspection • Unit test • Function test • System test • Acceptance test • Qualification test • Reliability growth test	• Design conformance • Compatibility • Concurrency • Coverage • Sequencing • Interaction • Configuration	• Installability • Serviceability • Integrity/Security/safety • Reliability • Maintenance • Accessibility • Usability
NOTE 1 Software faults are often hard to replicate. It is important to capture the circumstances leading up to and surrounding the incidence of the fault detection. NOTE 2 Traceability to requirements is needed; either traceable to the software requirements or traceable to a specific test case.		
[a] *Activity*: actual activity that was being performed at the time the fault was discovered. [b] *Trigger:* the environment or condition that had to exist for the fault exposure. [c] *Impact*: for development faults, the impact is assessed as the potential effect and severity to the user; for field reported fault, the impact is the failure effect and severity to the user.		

D.3 Closer section

A closer section is to classify attributes when a fault is fixed.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

When a fault is closed after the fix is applied, the exact nature of the fault and the scope of the fix are known. Typical attributes of fault information captured when a fault is fixed can be summarized in Table D.2.

Table D.2 – Classification of software defect attributes when a fault is fixed

Target [a]	Type [b]	Qualifier [c]	Age [d]	Source [e]
• Design/Code	• Initiation • Checking • Function • Timing • Interface	• Missing • Incorrect • Extraneous	• Base • New • Rewritten	• Developed in-house • Outsourced • Reused from library • Ported

a *Target*: the high level identity of the entity that was fixed.

b *Type*: the nature of the actual correction that was made.

c *Qualifier (applies to the defect Type):* describes the element of either non-existent, or wrong, or irrelevant implementation.

d *Age*: identifies the history of the *Target* such as *Design/Code*, which had the defect.

e *Source*: identifies the origin of the *Target* such as *Design/Code,* which had the defect.

D.4 Activity to trigger mapping

The mappings of activity to trigger group the applicable triggers relevant to the software design review, inspection and test activities. Tables D.3, D.4, D.5, and D.6 show some generic examples of the activity to trigger mappings.

Table D.3 – Design review/code inspection activity to triggers mapping

Activity	Triggers
Design review/code inspection *Reviewing design or comparing the documented design against known requirements*	• *Design conformance* The document reviewer or the code inspector detects the fault while comparing the design element or code segment being inspected with its specification in the preceding stage. This would include design documents, code, development practices and standards, or to ensure design requirements are not missing or ambiguous. • *Logic/flow* The inspector uses knowledge of basic programming practices and standards to examine the flow of logic or data to ensure they are correct and complete. • *Backward compatibility* The inspector uses extensive product/component experience to identify an incompatibility between the function described by the design document or the code, and that of earlier versions of the same product or component. From a field perspective, the customer's application, which ran successfully on the prior release, fails on the current release. • *Lateral compatibility* The inspector with broad-based experience, detects an incompatibility between the function described by the design document or the code, and the other systems, products, services, components, or modules with which it must interface. • *Concurrency* The inspector is considering the serialization necessary for controlling a shared resource when the fault is discovered. This would include the serialization of multiple functions, threads, processes, or kernel contexts as well as obtaining and releasing locks. • *Internal document* There are incorrect information, inconsistency, or incompleteness within internal documentation. Prologues, code comments, and test plans represent some examples of documentation, which would fall under this category. • *Language dependency* The developer detects the defect while checking the language specific details of the implementation of a component or a function. Language standards, compilation concerns, and

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Activity	Triggers
	language specific efficiencies are examples of potential areas of concern. • *Side Effects* The inspector uses extensive experience or product knowledge to foresee some system, product, function, or component behaviour which can result from the design or code under review. The side effects would be characterized as a result of common usage or configurations, but outside of the scope of the component or function with which the design or code under review is associated. • *Rare situation* The inspector uses extensive experience or product knowledge to foresee some system behaviour, which is not considered or addressed by the documented design or code under review, and would typically be associated with unusual configurations or usage. Missing or incomplete error recovery would not, in general, be classified with a trigger of *rare situation*, but would most likely fall under *design conformance* if detected during *review/inspection*.

Table D.4 – Unit test activity to triggers mapping

Activity	Triggers
Unit test *White box testing or execution based on detailed knowledge of the code internals*	• *Simple path* The test case was motivated by the knowledge of specific branches in the code and not by the external knowledge of the functionality. This trigger would not typically be selected for field reported defects, unless the customer is very knowledgeable of the code and design internals, and is specifically invoking a specific path (as is sometimes the case when the customer is a business partner or vendor). • *Complex path* In white/grey box testing, the test case that found the defect was executing some contrived combinations of code paths. The tester attempted to invoke execution of several branches under several different conditions. This trigger would only be selected for field reported defects under the same circumstances as those described under *simple path.*

Table D.5 – Function test activity to triggers mapping

Activity	Triggers
Function test *Black box execution based on external specifications of functionality*	• *Coverage* During black box testing, the test case that found the defect was a straightforward attempt to exercise code for a single function, using no parameters or a single set of parameters. • *Variation* During black box testing, the test case that found the defect was a straightforward attempt to exercise code for a single function but using a variety of inputs and parameters. These might include invalid parameters, extreme values, boundary conditions, and combinations of parameters. • *Sequencing* During black box testing, the test case that found the defect executed multiple functions in a very specific sequence. This trigger is only chosen when each function executes successfully when run independently, but fails in this specific sequence. It is also possible to execute a different sequence successfully. • *Interaction* During black box testing, the test case that found the defect initiated an interaction among two or more bodies of code. This trigger is only chosen when each function executes successfully when run independently, but fails in this specific combination. The interaction involves more than a simple serial sequence of the executions.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table D.6 – System test activity to triggers mapping

Activity	Triggers
System test *Testing or execution of the complete system, in the real environment, requiring all resources*	• *Workload/stress* The system is operating at or near some resource limit, either upper or lower. These resource limits can be created by means of a variety of mechanisms, including running small or large loads, running a few or many products at a time, letting the system run for an extended period of time. • *Recovery/exception* The system is being tested with the intent of invoking an exception handler or some type of recovery code. The defect would not have surfaced if some earlier exception had not caused exception or recovery processing to be invoked. From a field perspective, this trigger would be selected if the defect were in the system or product ability to recover from a failure, not the failure itself. • *Start-up/restart* The system or subsystem was being initialized or restarted following some earlier shutdown or complete system or subsystem failure. • *Hardware configuration* The system is being tested to ensure functions execute correctly under specific hardware configurations. • *Software configuration* The system is being tested to ensure functions execute correctly under specific software configurations. • *Blocked test (normal mode)* The product is operating well within resource limits and the defect surfaced while attempting to execute a system test scenario. This trigger would be used when the scenarios could not be run because there are basic problems, which prevent their execution. This trigger must not be used in customer reported defects.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annex E
(informative)

Examples of software data metrics obtained from data collection

E.1 Fault data metrics

Metric	Application
Problem reporting data • Date and time fault detected • Detected fault description • Fault detected in program area • Person detected the fault • Fault symptom and status • Severity and priority	Data collected on software projects should be used for reporting problem on fault identification and occurrence
Corrective action data • Date fault corrected • Person corrected the fault • Maintenance action taken • Description of modification • Identification of modules modified • Version control information • Time required to correct fault • Date verified as fault corrected • Person verified the correction	Data collected on corrective actions and verified as fault corrected should be used for reporting problem resolution.
Cumulated faults detected	Cumulated faults detected data should be used to determine fault rate and reliability trend over a period of time
Cumulated faults corrected	Cumulated faults corrected data should be used to determine known faults that require corrective actions and tracking the effectiveness of maintenance actions
Faults detection rate	Faults detection rate is used to indicate trend to facilitate planning of maintenance strategy and resource management
Faults correction rate	Faults correction rate is used to indicate trend to facilitate planning of maintenance strategy and resource management. Priority setting for maintenance action is based on the severity of the fault problem
Faults per location	Fault tracking according to software functions to identify specific area of the code is more error-prone
Criticality of faults	Classifying the degree of impact of faults to set priority for maintenance actions
Number and percentage of severe faults	Indications for planning maintenance strategy
Structural complexity per location	Use with other metrics to determine impact of faults generated related to the complexity of the software structure and location
Functional complexity per location	Use with other metrics to determine impact of faults generated related to the complexity of the software functions and location

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

E.2 Product data metrics

Metric	Application
Number and percentage of modules that perform more than one function	Indication of cohesiveness of the overall software design on functional complexity. High complexity module will result in low cohesiveness that would require redesign
Number and percentage of modules that have a high structural complexity	Indication of the overall software design requiring redesign to reduce complexity
Number and percentage of modules that have exactly one entrance and one exit	Indication of a cohesive design that should be used as basis for structured design practice
Number and percentage of modules that are documented according to standards	Indication of code completeness that should be used to determine if the code contains all the requirements and addresses the requirements completely
Number and percentage of faults that are found in reused code	Indication of the unreliability of the reused code

E.3 Process data metrics

Metric	Application
Faults introduced by life cycle stage	Indication of when and at what stage the faults were introduced, and to take appropriate actions.
Faults detected by life cycle stage	Indication of when and at what stage the faults were detected, and justification for delay corrective actions for fault removal.
Total time spent in analysis	Indication of the time spent on analysis for problem identification and isolation for corrective action, and the associated resources required.
Total time spent in design	Indication of the time spent for software design, and the associated resources required.
Total time spent in coding	Indication of the time spent for coding and programming, and the associated resources required.
Total time spent in unit testing	Indication of the time spent on unit testing, and the associated resources required.
Total time spent in system testing	Indication of the time spent on system testing, and the associated resources required.
Total maintenance time	Indication of the time spent on maintenance activities, and the associated resources required.
Average maintenance administration time	Indication of the time spent on maintenance administration, and the associated resources required. Maintenance administrative duties include before and after the fault is corrected, such as time spent in assigning maintenance personnel, release of correction in a new version.
Average corrective action time	Indication of the time spent on corrective actions, and the associated resources required. This reflects the cost-effectiveness in the maintenance activities.
Reason for corrective action	This is used to determine the source of faults. Typical reasons include: • previous maintenance action • new requirement • requirement change • misinterpreted requirement • missing requirement • ambiguous requirement • change in software environment • change in hardware environment • code/logic error • performance error
Cost of corrective action	Indication of the total cost of corrective action including fault isolation, problem resolution, and administration for effective maintenance action.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Metric	Application
Percentage of functions tested and verified	Indication of test coverage, test efficiency and completeness.
Percentage of independent paths tested and verified	Indication of test coverage of structural testing and completeness.
Percentage of source lines of code test and verified	Indication of test coverage of software code and completeness.
Historical data	Provision of data history on problem areas related to design, process and product issues.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annex F
(informative)

Example of combined hardware/software reliability functions

A monitoring control system is shown as a combined hardware/software system consisting of the electronic operation and application functions represented by the system hierarchy. The following provides the basic system hardware and software functional descriptions:

- hardware actuation function to operate the monitoring control mechanism;
- actuation electronic circuit components;
- hardware sensing function to start or stop the activation;
- sensing electronic circuit components;
- software application functions for controlling the rapid initiation and release of the hardware sensing function including
  - software unit for controlling initiation of activation;
  - software unit for controlling release and deactivation;
  - software unit for monitoring the speed of activation and deactivation.

These functions are represented by block diagram as shown in Figure F.1.

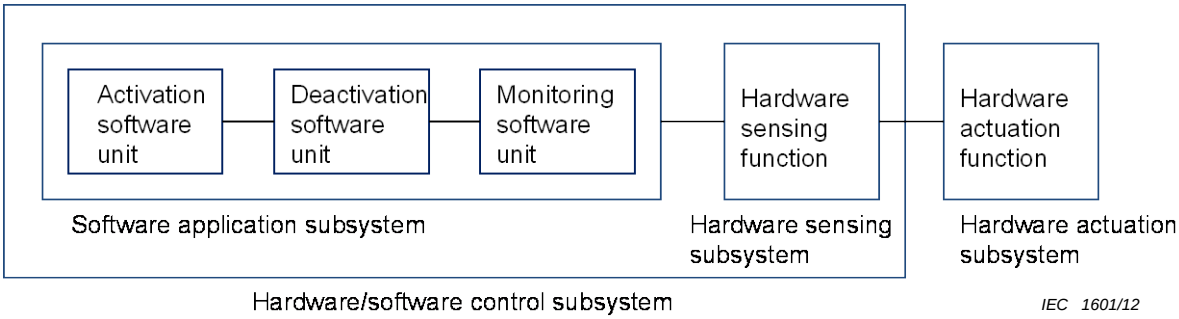


Figure F.1 – Block diagram for a monitoring control system

The *hardware actuation function* can be considered as a hardware subsystem with the complete actuation electronic circuitry comprising of electronic components.

The *hardware sensing function* can be considered as a hardware subsystem with the complete sensing electronic circuitry comprising of electronic components.

The *software application functions* consist of three separate software units designed to work together with the host hardware sensing subsystem. Each of the software unit is designed to perform exactly one function as designated. In order for the hardware sensing subsystem to work, it is necessary that all the software units are incorporated into the software application subsystem to perform as a combined hardware/software subsystem.

For reliability analysis the following should be noted in determining system and subsystem failure rates. Assumption is made of constant failure rates of electronic components, which comprise all hardware functions.

- The hardware actuation function failure rate can be determined by the sum of the failure rates of individual components of the actuation circuitry assuming no redundancy in design and operating continuously full time (λ_1).

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- The hardware sensing function failure rate can be determined by the sum of the failure rates of individual components of the sensing circuitry assuming no redundancy in design and operating continuously full time (λ_2).
- The reliability of the software application functions is determined by the combined reliability of the three software units due to their dependency associated with the monitoring control system operational profile. The operational profile does not require full calendar time for execution of these software units, but only needed for their part-time applications upon demand. The execution time associated with each of the software units should be incorporated in the calculation of fault density relevant to each respective software unit. The fault density is then converted to calendar time to determine the effective failure rate. Fault density is used for repairable items and failure rate is used for non-repairable items. Since software is non-repairable, failure rate applies here. In this respect, the entire software application functions are treated as one single software configuration item with failure rate (λ_3).
- The reliability of the monitoring control system in terms of failure rate is determined by the sum of $\lambda_1 + \lambda_2 + \lambda_3 = \lambda$.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annex G
(informative)

Summary of software reliability model metrics

There are many software reliability models existing today and used in industry practice. Most models have been developed for specific applications. Most of them have been computerized and automated to facilitate data processing, analysis and evaluation. There is no one single model that would fit all applications. It is up to the user to select and apply appropriate software reliability models to meet their own project needs. The following presents a list of common metrics used in most software reliability models.

a) Total number of inherent faults in the software. This metric is assumed to be fixed and finite.

b) Total number of latent faults (bugs) in the software. This metric is assumed to be variable because of the possibility of inserting new faults into the code over time.

c) Total number of faults corrected at some point in time, or after some usage or testing time has elapsed.

d) Total number of faults detected at some point in time, or after some usage or testing time has elapsed.

e) Number of testing periods or intervals. This is the number of intervals between fault correction activities. Some models assume that faults are corrected as soon as they are detected.

f) Total number of faults corrected up to a certain testing period.

g) Change in the failure rate.

h) Testing or usage time accumulated up to the present time or present number of detected faults.

i) Execution time accumulated.

j) Initial failure rate.

k) Present failure rate.

l) Growth rate.

m) Estimated number of lines of executable code at testing or usage time.

n) Total number of test cases run.

o) Total number of successful test cases run.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annex H
(informative)

Software reliability models selection and application

There are many models and metrics available today for estimating software reliability and measuring characteristics of software. All reliability models are developed for curve fitting exercises using the metric data collected for the model inputs. The validity and accuracy of the model application and resultant output depend on the assumptions made in the model formulation and the relevancy of the data input to the model to generate the output. Most models are developed to meet a specific need during the software life cycle. Examples include prediction model during software design, and estimation model to determine additional test time required before software release. Some models are developed to predict the reliability of software before the code is written. Data input for reliability prediction in such case is often based on historic data of similar software system and application. Other models are deployed for estimation of software reliability growth trends based on interim test data input. There is no one single model that is capable of covering the entire spectrum of the software life cycle. In practice, several models are often tried and used to determine software reliability. Statistical techniques, such as goodness of fit test to check how well a model fits a set of observations, are often used for model selection. Most software reliability models are automated due to their iterative computational needs. Interpretations of reliability modelling results require practical experience and reliability modelling expertise.

Table H.1 presents some examples of software reliability models used in industry practice. It is not the intention of this standard to provide detailed model formulation and parametric applications. References on software reliability models and their specific applications are well documented in the literature [14, 37].

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Table H.1 – Examples of software reliability models

	Model name	Assumptions	Data requirements	Application limitations
1	Musa basic	• Finite number of inherent errors (latent faults) • Constant error rate over time •Exponential distribution	• Number of detected faults at some point in time • Estimate of initial failure rate • Software system present failure rate	• Software is operational • Use after system integration • Assume no new faults are introduced in correction • Assume number of residual faults decreases linearly over time
2	Musa-Okumoto	• Infinite number of inherent errors (latent faults) • Changing error rate over time •Logarithmic distribution	• Number of detected faults at some point in time • Estimate of initial failure rate • Relative change of failure rate over time • Software system present failure rate	• Software is operational • Use for unit to system tests • Assume no new faults are introduced in correction • Assume number of residual faults decreases exponentially over time
3	Jelinski-Moranda	• Finite and constant number of inherent errors (latent faults) • Constant error rate over time • Errors corrected as soon as detected • Binomial exponential distribution	• Number of corrected faults at some point in time • Estimate of initial failure rate • Software system present failure rate	• Software is operational • Use after system integration • Assume no new faults are introduced in correction • Assume number of residual faults decreases linearly over time
4	Littlewood-Verrall	• Uncertainty in correction process	• Estimate of the number of failures • Estimate of the reliability growth rate • Time between failures detected or the time of the failure occurrence	• Software is operational
5	Schneidewind	• No new faults are introduced in correction	• Estimate of failure rate at start of first interval • Estimate of proportionality constant of failure rate over time • Faults detected in equal time interval	• Software is operational • Rate of fault detection decreases exponentially over time
6	Geometric	• Inherent number of faults to be infinite	• Decreasing geometric progression function as failures are detected • Time between failure occurrences or time of failure occurrence	• Software is operational • Faults are independent and unequal in probability of occurrence and severity
7	Brooks-Motley	•Rate of fault detection constant over time	• Test effort of each test • Probability of fault detection in i^{th} test • Probability of correcting faults without introducing new ones • Number of faults remaining at start of i^{th} test • Total number of faults found in each test	• Software developed incrementally • Some software modules have different test effort than others

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

	Model name	Assumptions	Data requirements	Application limitations
8	Bayesian	• Software is relatively fault free	• Length of testing time for each interval • Number of failures detected in each interval	• Software is operational • Software is corrected at end of testing interval
9	Keene	• Correlates the delivered latent fault content with the development process capability and software size (KSLOCs)	• CMM maturity level to assess process capability, estimated KSLOCs of deliverable code, estimated number of months to reach maturity after release, fault latency, per cent of severity 1 and 2 faults, recovery time, use hours per week of the code, and per cent fault activation	• Per cent fault activation is an estimated parameter that represents the average percentage of seats of system users that are likely to experience a particular fault; the CMMI capability level should be assessed for the development organization as well as the maintenance organization

The following criteria should be used to facilitate model selection:

- failure profiles;
- maturity of software product;
- characteristics of software development;
- characteristics of software test;
- existing metrics and data.

For model execution, the use of automated computational tools is recommended. There are commercially available tools that cover some or all of the software reliability models identified in Table H.1. The main advantage of using automated computational tools is the time and cost savings of implementation for model applications. By choosing an appropriate tool it is possible to compare the results of a set of data runs on several models to determine best fit.

The following criteria should be considered in selecting a tool or tools for an organization:

- availability of the tool compatible with the organization's computer systems;
- cost of installing and maintaining the program;
- number of studies likely to be carried out for tool applications;
- types of software systems to be studied;
- quality of the tool documentation;
- ease of learning the tool;
- flexibility and power of the tool;
- technical support of the tool.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Bibliography

[1] IEC 62508, *Guidance on human aspects of dependability*

[2] ISO/IEC 12207, *Systems and software engineering – Software life cycle processes*

[3] IEC 60300-1, *Dependability management – Part 1: Dependability management systems*

[4] IEC 60300-2, *Dependability management – Part 2: Guidelines for dependability management*

[5] KLINE, M.B., *Software and hardware R&M: What are the differences?* Proceedings of the Annual Reliability and Maintainability Symposium, 1980

[6] LIPOW, M., SHOOMAN, M. L., *Software reliability, Tutorial Session*, Annual Reliability and Maintainability Symposium, 1986

[7] ISO/IEC 15288, *Systems and software engineering – System life cycle processes*

[8] *Capability Maturity Model® (CMM®)*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA USA

[9] *Capability Maturity Model Integration® (CMMI®) for Development, Version 1.2*; Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA USA 2006

[10] IEC 60300-3-3, *Dependability management – Part 3-3: Application guide – Life cycle costing*

[11] IEC 62347, *Guidance on system dependability specifications*

[12] IEC 61160, *Design review*

[13] CHILLAREGE, Ram, *Orthogonal Defect Classification – A concept for in process Measurements*, IEEE Transactions on Software Engineering, 1992

[14] LYU, M.R. (Ed.): *The Handbook of Software Reliability Engineering*, IEEE Computer Society Press and McGraw-Hill Book Company, 1996

[15] IEEE-1633: *Recommended Practice on Software Reliability, 2009*

[16] ISO/IEC 20926, *Software and systems engineering — Software measurement — IFPUG functional size measurement method 2009*

[17] IEC 61078, *Analysis techniques for dependability – Reliability block diagram and boolean methods*

[18] IEC 61025, *Fault tree analysis (FTA)*

[19] IEC 61165, *Application of Markov techniques*

[20] IEC 62551, *Analysis techniques for dependability – Petri net techniques*[2]

[21] DUGAN, J.B., *Fault Tree Analysis for Computer-based Systems, Tutorial Session*, Annual Reliability and Maintainability Symposium, 2000

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

[22] IEC 62198, *Project risk management – Application guidelines*

[23] IEC 60812, *Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*

[24] IEC 60300-3-1, *Dependability management – Part 3-1: Application guide – Analysis techniques for dependability – Guide on methodology*

[25] ISO/IEC 15026-3, *Systems and software engineering – System and software assurance – Part 3: System integrity levels*

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

[27] ISO/IEC 13335-1, *Information technology – Security techniques – Management of information and communications technology security – Part 1: Concepts and models for information and communications technology security management*
(withdrawn)

[28] IEC 62429, *Reliability growth – Stress testing for early failures in unique complex systems*

[29] IEC 61014, *Programmes for reliability growth*

[30] IEC 61164, *Reliability growth – Statistical test and estimation methods*

[31] IEC 62506, *Methods for product accelerated testing*[2]

[32] ISO/IEC/IEEE 42010, *Systems and software engineering – Architectural description*

[33] ISO/IEC 18019, *Systems and software engineering – Guidelines for the design and preparation of user documentation for application software*
(withdrawn)

[34] *Software Assurance Standard*, NASA-STD-8739.8 w/Change 1, May 2005

[35] ISO/IEC 15026-4, *Systems and software engineering – System and software assurance – Part 4: Assurance in the life cycle*[2]

[36] ISO/IEC 15026-2, *Systems and software engineering – System and software assurance – Part 2: Assurance case*

[37] LAKEY, P.B., NEUFELDER, A.M., *System and Software Reliability Assurance Notebook*, Rome Laboratory, 1996

[38] *National Information Assurance (IA) Glossary*, (CNSS Instruction No. 4009), National Security Telecommunications and Information Systems Security Committee (NSTISSC), published by the United States federal government (unclassified), June 2006

2 To be published.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

[39] *Software Assurance: An Overview of Current Industry Best Practices*, Software Assurance Forum for Excellence in Code, February 2008

[40] ISO/IEC TR 12182, *Information technology – Categorization of software*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

SOMMAIRE

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

LIGNES DIRECTRICES CONCERNANT LA SÛRETÉ DE FONCTIONNEMENT DU LOGICIEL

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de brevet. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale CEI 62628 a été établie par le comité d´études 56 de la CEI: Sûreté de fonctionnement.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
56/1469/FDIS	56/1480/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite;
- supprimée;
- remplacée par une édition révisée, ou
- amendée.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

INTRODUCTION

Les logiciels sont très répandus dans les produits et systèmes actuels, à titre d'exemples les applications logicielles dans les équipements de commande programmables, les systèmes informatiques et les réseaux de communication. Au cours des années, de nombreuses normes ont été développées en matière de génie logiciel, de gestion des processus logiciels, de qualité logicielle et de garantie de fiabilité, mais seules quelques normes traitent les questions logicielles d'un point de vue de la sûreté de fonctionnement.

La sûreté de fonctionnement est la capacité d'un système à fonctionner au moment voulu et tel que prévu de façon à satisfaire aux objectifs spécifiés dans des conditions d'utilisation données. La sûreté de fonctionnement d'un système suppose que le système est sûr et capable d'exécuter le service souhaité, sur demande, pour répondre aux besoins des utilisateurs. L'évolution croissante des applications logicielles dans l'industrie des services a entraîné un rapide développement des services Internet et du Web. Les interfaces et protocoles standardisés ont permis l'utilisation de logiciels-tiers sur Internet et par là-même, les applications inter-plates-formes, inter-fournisseurs et inter-domaines. Les logiciels sont devenus un moteur pour réaliser des opérations complexes et permettre la réalisation d'affaires électroniques viables pour une intégration sans fil et la gestion des processus d'affaires. La conception de logiciels a mis l'accent sur le traitement des données, la surveillance de la sécurité et les liens de communication dans les services de réseau. Ce changement de paradigme a mis le monde des affaires mondial dans une situation reposant fortement sur les systèmes logiciels pour soutenir les opérations financières. La sûreté de fonctionnement du logiciel joue un rôle dominant pour influencer le succès des performances d'un système et l'intégrité des données.

La présente Norme internationale présente les pratiques d'excellence actuelles et la méthodologie correspondante pour faciliter la réalisation de la sûreté de fonctionnement du logiciel. Elle identifie l'influence du management sur les aspects logiciels de la sûreté de fonctionnement et fournit les processus techniques correspondants à la sûreté de fonctionnement du logiciel dans les systèmes. L'évolution de la technologie logicielle et l'adaptation rapide des applications logicielles dans les pratiques industrielles ont créé le besoin d'une norme pratique relative à la sûreté de fonctionnement du logiciel pour le marché mondial des affaires. Une approche structurée est fournie afin de servir de lignes directrices pour l'utilisation de cette norme.

Les exigences et processus génériques en matière de sûreté de fonctionnement du logiciel sont présentés dans cette norme. Ils représentent la base des applications de sûreté de fonctionnement pour le développement de produits logiciels et la mise en œuvre de systèmes logiciels en grande partie. Des exigences supplémentaires sont requises pour les applications d'importance vitale, de sûreté et de sécurité. Les aspects liés à la qualification des logiciels spécifiques à l'industrie en termes de fiabilité et de qualité ne sont pas traités dans cette norme.

Cette norme peut également servir de lignes directrices pour la conception de la sûreté de fonctionnement de micro-logiciels. Elle ne porte cependant pas sur les aspects de mise en œuvre des micro-logiciels avec les logiciels contenus ou intégrés dans les puces matérielles pour réaliser leurs fonctions dédiées telles les puces à circuit intégré à application spécifique (ASIC) et les contrôleurs commandés par microprocesseur. Ces produits sont souvent conçus et intégrés dans le cadre des fonctionnalités matérielles physiques pour minimiser leur taille et leur poids et faciliter les applications en temps réel comme celles utilisées dans les téléphones cellulaires. Bien que les principes et pratiques de sûreté de fonctionnement générale, décrits dans cette norme, puissent faciliter la conception et l'application de micro-logiciels, des exigences spécifiques sont nécessaires pour leur construction physique, la fabrication des appareils et la mise en œuvre des produits logiciels intégrés. La physique de défaillance des appareils spécifiques à l'application se comporte différemment par rapport aux défaillances du système logiciel.

La présente Norme Internationale n'est pas conçue à des fins d'évaluation de la conformité ou de certification.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

LIGNES DIRECTRICES CONCERNANT LA SÛRETÉ DE FONCTIONNEMENT DU LOGICIEL

1 Domaine d'application

La présente Norme internationale porte sur les problèmes concernant la sûreté de fonctionnement du logiciel et définit les lignes directrices pour la réalisation de la sûreté de fonctionnement dans les performances logicielles influencées par les disciplines de management, les processus de conception et les environnements d'application. Elle définit un cadre générique pour les exigences en matière de sûreté de fonctionnement du logiciel, fournit un processus de sûreté de fonctionnement du logiciel pour les applications du cycle de vie du système, présente les critères d'assurance et la méthodologie pour la conception et la mise en œuvre de la sûreté de fonctionnement du logiciel, et fournit des approches et mesures pratiques des caractéristiques de sûreté de fonctionnement dans les systèmes logiciels.

La présente norme s'applique aux développeurs et fournisseurs de systèmes logiciels, aux intégrateurs de systèmes, aux opérateurs et aux spécialistes de la maintenance, ainsi qu'aux utilisateurs de systèmes logiciels qui sont concernés par les approches pratiques et l'ingénierie d'application pour atteindre la sûreté de fonctionnement des produits et systèmes logiciels.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 60050-191, *Vocabulaire Electrotechnique International – Chapitre 191: Sûreté de fonctionnement et qualité de service*

CEI 60300-3-15, *Gestion de la sûreté de fonctionnement – Partie 3-15: Guide d'application – Ingénierie de la sûreté de fonctionnement des systèmes*

3 Termes, définitions et abréviations

Pour les besoins du présent document, les termes et définitions donnés dans la CEI 60050-191, ainsi que les suivants, s'appliquent.

3.1 Termes et définitions

3.1.1
logiciel
programmes, procédures, règles, documentation et données d'un système de traitement de l'information

Note 1 à l'article: Un logiciel est une création intellectuelle qui est indépendante du moyen sur lequel elle est enregistrée.

Note 2 à l'article: Un logiciel nécessite des équipements pour exécuter les programmes et pour stocker et transmettre les données.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Note 3 à l'article: Les types de logiciels incluent les micro-logiciels, les logiciels de base et les logiciels d'application.

Note 4 à l'article: La documentation inclut: les exigences, les spécifications, les spécifications de conception, les listes de codes sources, les commentaires dans le code source, le texte «d'aide» et les messages à afficher sur l'interface homme/machine, les instructions d'installation, les instructions de fonctionnement, les modes d'emploi et les guides d'assistance utilisés dans la maintenance des logiciels.

3.1.2
micro-logiciel
logiciel contenu dans un support de mémoire en lecture seule et non destiné à la modification

EXEMPLE Système d'entrée/sortie de base (BIOS) d'un ordinateur personnel.

Note 1 à l'article: Une modification logicielle nécessite le remplacement ou la reprogrammation de l'équipement le contenant.

3.1.3
logiciel intégré
logiciel dans un système dont le principal objectif n'est pas informatique

EXEMPLES Logiciel utilisé dans le système de gestion d'un moteur ou les systèmes de commande des freins des véhicules à moteur.

3.1.4
unité logicielle
module logiciel
élément logiciel qui peut être compilé séparément en codes de programmation pour effectuer une tâche ou activité afin d'obtenir le résultat souhaité d'une fonction logicielle ou de fonctions logicielles

Note 1 à l'article: Les termes «module» et «unité» sont souvent utilisés de manière interchangeable ou définis comme sous-éléments d'un autre, de différentes manières selon le contexte. La relation entre ces termes n'est pas encore standardisée.

Note 2 à l'article: Dans l'idéal, une unité logicielle peut être conçue et programmée pour exécuter exactement une fonction spécifique. Dans certaines applications, il peut être nécessaire de combiner deux unités logicielles ou plus pour obtenir la fonction logicielle spécifiée. Dans ces cas, ces unités logicielles sont soumises à essai sous forme de fonction logicielle unique.

3.1.5
élément de configuration logiciel
élément logiciel qui a été configuré et traité comme élément unique dans le processus de gestion de la configuration

Note 1 à l'article: Un élément de configuration logiciel peut être composé d'une ou plusieurs unités logicielles pour exécuter une fonction logicielle.

3.1.6
fonction logicielle
opération élémentaire exécutée par le module logiciel ou l'unité logicielle comme spécifié ou défini selon les exigences indiquées

3.1.7
système logiciel
ensemble défini d'éléments logiciels qui, lorsqu'ils sont intégrés, agissent collectivement pour répondre à une exigence

EXEMPLES Logiciels d'application (logiciels pour la comptabilité et la gestion des informations); logiciels de programmation (logiciels pour l'analyse des performances et les outils CASE) et logiciels de base (logiciels pour le contrôle et la gestion du système informatique, par exemple systèmes d'exploitation).

3.1.8
sûreté de fonctionnement du logiciel
capacité de l'élément logiciel à fonctionner au moment voulu et tel que requis lorsqu'il est intégré dans l'exploitation du système

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

3.1.9
défaut logiciel
bogue
état d'un élément logiciel qui peut empêcher sa bonne exécution

Note 1 à l'article: Les défauts logiciels sont soit des défauts de spécification, des défauts de conception, des défauts de programmation, des défauts insérés dans le compilateur ou des défauts introduits pendant la maintenance du logiciel.

Note 2 à l'article: Un défaut logiciel est inactif jusqu'à ce qu'il soit activé par un déclencheur spécifique, et redevient généralement inactif lorsque le déclencheur est retiré.

Note 3 à l'article: Dans le cadre de cette norme, un bogue est un cas spécial de défaut logiciel, également connu sous le nom de défaut logiciel latent.

3.1.10
défaillance logicielle
défaillance qui est une manifestation d'un défaut logiciel

Note 1 à l'article: Un seul défaut logiciel continue à se manifester comme une défaillance jusqu'à ce qu'il soit éliminé.

3.1.11
code
caractère ou profil binaire auquel est assignée une signification particulière pour exprimer un programme informatique dans un langage de programmation

Note 1 à l'article: Les codes sources sont des instructions codées et des définitions de données exprimées dans un format approprié pour l'entrée dans un assembleur, compilateur ou autre traducteur.

Note 2 à l'article: Le codage est le processus de transformation de la logique et de données à partir des spécifications de conception ou descriptions dans un langage de programmation.

Note 3 à l'article: Un langage de programmation est un langage utilisé pour exprimer des programmes informatiques.

3.1.12
programme (informatique)
ensemble d'instructions codées exécutées pour effectuer des opérations logiques et mathématiques spécifiées sur les données

Note 1 à l'article: La programmation est l'activité générale de développement de logiciels dans laquelle le programmeur ou l'informaticien définit un ensemble spécifique d'instructions que l'ordinateur doit exécuter.

Note 2 à l'article: Un programme consiste en une combinaison d'instructions codées et de définitions de données qui permettent au matériel informatique d'effectuer des fonctions de calcul et de commande.

3.2 Abréviations

Abréviation	Terme en français	Terme en anglais
ASIC	Circuit intégré à application spécifique	Application specific integrated circuit
CASE	Génie logiciel assisté par ordinateur	Computer-aided software engineering
CMM	Modèle d'évolution des capacités	Capability maturity model
CMMI	Intégration du modèle d'évolution des capacités	Capability maturity model integration
COTS	Produits commerciaux	Commercial-off-the-shelf
AMDE	Analyse de modes de défaillance et de leurs effets	Failure mode and effects analysis
APP	Analyse par arbre de pannes	Fault tree analysis
IP	Protocole internet	Internet protocol
TI	Technologie de l'information	Information technology
KSLOC	Kilo (millier) de lignes sources de code	Kilo-(thousand) source lines of code
ODC	Classification orthogonale des fautes	Orthogonal defect classification

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Abréviation	Terme en français	Terme en anglais
BDF	Bloc-diagramme de fiabilité	Reliability block diagram
USB	Bus USB	Universal serial bus

4 Présentation de la sûreté de fonctionnement du logiciel

4.1 Logiciels et systèmes logiciels

Un logiciel est une entité virtuelle. Dans le cadre de la présente norme, un logiciel fait référence aux procédures, programmes, codes, données et instructions pour le contrôle du système et le traitement de l'information. Un système logiciel comprend un ensemble intégré d'éléments logiciels, tels les programmes informatiques, procédures et codes exécutables, et est intégré dans un hôte physique du matériel de traitement et de commande pour réaliser l'exploitation du système et fournir des fonctions de performance. La hiérarchie du système logiciel peut être considérée comme une structure représentant l'architecture du système et composée de programmes logiciels de sous-système et d'unités logicielles de niveau inférieur. Une unité logicielle peut être soumise à essai comme spécifié dans la conception d'un programme. Dans certains cas, deux unités logicielles ou plus sont nécessaires pour former une fonction logicielle. Le système inclut les éléments matériels et logiciels qui interagissent entre eux pour fournir des fonctions utiles pour rendre les services de performance exigés.

Dans un système matériel/logiciel combiné, les éléments logiciels du système ont deux rôles principaux: a) logiciel d'exploitation à exécuter en continu pour supporter les éléments matériels lors de l'exploitation du système; et b) logiciel d'application à exécuter, si nécessaire, à la demande de l'utilisateur pour fournir des services clients spécifiques. L'analyse de la sûreté de fonctionnement des sous-systèmes logiciels doit considérer les facteurs de temps d'application des logiciels dans le profil opérationnel du système et les éléments logiciels qui nécessitent une exploitation du système à plein temps. La modélisation des logiciels est nécessaire pour le bilan de fiabilité et l'évaluation de la sûreté de fonctionnement des systèmes logiciels.

Les aspects humains de la sûreté de fonctionnement [1][1] jouent un rôle pivot pour la conception et la mise en œuvre efficaces des logiciels. L'interface homme-machine et l'environnement d'exploitation influencent le résultat de l'interaction logicielle et matérielle et affectent la sûreté de fonctionnement des performances du système. Ceci conduit à un besoin stratégique de conception de la sûreté de fonctionnement du logiciel et d'efforts de maintenance dans le processus de cycle de vie du logiciel [2].

4.2 Sûreté de fonctionnement du logiciel et organisations logicielles

La sûreté de fonctionnement du logiciel est atteinte grâce à une conception correcte et une intégration appropriée dans l'exploitation du système. Cette norme présente une approche où les techniques de sûreté de fonctionnement existantes et les pratiques d'excellence établies peuvent être identifiées et utilisées pour la conception et la mise en œuvre de la sûreté de fonctionnement du logiciel. Les systèmes de gestion de la sûreté de fonctionnement [3, 4] décrivent où les activités de sûreté de fonctionnement correspondantes peuvent être mises en œuvre efficacement dans les processus du cycle de vie. La réalisation de la sûreté de fonctionnement du logiciel est influencée par

- la politique de gestion et la direction technique;
- les processus de conception et de mise en œuvre;
- les besoins spécifiques au projet et les environnements d'application.

Les organisations logicielles sont des groupes organisés et gérés qui ont des personnes et établissements avec des responsabilités, pouvoirs et relations impliquant des logiciels dans le

1 Les références entre crochets se réfèrent à la bibliographie.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

cadre de leurs activités périodiques. Elles existent dans les gouvernements, les organismes publics et privés, les entreprises, les associations et les institutions. Les organisations logicielles sont structurées en fonction de besoins financiers et d'environnements d'application spécifiques pour diverses combinaisons de fournitures de développement, d'exploitation et de service.

Les organisations logicielles-types incluent celles qui

a) développent des logiciels comme produit principal,

b) développent des équipements avec logiciels intégrés,

c) fournissent une assistance logicielle aux clients,

d) exploitent et entretiennent les réseaux et systèmes logiciels.

L'Annexe A décrit la catégorisation des logiciels et applications logicielles fournis par les organisations logicielles-types.

4.3 Relation entre la sûreté de fonctionnement du logiciel et du matériel

Le comportement et les caractéristiques de performance du logiciel diffèrent, du point de vue de la sûreté de fonctionnement, de ceux rencontrés pour les matériels. Les codes logiciels sont créés par les hommes. Ils sont sujets aux erreurs humaines, qui sont influencées par l'environnement de conception et la culture organisationnelle. Alors que la plupart des données de défaillance d'un composant matériel sont bien documentées et rencontrées dans l'environnement d'utilisation, la nature des défauts logiciels et la traçabilité de leurs causes et effets ne sont pas faciles à déterminer dans l'exploitation du système. Dans la plupart des cas, les défauts logiciels conduisant à des défaillances du système ne peuvent pas être dupliqués de manière cohérente. Les actions correctives sur les défaillances de système en raison de défauts logiciels ne garantissent pas l'élimination totale des causes profondes du problème logiciel.

Un bogue, après avoir été déclenché, entraîne une défaillance logicielle (évènement) et se manifeste sous forme de défaut logiciel (état). Tous les défauts logiciels qui entraînent l'incapacité du logiciel à accomplir ses fonctions prévues sont observés par l'utilisateur du logiciel. Les défauts et bogues entraînent des problèmes pour exécuter le logiciel comme prévu. Le logiciel contenant des bogues pourrait continuer à accomplir sa fonction prévue sans que cela soit visible pour l'utilisateur. Les bogues pourraient entraîner des défaillances, mais également générer des nuisances qui n'affectent pas une certaine fonction. Un défaut logiciel peut entraîner une défaillance du système, qui peut présenter un symptôme de défaillance systématique.

Les systèmes logiciels et produits matériels présentent également de nombreuses similitudes. Ils sont tous les deux gérés via leurs étapes de conception et de développement, et sont suivis par intégration, essai et production. La découverte de défaillances et de défauts latents a lieu au moyen d'un processus rigoureux d'analyse, d'essai et de vérification avec des niveaux élevés d'essai ou de couverture des défauts. Les hauts niveaux de couverture du processus de vérification sont déterminés par l'évaluation de son pourcentage de détection des défauts ou la probabilité de détection des défauts. Alors que les techniques de gestion sont similaires, elles présentent également des différences [5, 6], par exemple:

- Un logiciel n'a pas de propriétés physiques, contrairement à un matériel. Un logiciel ne s'use pas. Les défaillances attribuables aux défauts logiciels apparaissent sans avertissement préalable et ne fournissent souvent aucune indication qu'elles se sont produites. Le matériel fournit souvent une période d'usure progressive et éventuellement une dégradation progressive jusqu'à atteindre une condition de défaillance.
- Les changements apportés aux logiciels sont flexibles; ils demandent moins de temps ou sont moins coûteux que les modifications de conception du matériel. Les changements apportés au matériel nécessitent un certain nombre de réglages qui demandent du temps, en ce qui concerne le matériel d'équipement, l'approvisionnement en matériel, la fabrication, l'assemblage et la documentation. Cependant, les essais de régression des

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

grands programmes logiciels complexes peuvent être limités en termes de temps et de coût.

- La vérification et l'essai du matériel sont simplifiés car il est possible d'effectuer des essais limités avec des connaissances de la physique de l'appareil pour analyser et prévoir un comportement. Les essais de logiciel peuvent également être simplifiés avec les essais de non-régression et l'analyse pour vérifier les modifications mineures apportées aux logiciels en raison d'une cause déterministe des défaillances. Cependant, les modifications mineures pour corriger les causes probabilistes des défaillances logicielles, telles les situations de concurrence, peuvent conduire à des cycles d'essai et de vérification très élaborés pour montrer la correction adéquate du problème.
- Les actions de réparation et de maintenance remettent généralement le matériel en état de fonctionnement sans modifications de conception. La réparation et la maintenance des logiciels impliquent des modifications de conception avec de nouveaux services ou versions logicielles pour corriger ou rectifier les défauts logiciels.

4.4 Interaction du logiciel et du matériel

L'interaction du logiciel et du matériel a lieu dans l'exploitation du système. Les problèmes de sûreté de fonctionnement existent dans l'interface entre le matériel et le système d'exploitation. Les problèmes sont généralement résolus en intégrant les techniques de détection et de correction des erreurs et le traitement des exceptions du logiciel et du système d'exploitation pour minimiser les défauts physiques, ainsi que les erreurs d'informations et de temporisation qui existent dans l'interaction. L'arrivée des processeurs multi-cœur a permis le traitement multifilière redondant et amélioré la sûreté de fonctionnement dans l'exécution du système. Ceci permet à l'utilisateur, au programmeur ou à l'architecte de système d'influencer et d'exploiter la redondance inhérente aux processeurs multi-cœur pour améliorer la détection et la reprise sur erreurs. Ceci offre également une opportunité de reprise à partir d'erreurs logicielles ou transitoires qui affectent le matériel ou le logiciel ou les deux. Il convient de prendre en considération l'exploitation de la complexité accrue dans la redondance multi-cœur dans ces applications.

Dans tout système de commande, le système contrôle certains processus physiques des équipements réels, tels des capteurs et actionneurs qui peuvent tomber en panne lors de l'exploitation du système. La plupart de ces équipements comportent des logiciels intégrés non accessibles au concepteur ou à l'architecte du système, tels les capteurs intelligents qui comportent des fonctions de détection des erreurs, de redondance et de correction des erreurs, qui sont commandées par le logiciel intégré. Il est important d'étudier les algorithmes de contrôle logiciel. Ceci a pour but de garantir que les algorithmes de contrôle sont résilients aux données incorrectes et aux valeurs manquantes du capteur, qu'ils peuvent détecter tout défaut de commande et sont capables de le neutraliser ou de retourner à un état de sécurité. Le retour du capteur est essentiel pour confirmer la bonne réalisation de la commande. Il convient que le mécanisme de retour inclue un contrôle indépendant des effets de la commande effectuée. Il convient de considérer dans la conception du système de contrôle logiciel, le comportement du système de contrôle, les hypothèses et modes de défaillance.

Toute introduction intentionnelle et malveillante de défauts matériels pour déjouer ou faire échouer les algorithmes logiciels peut se produire lorsque le système est exposé à une cyber-attaque délibérée. Il est possible, par exemple, d'introduire les défauts matériels dans un système cryptographique pour extraire la clé ou d'injecter un virus dans le périphérique USB qui est utilisé pour initialiser une machine à voter. L'interaction logicielle et matérielle pourrait générer de graves problèmes pour l'exploitation du système et affecter la sûreté de fonctionnement du système.

Les problèmes d'interopérabilité associés à l'interaction logicielle et matérielle peuvent également exister lorsque le logiciel est mal réutilisé dans un environnement différent ou pour une application différente.

La solution aux problèmes de sûreté de fonctionnement liés à l'interaction logicielle et matérielle consiste à améliorer la compréhension du fonctionnement du nouveau système technologique, et à effectuer avec précaution l'évaluation de la sûreté de fonctionnement et

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

les essais pour considérer complètement les effets des défaillances matérielles sur le système logiciel.

5 Ingénierie et application de la sûreté de fonctionnement du logiciel

5.1 Cadre du cycle de vie du système

Il convient d'établir un cadre du cycle de vie du système pour guider le développement des produits et la mise en œuvre du système. Ce cadre est utilisé pour définir le cycle de vie du système et diriger la performance des processus du cycle de vie du système. La norme CEI 60300-3-15 décrit l'ingénierie de la sûreté de fonctionnement du système et de la mise en œuvre du cycle de vie, qui est basée sur les processus techniques de la norme ISO/CEI 15288 [7]. Ceci s'applique à tout système, composé de matériels ou de logiciels, ou des deux.

5.2 Mise en œuvre du projet de sûreté de fonctionnement du logiciel

Il convient de planifier, de coordonner et de gérer les activités de génie logiciel pendant le cycle de conception et la durée de vie utile du cycle de vie du système en fonction de leurs homologues matériels. Les activités d'ingénierie pendant la durée de vie utile impliquent des modifications de conception qui seraient dues à des taux de défaillance élevés dans l'application client, ou à l'obsolescence du matériel tout en fournissant des pièces de rechange pour la poursuite des opérations. Comme le matériel change pendant le cycle de vie du produit, le logiciel nécessiterait également un changement. Les modifications apportées au logiciel sont nécessaires car la conception du système nécessite une post/rétro-compatibilité entre les différentes versions et configurations de la conception du système.

Il convient d'intégrer les activités de sûreté de fonctionnement dans les plans de projets correspondants et dans les tâches d'ingénierie du système pour une conception, réalisation, mise en œuvre, exploitation et maintenance efficaces du système. Les lignes directrices relatives à l'ingénierie de la sûreté de fonctionnement dans les systèmes selon la norme CEI 60300-3-15 s'appliquent à cette norme. Les lignes directrices relatives aux aspects logiciels de la sûreté de fonctionnement incluent les procédures recommandées suivantes pour la réalisation de la sûreté de fonctionnement du logiciel dans la mise en œuvre du projet logiciel:

a) identifier les objectifs et exigences d'application du logiciel concernant le cycle de vie du logiciel (voir 5.3) et l'environnement d'application (voir l'Article A.2);

b) identifier les attributs de sûreté de fonctionnement du logiciel applicables (voir 5.4) concernant le projet logiciel;

c) examiner l'adéquation des processus de gestion de la sûreté de fonctionnement et des ressources disponibles pour soutenir le développement et la mise en œuvre du projet logiciel (voir 5.5);

d) définir les exigences et objectifs de sûreté de fonctionnement du logiciel (voir 5.6, Annexe B);

e) classifier les défauts logiciels (voir 5.7) et identifier les mesures logicielles correspondantes (voir 6.2, Annexe E) pour la mise en œuvre de la stratégie de sûreté de fonctionnement du logiciel (voir 5.8);

f) appliquer la méthodologie de sûreté de fonctionnement correspondante pour la conception et la réalisation des logiciels (voir 6.1, 6.3);

g) initier l'amélioration de la sûreté de fonctionnement, si possible en tenant compte des diverses contraintes et limitations pour la personnalisation du projet (voir 6.4, 7.2);

h) surveiller le processus de développement et de mise en œuvre pour le contrôle et le retour d'expérience afin de soutenir l'exploitabilité du système et garantir la sûreté de fonctionnement lors de l'exploitation du système (voir l'Article 7).

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

5.3 Activités du cycle de vie du logiciel

Le cycle de vie du logiciel inclut les activités suivantes:

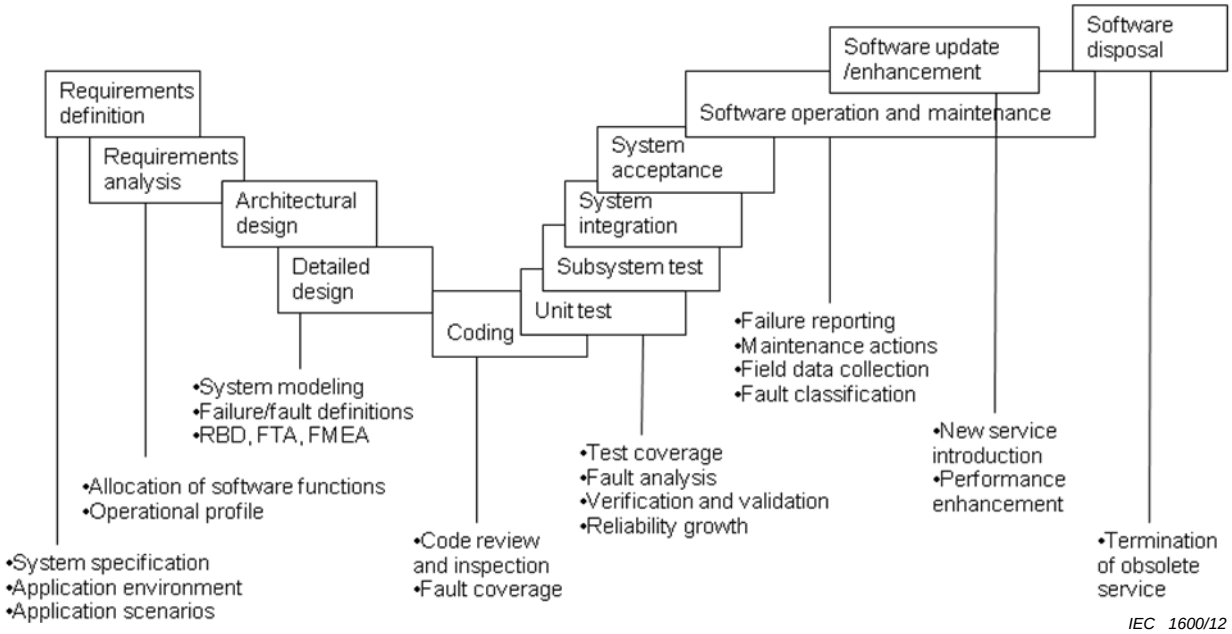
- *la définition des exigences* identifie les exigences du système pour les éléments matériels et logiciels combinés en réponse aux besoins et contraintes des utilisateurs des applications du système;
- *l'analyse des exigences* détermine les options de conception faisables et transforme les exigences du système pour les applications de service en une vue technique de la conception et du développement des sous-systèmes et système matériels et logiciels et du développement du système;
- *la conception architecturale* apporte une solution pour satisfaire aux exigences du système par allocation d'éléments de système dans les blocs de construction du sous-système afin d'établir une structure de base pour la décomposition du sous-système logiciel et d'identifier les fonctions logicielles correspondantes afin de satisfaire aux exigences spécifiées;
- *la conception détaillée* fournit un concept de chaque fonction identifiée dans l'architecture du système et crée les unités logicielles et interfaces nécessaires pour la fonction qui peut être affectée au logiciel, au matériel ou aux deux. Les fonctions affectées au logiciel sont définies avec des détails suffisants pour permettre le codage et l'essai. La fonction logicielle peut être identifiée comme sous-système logiciel et identifiée comme élément de configuration logiciel pour le contrôle de la conception;
- *la réalisation* produit les unités logicielles exécutables qui satisfont aux critères de vérification et exigences de conception, y compris les activités de niveau inférieur de

  – *codage* des unités logicielles;
  – *l'essai de l'unité* pour vérifier si l'unité logicielle satisfait aux exigences de conception;
  – *l'essai du sous-système* pour vérifier si les fonctions du programme logiciel satisfont aux exigences de conception;
- *l'intégration* assemble les unités logicielles et sous-systèmes en fonction de la configuration de la conception architecturale et installe le système logiciel complet dans le système matériel hôte pour l'essai;
- *l'acceptation* confirme la capacité du système et valide les applications logicielles pour la fourniture des services de performances requis pour les opérations spécifiées dans l'environnement de destination; les essais d'acceptation des logiciels incluent les activités de niveau inférieur destinées à
  – *accroître la fiabilité du système logiciel via l'essai de croissance de fiabilité*; l'essai est effectué après avoir entièrement intégré et exécuté le système logiciel dans les conditions d'exploitation sur le terrain simulées, représentant l'environnement de destination;
  – *valider l'acceptation du système logiciel en vue de la validation client, via l'essai de la qualification;*
- *l'exploitation et la maintenance du logiciel* engagent le logiciel dans l'exploitation du système, supportent la capacité opérationnelle du système et répondent aux demandes de services d'application pour fournir des services opérationnels spécifiques;
- *la mise à jour/amélioration du système* améliore la performance des logiciels avec de nouvelles fonctions;
- *l'élimination du logiciel* met fin à la fourniture du service logiciel spécifique.

L'Annexe B présente les exigences-types du système logiciel et les activités de sûreté de fonctionnement associées pour les étapes du cycle de vie du logiciel.

La Figure 1 illustre les principales activités de sûreté de fonctionnement qui sont importantes pour le cycle de vie du logiciel identifié pour la mise en œuvre du projet.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe



Légende

Anglais	Français
Requirements definition	Définition des exigences
Requirements analysis	Analyse des exigences
Architectural design	Conception architecturale
Detailed design	Conception détaillée
System modeling Failure/fault definitions RBD, FTA, FEMA	Modélisation du système Définitions des défaillances/défauts BDF, AAP, AMDE
Allocation of software functions Operational profile	Allocation de fonctions logicielles Profil opérationnel
System specification Application environment Applications scenarios	Spécification du système Environnement d'application Scénarios d'applications
Coding	Codage
Unit test	Essai de l'unité
Subsystem test	Essai du sous-système
System integration	Intégration du système
System acceptance	Réception du système
Software operation and maintenance	Fonctionnement et maintenance du logiciel
Software update/enhancement	Mise à jour/amélioration du logiciel
Software disposal	Elimination du logiciel
Code review and inspection Fault coverage	Examen et inspection du code Couverture des défauts
Test coverage Fault analysis Verification and validation Reliability growth	Couverture de l'essai Analyse des défauts Vérification et validation Croissance de la fiabilité
New service introduction Performance enhancement	Introduction d'un nouveau service Amélioration des performances
Termination of obsolete service	Fin du service obsolète

Figure 1 – Activités du cycle de vie du logiciel

5.4 Attributs de sûreté de fonctionnement du logiciel

Les attributs de sûreté de fonctionnement du logiciel sont les caractéristiques de performance inhérentes au système logiciel par conception. Il convient de prendre en considération les attributs de performance associés à une application spécifique pour l'intégration dans la

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

conception et la construction du système afin d'atteindre les objectifs de sûreté de fonctionnement du système matériel/logiciel.

Les principaux attributs de sûreté de fonctionnement du logiciel ou les caractéristiques de sûreté de fonctionnement inhérentes au logiciel contribuant aux objectifs de sûreté de fonctionnement du système incluent:

- *la disponibilité:* pour la disponibilité d'exploitation du logiciel;
- *la fiabilité*: pour la continuité du service du logiciel;
- *la maintenabilité:* pour la facilité de modification, mise à niveau et amélioration du logiciel;
- *la récupérabilité*: pour la restauration du logiciel à la suite d'une défaillance, avec ou sans interventions externes;
- *l'intégrité*: pour l'exactitude des données logicielles.

Les attributs de performance associés à une application spécifique, qui contribuent aux objectifs de sûreté de fonctionnement du système, incluent, mais sans limitation, ce qui suit:

- *la sécurité*: pour la protection contre les intrusions dans l'application et l'utilisation du logiciel;
- *la sûreté*: pour la protection contre les dommages dans l'application et l'utilisation du logiciel;
- *l'exploitabilité*: pour une exploitation robuste, à tolérance de pannes et sans coupure;
- *la réutilisabilité*: pour l'utilisation d'un logiciel existant pour d'autres applications;
- *la soutenabilité*: pour le maintien des performances du système à l'aide des ressources logistiques et de maintenance;
- *la portabilité*: pour les applications inter-plateformes.

Ces caractéristiques de sûreté de fonctionnement inhérentes ainsi que les attributs de performance associés à une application spécifique représentent la base de la conception et de l'application du système logiciel.

5.5 Environnement de conception du logiciel

L'objectif de gestion de la sûreté de fonctionnement est de fournir un environnement de conception équilibré pour la créativité en tenant compte des ressources budgétaires du projet, du planning et des dates de livraison. Les organisations associées au développement de logiciels et à la fourniture de services logiciels sont orientées vers les applications utilisateur. La personnalisation des projets de logiciels est nécessaire pour gérer l'attribution des ressources disponibles et pour rechercher des options de conception appropriées pour une mise en œuvre efficace. La sélection et l'adoption de processus applicables pour l'ingénierie de la sûreté de fonctionnement dans un système logiciel spécifique sont réalisées via le processus de personnalisation du projet pour une gestion efficace de la sûreté de fonctionnement. Les tâches recommandées pour la mise en œuvre du processus de personnalisation figurent dans 7.2. Il convient d'explorer les opportunités d'externalisation de la construction, de réutilisation logicielle et d'application de logiciels commerciaux (COTS) pour l'intégration du système.

L'environnement de conception des logiciels est basé sur un processus organisé destiné à promouvoir les bonnes pratiques de conception pour la génération de codes sans erreur, minimiser les erreurs de définition des exigences et garantir les essais de validation pour l'édition de logiciels. Les aspects culturels de l'approche de la gestion des logiciels adoptent souvent un concept de modèle d'évolution des capacités pour le développement de l'infrastructure [8]. Ceci est similaire à la mise en œuvre formelle de l'*intégration du modèle d'évolution des capacités (CMMI)* [9] décrite en Annexe C pour la gestion des processus logiciels. Le développement de logiciels est un processus technique suivant des disciplines de génie logiciel et des instructions d'application établies. Il convient d'inclure l'environnement

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

de conception des logiciels et les principes pratiques dans la politique de l'organisation pour définir les missions et objectifs en vue de la réalisation de la sûreté de fonctionnement.

La conception de logiciels fait souvent appel aux applications des outils CASE (génie logiciel assisté par ordinateur). Un système automatisé efficace assure la précision de calcul, la traçabilité des données, la gestion de la configuration et un moyen de collecte des mesures ou entrées métriques requises dans les modèles automatisés. La plupart des systèmes de collecte de données pour le signalement des défaillances sur le terrain, l'analyse et les actions correctives sont automatisés pour les mêmes raisons. Les données historiques sur les produits et services logiciels sont un élément indispensable et de valeur.

5.6 Définition des exigences et objectifs de sûreté de fonctionnement du logiciel

Il convient de définir les exigences du logiciel pour les étapes de son cycle de vie. Il convient d'identifier les activités de sûreté de fonctionnement applicables pour la mise en œuvre correspondante à chaque étape. La planification pour la mise en œuvre des activités de sûreté de fonctionnement est importante. Les applications de sûreté de fonctionnement sont dépendantes du temps et ont un énorme impact sur le coût du cycle de vie du système [10]. La personnalisation du projet est essentielle pour les améliorations de conception et pour la résolution des contraintes. Il convient que les exigences et les objectifs de sûreté de fonctionnement associés au logiciel fassent partie des spécifications générales des produits logiciels. La stratégie relative à la mise en œuvre de la sûreté de fonctionnement du logiciel est décrite au 5.8. La méthodologie relative à l'ingénierie de la sûreté de fonctionnement dans les modules ou unités logiciel(le)s et la création de l'architecture du système est traitée à l'Article 6. Le processus de personnalisation est décrit au 7.2.

Les activités de sûreté de fonctionnement associées aux exigences logicielles sont spécifiques aux applications. Elles reflètent les besoins de conception et de mise en œuvre des logiciels pour fournir les fonctions du système requises pour les applications de performance de services. Les approches systématiques pour la mise en œuvre d'activités de sûreté de fonctionnement au cours du cycle de vie du logiciel garantissent la réalisation des objectifs de sûreté de fonctionnement. Les objectifs de sûreté de fonctionnement spécifiques sont dérivés de la sélection d'attributs de sûreté de fonctionnement-clés et des mesures quantitatives correspondantes. Les exigences de sûreté de fonctionnement du logiciel peuvent être formulées pour des projets spécifiques à l'aide des informations de base contenues dans l'Annexe B.

Les conditions d'influence sur les spécifications de sûreté de fonctionnement du système matériel/logiciel combiné sont décrites dans les spécifications de sûreté de fonctionnement du système [11]. Il convient de considérer les facteurs d'influence suivants affectant la réalisation de la sûreté de fonctionnement dans le développement des logiciels:

- la culture de conception, le processus d'évolution des capacités, et l'expérience de l'organisation dans le domaine de la conception, du développement et de la mise en œuvre des logiciels (voir Annexe C);
- la compréhension des environnements d'application, des besoins des utilisateurs, et la modification de la dynamique du marché pour le développement d'une nouvelle plateforme ou fonctionnalité pour la mise en œuvre pratique;
- la documentation des processus, tels le signalement des défaillances, la collecte des données, la gestion de la configuration des logiciels pour le contrôle des versions logicielles et la maintenance des dossiers historiques;
- l'application de règles de conception de logiciels pour empêcher les défauts en contrôlant les processus de conception destinés à optimiser les performances des logiciels en termes de complexité des logiciels, de complexité des programmes et de complexité fonctionnelle;
- l'utilisation efficace des méthodes et outils logiciels applicables, tels que la conception structurée, la tolérance des pannes, la revue de conception [12] et la gestion des défauts logiciels pour améliorer la croissance de la fiabilité;

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- la sélection d'un niveau supérieur approprié de langages de programmation plus adaptés pour un développement structuré des logiciels spécifiques;
- les exigences établies pour la qualification et la mesure des caractéristiques de sûreté de fonctionnement du logiciel.

5.7 Classification des défauts logiciels

Les défauts logiciels peuvent être classés comme des défauts de spécification, des défauts de conception, des défauts de programmation, des défauts insérés dans le compilateur, ou des défauts introduits pendant la maintenance des logiciels.

La classification des défauts logiciels fournit un moyen de collecter et regrouper les informations sur les défauts logiciels. Le processus de classification aide les concepteurs de logiciels à découvrir les configurations de défaut inhabituelles pour les actions correctives. L'objectif est d'éliminer la récurrence de la classe des défauts similaires.

La *classification orthogonale des défauts* (ODC) [13] est une méthode utilisée dans le génie logiciel pour l'analyse des données de défaut logiciel (panne). L'ODC porte sur les effets de causalité des problèmes de qualité en matière de conception de logiciels et de code dans un environnement de langage procédural. Une panne est le non-respect d'une exigence liée à une utilisation prévue ou spécifiée du logiciel. A cet égard, une panne due à l'incapacité du logiciel à exécuter ses fonctions requises affiche les caractéristiques des attributs de panne dans le système ODC. Les attributs de panne sont la signature d'une panne contenant des informations liées au défaut logiciel. La méthode ODC collecte les informations de défaut logiciel des attributs de panne pour l'analyse et la modélisation. L'analyse des données ODC fournit une méthode de diagnostic de valeur pour évaluer la maturité du produit logiciel à diverses étapes du cycle de vie du logiciel. L'ODC peut également être utilisée pour évaluer le processus en analysant les types de déclencheurs pour identifier les besoins techniques spécifiques pour stimuler les déclencheurs manquants. L'analyse causale des données de défaut (panne) présente un moyen de réduction des défauts logiciels et d'amélioration de la fiabilité.

Les attributs de panne ODC sont classés par *Activité*, *Déclencheur*, *Cible*, *Type de panne*, *Qualificatif de panne*, *Source*, *Impact* et *Age*. Ils sont normalement collectés et analysés pendant le développement de logiciels pour contribuer à l'amélioration de la conception. Les informations sur les pannes sont disponibles à deux moments spécifiques. Si un défaut est observé, les circonstances conduisant à l'exposition au défaut et l'impact probable pour l'utilisateur sont généralement connus. Si un défaut est éliminé avant d'être réparé, la nature exacte du défaut et l'étendue de la réparation sont connues. Les catégories d'ODC capturent la sémantique d'un défaut (panne) à partir de ces deux perspectives. En définissant les *activités* pendant le processus de développement et leur allocation dans les *déclencheurs* de l'ODC, l'ODC fournit des aperçus intéressants et personnalisés des données de défaut (panne) pour l'organisation du développement de logiciels.

Les groupes d'attributs de défaut sont résumés: a) lorsqu'un défaut est observé, c'est ce qu'on appelle la *section d'ouverture*, et b) lorsqu'un défaut est éliminé, c'est ce qu'on appelle la *section de fermeture*. L'ODC est plus utile dans les organisations logicielles matures où de vastes données sont normalement collectées et analysées pour l'amélioration du produit logiciel.

L'Annexe D présente un résumé de la classification des attributs des défauts logiciels.

5.8 Stratégie relative à la mise en œuvre de la sûreté de fonctionnement du logiciel

5.8.1 Evitement des défauts logiciels

Les codes logiciels sont générés pour produire un produit logiciel. Une erreur effectuée pendant la conception pourrait se manifester pour devenir un défaut logiciel conduisant à une défaillance du système. Comme les défauts représentent la principale cause des défaillances de système, la prévention des défauts avant leur introduction pendant la conception, telle

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

l'examen de code, et l'élimination des défauts résiduels qui ont échappé à la détection au moyen d'essais sont les approches communes pour réduire l'existence de problèmes pendant le cycle de vie du logiciel. La stratégie recommandée en matière d'évitement des défauts logiciels inclut la prévention et l'élimination des défauts.

a) Prévention des défauts

- Etablir les objectifs de prévention des défauts dans les disciplines de génie logiciel.
- Initier l'examen des exigences.
- Effectuer une interaction avec l'utilisateur anticipée et une amélioration des exigences logicielles.
- Introduire des méthodes formelles, si applicables et possibles.
- Mettre en œuvre des techniques systématiques pour la réutilisation des logiciels et la garantie d'application.

b) Elimination des défauts

- Détecter et éliminer l'existence de défauts logiciels au moyen d'essais.
- Effectuer une inspection formelle concernant la localisation des défauts, la correction des défauts et le contrôle des corrections.
- Effectuer des actions de maintenance corrective et d'amélioration pendant l'exploitation des logiciels.

5.8.2 Contrôle des défauts logiciels

Les défauts logiciels sont difficiles à détecter et l'élimination des défauts peut être réalisée avec divers moyens, y compris des essais et contrôles rigoureux de logiciels. Les essais exhaustifs de logiciels sont souvent limités par des contraintes de temps et de coût au niveau de la gestion de projets. L'assurance de la sûreté de fonctionnement, fondée uniquement sur des essais, ne garantit pas l'élimination complète des défauts. Le contrôle des défauts logiciels utilise des méthodes de tolérance et de prévision des défauts pour minimiser la manifestation de défauts logiciels latents ou bogues qui peuvent persister après l'autorisation d'utilisation du logiciel. La stratégie recommandée en matière de contrôle des défauts logiciels inclut la tolérance des pannes et la prévision des défauts/défaillances.

a) Tolérance des pannes

- Etablir la méthodologie relative au confinement des défauts, à la détection des défauts et à la récupération des défauts.
- Mettre en œuvre les systèmes diversifiés de conception des logiciels et de traitement de secours.
- Appliquer des techniques de programmation multi-versions.
- Mettre en œuvre des techniques de programmation d'auto-contrôle.

b) Prévision des défauts/défaillances

- Etablir les relations défaut/défaillance dans l'environnement d'exploitation.
- Etablir le système de collecte de données pour capturer les données pertinentes.
- Mener l'essai de croissance de la fiabilité, si applicable.
- Développer et mettre en œuvre les modèles de fiabilité pour l'estimation des défauts/défaillances.
- Améliorer les techniques de prévision pour la projection dans le temps de l'édition de la version logicielle.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

6 Méthodologie relative aux applications de sûreté de fonctionnement du logiciel

6.1 Pratiques de développement de logiciels pour la réalisation de la sûreté de fonctionnement

La maturité des capacités en matière de développement de logiciels reflète la capacité d'une organisation à développer des logiciels et des produits stables pour les applications prévues. Les techniques d'évitement et de contrôle des défauts ci-après sont recommandées en matière d'intégration, si applicable, dans le développement de logiciels:

a) standardiser les méthodes en matière de conception architecturale de niveau supérieur, de conception détaillée, de codage, de construction, et de documentation pour faciliter les communications et l'évitement des défauts;

b) construire des conceptions modulaires pour les unités logicielles et les sous-systèmes avec des fonctions et interfaces logicielles bien définies en construisant des unités logicielles simples, séparées et indépendantes pour faciliter l'interaction de la conception, la maintenance, la traçabilité des erreurs, la minimisation des défauts et l'élimination des bogues;

c) utiliser les configurations de conception, qui sont des solutions réutilisables générales de logiciels reconnus, sous forme de modèles pour résoudre les problèmes de conception de logiciels en vue d'accélérer le processus de développement;

d) instituer les méthodes de conception formelles, si appropriées, pour le contrôle et la documentation du processus de conception et de développement de logiciels;

e) utiliser les techniques d'*ingénierie de la fiabilité des logiciels* [14] pour l'évaluation et l'amélioration de la fiabilité des logiciels [15];

f) réutiliser les logiciels disponibles dans la bibliothèque des logiciels sur les unités logicielles et sous-systèmes reconnus pour une application et un profil opérationnel similaires afin de réduire le coût et le temps de développement, et minimiser l'introduction de défauts de conception;

g) développer les méthodes d'essai de non-régression pour garantir la fonctionnalité des logiciels existants lorsque la nouvelle fonctionnalité est introduite ou l'élimination des défauts est réalisée;

h) soumettre à essai les unités logicielles et sous-systèmes pour vérifier les fonctions de conception de niveau inférieur et valider la performance du système architectural de conception de haut niveau intégré pour une élimination progressive des bogues pour empêcher la propagation des défauts;

i) effectuer des inspections et revues des exigences de conception des logiciels, des codes logiciels, modes d'emploi, matériels de formation et documents d'essai pour détecter et éliminer le plus possible les erreurs; il convient de considérer et d'employer, si possible, différentes équipes d'examen pour la comparaison des résultats;

j) contrôler les changements pour réduire les occurrences des défauts, au travers, par exemple, de processus de contrôle des versions et des changements dans la gestion de la configuration des logiciels;

k) analyser les causes profondes des problèmes et mettre en œuvre les actions correctives appropriées pour une amélioration continue des logiciels;

l) établir un système de collecte de données pour emmagasiner une base de connaissances des défauts logiciels et d'historiques de performances.

6.2 Mesures de la sûreté de fonctionnement du logiciel et collecte de données

Les mesures de la sûreté de fonctionnement du logiciel sont des mesures des caractéristiques de sûreté de fonctionnement d'un système logiciel. La réalisation des mesures fournit une échelle quantitative et une méthode pour déterminer la valeur d'une caractéristique spécifique associée au système logiciel. Ces mesures conformes aux normes de l'industrie sont obtenues par mesure directe ou par déduction. Elles sont utilisées pour les

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

mesures de performances du système logiciel. Les mesures suivantes des logiciels sont de facto des normes de l'industrie concernant l'application. Il convient de les considérer, si approprié, pour l'évaluation de la sûreté de fonctionnement du système logiciel.

a) *Disponibilité*: fournit une mesure du temps de disponibilité pendant la durée de fonctionnement du système.

b) *Fréquence de défaillance*: fournit une mesure du nombre de défaillances pendant la durée de fonctionnement du système.

c) *Temps de fonctionnement avant défaillance*: fournit une mesure de la période de temps sans défaillance.

d) *Temps de restauration*: fournit une mesure du temps nécessaire pour la restauration d'un système à partir d'une condition de défaillance (état d'indisponibilité) à un fonctionnement normal (état de disponibilité).

e) *Densité du défaut*: fournit une mesure du nombre de défauts contenus par kilo de lignes sources de codes (KSLOC) ou par point fonctionnel et est utilisée pour l'évaluation de la fiabilité des logiciels.

f) *Point fonctionnel*: fournit une mesure de la taille fonctionnelle du logiciel d'application pour la planification du projet de logiciels au moyen de la méthode d'*analyse des points fonctionnels* [16].

g) *Couverture des codes*: fournit une mesure du taux de couverture de test du code source et des branches logiques d'un programme logiciel; la *couverture des codes* est un indicateur de la rigueur des essais de logiciels, qui est utilisé pour représenter la *couverture des défauts* qui indique le pourcentage de défauts détectés pendant l'essai lors de l'exécution des codes.

h) *Taux d'élimination des défauts*: fournit une mesure du nombre de défauts détectés et corrigés dans un produit logiciel pour une période de temps ou une durée d'exécution du logiciel définie; le *taux d'élimination des défauts* est utilisé dans la croissance de la fiabilité pour établir la tendance à l'amélioration de la fiabilité.

i) *Défauts résiduels dans les logiciels*: fournit une mesure du nombre estimé de bogues encore présents dans le produit logiciel après l'essai pour élimination des bogues.

j) *Temps jusqu'à l'édition du logiciel*: fournit une mesure du temps estimé pour le planning d'édition d'un produit logiciel à partir de critères établis et selon un niveau acceptable de bogues encore présents dans le produit logiciel pour la gestion du projet de logiciels.

k) *Complexité logicielle*: fournit une mesure du degré de difficulté pour la conception et la mise en œuvre d'une fonction logicielle ou d'un système logiciel; les autres mesures de complexité basées sur le concept de complexité incluent la complexité du programme, la complexité fonctionnelle, la complexité opérationnelle; les métriques relatives à la complexité sont utilisées comme entrées pour les évaluations de fiabilité et les modèles de prédiction.

Il existe de nombreuses métriques utilisées pour diverses raisons pendant le cycle de vie du logiciel. Les métriques logicielles peuvent être regroupées en trois catégories principales pour faciliter la collecte des données.

1) *Métriques des données de défaut*: capturent les données de signalement des problèmes logiciels pour mesurer l'impact des défauts et l'efficacité du processus de signalement afin d'améliorer la maintenance des logiciels.

2) *Métriques des données de produit*: capturent les données relatives au produit logiciel en les classant par taille, fonctionnalité, complexité, lieu d'utilisation et autres caractéristiques pour faciliter les données rencontrées comme entrées pour profiter du développement de nouveaux produits. Les mesures fournissent un historique de performance et des données des divers groupes de produits logiciels.

3) *Métriques des données de processus*: capturent les données et conditions du processus de restauration de logiciels au moment de la détection et de l'élimination des défauts pour les entrées de modèle de fiabilité dans la prédiction de la fiabilité.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Le processus de collecte de données est critique pour mesurer les attributs de sûreté de fonctionnement et les caractéristiques de performance des logiciels. Il convient qu'un système de collecte de données efficace soit pratique à mettre en œuvre. Il convient que la quantité et le type de données soient relativement simples à collecter, faciles à interpréter pour l'analyse des données et utiles pour l'évaluation et l'amélioration de la sûreté de fonctionnement du logiciel. Les données collectées sont utilisées pour déterminer les tendances du système en matière de fiabilité, la fréquence et la durée nécessaires pour la maintenance des logiciels, le temps de réponse pour les appels de service, la restauration des performances dégradées et les exigences de support de maintenance.

L'Annexe E présente des exemples de mesures de données logicielles obtenues à partir de la collecte de données.

6.3 Evaluation de la sûreté de fonctionnement du logiciel

6.3.1 Processus d'évaluation de la sûreté de fonctionnement du logiciel

L'objectif de la mise en œuvre du processus de sûreté de fonctionnement du logiciel est de garantir la maturité du système logiciel pour la réalisation du développement et de la sûreté de fonctionnement. Le processus d'évaluation est le mécanisme permettant de garantir la vérification des exigences logicielles et la validation de la sûreté de fonctionnement du logiciel dans les résultats de performance du système. Le processus d'évaluation de la sûreté de fonctionnement intègre des activités de génie logiciel cruciales, adoptées à partir de pratiques industrielles établies [14]. Le processus suivant pour mener une évaluation de la sûreté de fonctionnement du logiciel est recommandé:

- identifier les besoins des utilisateurs et l'objectif de performance du système et développer la spécification relative à la sûreté de fonctionnement;
- établir un profil opérationnel des logiciels;
- allouer des attributs de sûreté de fonctionnement applicables;
- effectuer une analyse et évaluation de la sûreté de fonctionnement pour déterminer les options et solutions possibles;
- mener des essais et mesures des logiciels;
- mener une vérification du logiciel et une validation du système logiciel;
- effectuer une croissance de la fiabilité du logiciel et prévoir les tendances à l'amélioration;
- évaluer les résultats d'évaluation et les retours d'informations.

Les activités d'évaluation de la sûreté de fonctionnement du logiciel sont décrites dans les paragraphes suivants.

6.3.2 Spécification relative à la performance et à la sûreté de fonctionnement du système

Le but est d'identifier l'objectif de performance du système pour le développement d'une spécification relative à la sûreté de fonctionnement du système [11] si elle n'est pas fournie par le client ou l'utilisateur. Le processus suivant est recommandé:

- identifier le scénario de performance du système et l'environnement d'application;
- identifier les facteurs d'influence des performances;
- identifier les limites et interfaces du système avec des systèmes externes;
- identifier les attributs de performance du système;
- identifier l'architecture du système, la configuration matérielle/logicielle;
- identifier les fonctions matérielles et logicielles interopérables de la configuration du système;

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- caractériser et quantifier les attributs de sûreté de fonctionnement des fonctions matérielles et logicielles correspondantes, y compris la disponibilité, la fiabilité et la récupérabilité associées aux critères de support de maintenance.

Il convient que la documentation de la spécification relative à la sûreté de fonctionnement du système inclue les données suivantes dans le cadre de la spécification du système:

- identification du système;
- objectif de performance du système;
- profil opérationnel du système;
- objectifs de performance de la sûreté de fonctionnement du système;
- configuration du système;
- fonctions du système;
- exigences de sûreté de fonctionnement pour chaque fonction;
- exigences de support de maintenance du système.

Pour le système logiciel, il est essentiel de considérer le profil opérationnel qui affecte le temps d'exécution des fonctions logicielles pour les applications à la demande. Un *bloc-diagramme de fiabilité* [17] fonctionnel peut être construit pour représenter le système matériel/logiciel. Les blocs fonctionnels créés faciliteraient l'allocation des mesures de sûreté de fonctionnement du logiciel à chaque fonction logicielle établie en fonction de l'architecture du système logiciel et de la configuration du logiciel.

6.3.3 Etablir le profil opérationnel du logiciel

Un profil opérationnel est la séquence d'activités à exécuter par le système matériel/logiciel combiné pour remplir sa mission ou son objectif de service. La performance du système dépend fortement de l'environnement dans lequel le système fonctionne. L'environnement peut affecter les changements physiques du matériel, mais ne peut pas affecter les fonctions logicielles fournies par l'exécution des programmes logiciels pendant l'exploitation du système.

Le développement d'un profil opérationnel est une caractérisation quantitative de la manière dont le logiciel est utilisé. Les données opérationnelles et informations pertinentes sont généralement collectées au moyen d'enquêtes menées auprès des clients et acquises par les expériences sur le terrain. Les processus suivants sont recommandés pour le développement d'un profil opérationnel:

a) déterminer le profil client en établissant les besoins et les types de clients, par exemple: organisation ou personne devant acquérir ou acheter le système logiciel;

b) établir le profil utilisateur selon les différents types d'utilisateurs, par exemple: personne ou employé d'une organisation, ou systèmes d'application de logiciels interactifs, exploitant ou utilisant le système logiciel pour les applications spécifiques;

c) définir le profil de mode de système concernant la manière dont le système est exploité et dans quelle séquence ou dans quel ordre il est exprimé en termes de modes de fonctionnement, tels l'essai du logiciel pour l'amélioration de la maintenance, ou le traitement de données de lot normales lors de l'exécution du système logiciel;

d) déterminer le profil fonctionnel en évaluant chaque mode de système pour les fonctions de performance et les caractéristiques de service, par exemple créer un e-mail ou une consultation d'adresse pour satisfaire aux exigences fonctionnelles des logiciels;

e) déterminer le profil opérationnel en fonction des profils fonctionnels établis pour les fonctions de performance du système;

f) déterminer le profil d'information en collectant les données d'application des logiciels au cours du cycle de vie de développement du logiciel.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Le profil fonctionnel est une vue orientée vers l'utilisateur des capacités du système. D'un point de vue du développeur, le profil fonctionnel représente les opérations du système qui mettent en œuvre réellement les fonctions requises. D'un point de vue de la sûreté de fonctionnement, le profil opérationnel est un ensemble de différents scénarios d'exploitation du système et de leurs probabilités d'occurrence. Le profil opérationnel fournit les entrées nécessaires pour le développement d'essais-types pour simuler les opérations sur le terrain du système logiciel et les applications spécifiques de l'utilisation des fonctionnalités logicielles. L'exécution des essais-types pour l'essai du logiciel fournit des informations de valeur et la capture de données pour évaluer la fiabilité du logiciel sur le terrain et valider la fourniture de fonctions de maintenance et l'efficacité de la performance du support de maintenance.

6.3.4 Allocation d'attributs de sûreté de fonctionnement

L'allocation d'attributs de sûreté de fonctionnement et de mesures pour le système logiciel est basée sur le concept de fonctions architecturales du système de modélisation pour refléter les exigences de l'objectif de sûreté de fonctionnement du système. L'assignation de la valeur initiale des métriques de sûreté de fonctionnement applicables, telles la fiabilité et la disponibilité, est plus vraisemblablement basée sur les données historiques. Ces valeurs métriques sont affinées via un processus d'analyse et d'évaluation itératif. L'affectation de valeurs de fiabilité et de disponibilité aux divers sous-systèmes logiciels et unités fonctionnelles est effectuée en fonction de leur complexité, criticité, objectifs de performance réalisables en matière de fiabilité ou de disponibilité, et autres facteurs d'influence concernant le processus d'affectation.

Le développement du modèle du système pour le logiciel diffère considérablement du matériel en raison de ses caractéristiques de fonctionnement inhérentes. Pour chaque mode d'exploitation du système qui implique des fonctions de programme logiciel comme éléments de configuration, différents ensembles d'unités logicielles sont exécutés. Chaque mode a un temps unique d'application associé à la durée d'exécution de l'unité logicielle sur demande lors de l'exploitation du système. Ceci indique la durée de chaque mode de système. La modélisation du système logiciel inclut le nombre de lignes de codes sources dans chaque unité logicielle, la complexité du code et d'autres informations concernant les ressources de développement des logiciels, par exemple le langage de programmation et l'environnement de conception. Ils sont utilisés pour établir le taux de panne initial pour la prévision de la fiabilité ou de la disponibilité des éléments de configuration logiciels.

6.3.5 Analyse et évaluation de la sûreté de fonctionnement

Les activités suivantes d'analyse et d'évaluation de la sûreté de fonctionnement sont nécessaires pour supporter le développement du système logiciel. Le processus est itératif pour optimiser les exigences de conception de la sûreté de fonctionnement afin de satisfaire à l'objectif de performance du système. La modélisation de la disponibilité/fiabilité est utilisée pour l'analyse et l'évaluation des fonctions de performance temporelles du logiciel.

a) Modélisation des fonctions de disponibilité/fiabilité

Une simple approche pour analyser la disponibilité ou la fiabilité d'un système composé de matériels et de logiciels consiste à former un modèle structurel du système. Un modèle de disponibilité/fiabilité fonctionnelle pour le système matériel/logiciel combiné, composé de blocs fonctionnels, peut être construit à l'aide de la technique du *bloc-diagramme de fiabilité* (BDF) [17]. Le modèle est décomposé en modèles de sous-système séparés représentant les éléments matériels et logiciels constitutifs du système. L'analyse par arbre de pannes (AAP) [18], les chaînes de Markov [19] et les réseaux de Pétri [20] sont également utiles pour le développement du modèle de disponibilité/fiabilité du système. Par exemple, l'AAP peut être efficacement utilisée pour modéliser la fiabilité du système avec des portes dynamiques pour déterminer les fonctions de disponibilité/fiabilité matérielles et logicielles pour les améliorations de fiabilité [21]. Il convient de noter que le BDF et l'AAP sont logiquement équivalents. Le BDF met l'accent sur le succès, l'AAP sur l'échec.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Le modèle de disponibilité/fiabilité du sous-système matériel est composé de tous les éléments matériels du système avec les blocs fonctionnels de disponibilité/fiabilité construits, si approprié, pour représenter la structure du sous-système matériel et la configuration de la redondance. Ceci a pour but de faciliter la prévision des taux de défaillance des composants matériels individuels et la détermination de la disponibilité/fiabilité du sous-système matériel selon les techniques de prévision de la fiabilité. Il peut y avoir un ou plusieurs sous-systèmes matériels réalisant différentes fonctions dans la configuration du système.

Le modèle de fiabilité du sous-système logiciel est construit à l'aide d'unités logicielles comme blocs de construction pour fournir des fonctions de programmes logiciels. Une unité logicielle est le niveau le plus bas d'un élément logiciel configurable. Les unités logicielles ne tombent pas en panne de manière indépendante comme les composants matériels. Les codes logiciels sont des entités virtuelles qui ne sont pas sujettes à des modifications physiques. Les unités logicielles n'arrivent pas s'associer au profil opérationnel du système, ce qui affecte le système de configuration de la structure du modèle de fiabilité logicielle. Il est nécessaire que la modélisation de la fiabilité logicielle intègre les informations du profil opérationnel lors du développement de la structure de configuration logicielle. Le programme du sous-système logiciel peut comprendre une ou plusieurs unités logicielles constitutives pour fournir les fonctions requises. Un programme de sous-système logiciel résidant dans un sous-système matériel hôte est configuré pour former un élément de configuration logiciel. L'interopération et la dépendance mutuelle du sous-système logiciel et de son hôte matériel désigné sont nécessaires pour fournir des fonctions logicielles de sous-système spécifiques pour l'exploitation du système. Il peut y avoir plusieurs sous-systèmes logiciels et matériels combinés réalisant différentes fonctions dans la configuration de l'ensemble du système. L'Annexe F présente un exemple pour illustrer les interactions des fonctions de fiabilité matérielles/logicielles combinées pour dériver le taux de défaillance du système pour l'évaluation de la fiabilité.

Il convient que l'évaluation de la disponibilité du système matériel/logiciel combiné établisse d'abord les interactions des fonctions de fiabilité logicielles/matérielles combinées avant de dériver les fonctions de disponibilité du système. Le temps d'arrêt, ou le temps de restauration total du système pour la durée d'exploitation du système, est exigé pour l'évaluation de la disponibilité du système.

b) Détermination de la fiabilité des fonctions logicielles

Les défaillances logicielles peuvent se produire pendant l'exploitation du système. La détermination des taux de défaillance logicielle pour une utilisation dans la modélisation de la fiabilité nécessite que le logiciel soit traité comme un sous-système, qui réside dans un sous-système matériel hôte, configuré pour former un élément de configuration logiciel. Le sous-système logiciel peut exécuter une ou plusieurs de ses fonctions requises. Une fonction est une capacité du système à fournir un service requis du point de vue de l'utilisateur final. La fonction peut être accomplie par un élément de configuration logiciel, où la fonction de service requise est reconnue par le contrôle de version. Une unité logicielle conçue pour exécuter exactement une seule fonction peut être un élément de configuration. Un programme de sous-système logiciel nécessitant plusieurs unités logicielles pour exécuter une seule fonction peut également être un élément de configuration. Un sous-système logiciel peut comporter plusieurs programmes logiciels pour fournir un ensemble de fonctions associées. Chacun de ces programmes logiciels est un élément de configuration par définition. Le concept d'élément de configuration logiciel est considéré d'un point de vue du contrôle de version de conception du logiciel. L'élément de configuration logiciel est essentiel pour le suivi des modifications de conception. A chaque modification de conception est assignée une publication de version pour identification. La version logicielle doit être référencée pour suivre l'efficacité des améliorations de maintenance des logiciels. La tendance de la croissance de la fiabilité est établie par une indication de l'amélioration des performances avec le système exécutant la nouvelle version remplaçant l'ancienne version. Fournir les fonctions requises lors de l'exploitation du système est le défi à relever pour satisfaire à l'objectif de fiabilité du système.

Les fonctions logicielles qui incluent un système sont associées dans une configuration de synchronisation et une topologie de fiabilité.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

La configuration de la synchronisation est un problème lorsque les diverses fonctions sont actives et inactives pendant une période de temps spécifique pendant l'exploitation du système. Les principales relations de synchronisation entre les fonctions logicielles sont concurrentes et séquentielles. Les fonctions sont concurrentes si elles sont actives simultanément. Les fonctions sont séquentielles si elles sont actives les unes après les autres. Il est également possible de faire chevaucher partiellement les temps, ce qui entraîne une configuration de synchronisation concurrente/séquentielle hybride. On retrouve les fonctions logicielles actives concurrentes dans les systèmes qui sont desservis par plusieurs unités centrales, par exemple dans un système multi-traitement ou un système distribué. Les références de temps d'exécution sont identifiées comme temps d'exécution, temps d'exploitation du système et temps calendaire.

La topologie de la fiabilité concerne le nombre de fonctions dans le système qui peuvent tomber en panne avant le système. La topologie de la fiabilité est la relation d'une défaillance fonctionnelle individuelle avec la défaillance du système agrégé. Les fonctions logicielles sont généralement associées dans une topologie en série. La défaillance d'une fonction entraîne la défaillance du système logiciel. La conception des logiciels à tolérances des pannes peut être utilisée pour protéger un système en cas de défaillance d'une ou plusieurs fonctions.

c) Référence du temps d'exécution de la fonction logicielle

Le taux de défaillance logicielle peut être exprimé par rapport à trois périodes de référence différentes.

- Le *temps d'exécution* est le temps d'exécution de l'unité centrale, qui augmente lorsque le programme logiciel exécute des instructions. Le temps d'exécution est utilisé pour déterminer le taux de défaillance pendant le temps d'exécution du sous-système du logiciel d'application.
- Le *temps d'exploitation du système* est incrémenté lorsque le système matériel/logiciel dans son ensemble fonctionne. Il est utilisé pour déterminer le taux de défaillance pendant le temps de fonctionnement du sous-système logiciel à fonctionnement continu.
- Le *temps calendaire* est la période de temps utilisée à des fins de planification des projets. Le temps calendaire est toujours incrémenté.

Pendant l'exploitation du système, les programmes logiciels ne sont pas toujours exécutés en continu. Certains programmes peuvent partager le temps d'une seule unité centrale. Plusieurs unités centrales peuvent également être présentes, ce qui permet de chevaucher les exécutions de programmes. Il est nécessaire de combiner les taux de défaillance des divers programmes pour arriver à un taux de défaillance logicielle général. Ils sont convertis en une période de référence commune dans le temps d'exploitation du système. C'est la même période utilisée pour exprimer les taux de défaillances matérielles afin de faciliter la détermination du taux de défaillance du système matériel/logiciel combiné.

d) Criticité de la fonction logicielle

Les fonctions logicielles sont souvent utilisées pour le contrôle d'un système critique où une défaillance peut avoir des conséquences catastrophiques. Il convient d'identifier la criticité des fonctions logicielles de manière anticipée dans la définition du concept de système et de l'évaluer pendant la conception architecturale logicielle du système. Il convient de classer la criticité des pannes fonctionnelles dans les spécifications du système, par exemple critique, majeure ou mineure, en fonction de critères établis et de la vérifier par analyse dans la performance de la fiabilité du système.

Le niveau de risque associé à la fonction logicielle critique peut être déterminé et évalué au moyen de techniques d'évaluation des risques. Il convient que la gestion des risques de projet [22] mette l'accent sur la prévention et la tolérance des défauts lorsque la gravité des conséquences des défaillances peut être atténuée.

L'*analyse par arbre de pannes (AAP)* [18] peut être utilisée pour identifier les causes possibles d'un événement majeur non désiré. Elle est utilisée pour étudier les défauts

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

potentiels et leurs causes, et quantifier leur contribution à l'indisponibilité du système. L'analyse par arbre de pannes est une approche technique descendante où le point de départ est le programme du sous-système logiciel de niveau supérieur qui s'étend à l'unité logicielle la plus basse via la structure hiérarchique logicielle. Les défauts potentiels peuvent être identifiés et évalués individuellement en fonction de leur probabilité respective d'occurrence de défaillance. L'évaluation quantitative fournit une indication ou magnitude de la criticité de la fonction logicielle. Elle est intéressante pour l'optimisation de la conception et l'évitement des défauts.

L'*analyse des modes de défaillance et de leurs effets (AMDE)* [23] peut être utilisée pour déterminer les modes de défaillance possibles et les défauts dans les unités logicielles et leurs effets sur le sous-système de niveau supérieur suivant de la structure hiérarchique logicielle. L'analyse des modes de défaillance et de leurs effets est une approche technique ascendante. Elle peut être étendue et utilisée pour l'analyse de la criticité des fonctions logicielles. L'analyse de la criticité combine la valeur quantitative de la probabilité de défaillance et les informations qualitatives sur la gravité de la défaillance pour supporter l'amélioration de la conception et l'atténuation des défauts.

D'autres techniques d'analyse de la sûreté de fonctionnement du système [24] sont utilisées pour la décomposition du logiciel et la simulation du système. Elles peuvent être utilisées de manière sélective pour une évaluation détaillée de la fiabilité et de la maintenabilité des fonctions logicielles dans des systèmes matériels/logiciels combinés.

Le niveau d'intégrité logicielle est une valeur représentant les caractéristiques propres à un projet qui définit l'importance du logiciel pour l'utilisateur. Exemples de caractéristiques propres à un projet: complexité logicielle, criticité, risque, niveau de sûreté, niveau de sécurité, performance souhaitée et fiabilité. Le niveau d'intégrité logicielle est déterminé par la classification de la criticité de l'impact des conséquences des défaillances et de la fréquence associée des occurrences [25]. La criticité des fonctions logicielles est également spécifique à l'application. Pour les systèmes relatifs à la sûreté, il convient de définir et d'intégrer le niveau d'intégrité de la sûreté pour le développement du système logiciel afin de satisfaire aux exigences de sûreté fonctionnelles [26]. Pour les systèmes relatifs à la sécurité, il convient d'intégrer des exigences de sécurité du système spécifiques [27].

6.3.6 Vérification du logiciel et validation du système logiciel

La spécification du logiciel tend à être beaucoup plus complexe que celle des systèmes matériels physiques comme les machines et systèmes électriques/électroniques. L'"exactitude" du logiciel est une préoccupation majeure. Le processus de vérification [2] consiste à déterminer que les exigences relatives au logiciel sont complètes et correctes pour les étapes du cycle de vie du logiciel. Le processus de validation [7] consiste à déterminer que les performances du système et services sont conformes aux exigences du client/de l'utilisateur. Des systèmes d'habilitation appropriés, tels les équipements d'essai, des installations et ressources supplémentaires, sont nécessaires pour supporter la mise en œuvre des processus de vérification et de validation. Le système d'habilitation ne contribue pas directement aux fonctions de performance du logiciel ou du système soumis à essai pendant ses diverses étapes du cycle de vie.

a) Vérification du logiciel

Le processus de vérification du logiciel a pour but de confirmer que les exigences spécifiées sont remplies par le système logiciel. Les activités suivantes du processus de vérification sont recommandées:

- définir la stratégie pour la vérification du logiciel;
- développer un plan de vérification basé sur les exigences du système logiciel;
- identifier les contraintes et limites associées aux décisions de conception;
- garantir que le système d'habilitation pour la vérification est disponible et que les installations et ressources d'essai associées sont préparées;

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- effectuer la vérification pour prouver la conformité avec les exigences de conception spécifiées;
- documenter les résultats de vérification et les données;
- analyser les données de vérification pour l'initiation d'une action corrective.

b) Validation du système logiciel

Le processus de validation du système logiciel consiste à fournir une preuve objective que la performance du système satisfait aux exigences du client/de l'utilisateur. Les activités suivantes du processus de validation sont recommandées:

- définir la stratégie pour la validation des services dans l'environnement d'exploitation et atteindre la satisfaction des clients/utilisateurs;
- préparer un plan de validation;
- garantir que le système d'habilitation pour la validation est disponible et que les installations et ressources d'essai associées sont préparées;
- effectuer la validation pour montrer la conformité des services avec les exigences des clients/utilisateurs;
- documenter les résultats de validation et les données;
- analyser, enregistrer et consigner les données de validation conformément aux critères définis dans la stratégie de validation.

6.3.7 Essai des logiciels et mesure

a) Considération générale pour tester un logiciel

Le test d'un logiciel est le processus consistant à exécuter un programme ou un ensemble d'instructions codées dans le but de vérifier les fonctions logicielles et de dépister les erreurs. Les objectifs du test varient avec les besoins du projet, la disponibilité du produit logiciel, le statut de la maturité logicielle et la planification du test pendant le cycle de vie du logiciel. Lors de la planification du test d'un logiciel, il convient de considérer ce qui suit.

- La planification du test est primordiale et il convient de la documenter afin de décrire les objectifs du test, le processus, les procédures et les ressources associés au test.
- Le test d'un logiciel nécessite des connaissances, des compétences ainsi qu'une bonne pratique d'essai. Même si de nombreux programmes et outils de test ont été automatisés et largement déployés dans l'industrie, de bonnes techniques de test demandent les compétences, l'expérience, l'intuition et la créativité de la personne en charge des tests pour obtenir des résultats stables. Il est important de maintenir un dossier d'enregistrement de test pour garantir l'exactitude et la traçabilité des données de test.
- Soumettre à essai est bien plus que déboguer le programme logiciel pour localiser les défauts et corriger les erreurs. Les tests sont également utilisés dans la vérification et validation du logiciel, et la mesure de la disponibilité et de la fiabilité.
- L'efficacité des tests et des processus sont des critères pour les techniques de test basées sur la couverture. L'automatisation des tests peut accélérer le temps de test du logiciel et réduire le coût du projet. Il convient de considérer la sélection d'outils de test appropriés, les coûts de formation et de support associés à l'acquisition d'outils de test.
- Les tests peuvent ne pas être nécessairement les moyens les plus efficaces pour améliorer la qualité logicielle à moins d'appliquer des actions de suivi appropriées. Il convient de considérer les méthodes alternatives comme l'inspection de code et l'examen de code.
- Le test des logiciels fait uniquement partie du processus de croissance de la fiabilité et d'amélioration du logiciel. Il nécessite d'autres efforts d'assurance pour atteindre les objectifs de sûreté de fonctionnement.
- Un test complet peut ne pas être faisable ou pratique, et être souvent prohibitif en termes de temps et de coût. La complexité logicielle influence l'étendue de réalisation du test. Le

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

problème de complexité limite souvent la capacité de la personne en charge des tests à détecter et éliminer les bogues au moyen du processus de test.

- Les défauts logiciels latents existent dans le logiciel après son autorisation d'utilisation. La prévision de la fiabilité logicielle fournit un moyen d'estimer le temps de test requis pour réduire les bogues logiciels résiduels à un nombre acceptable avant l'édition de la version logicielle.
- Il convient d'effectuer le test à l'aide d'équipes d'essai séparées et indépendantes de celles développant les logiciels.

b) Types de tests logiciels

Les types de tests logiciels effectués pendant le cycle de vie du logiciel sont présentés ci-après.

- *Test unitaire*: test d'une unité logicielle qui peut être compilée avant d'être intégrée dans le programme logiciel ou sous-système. L'unité logicielle est testée pour vérifier que les spécifications de conception détaillées de l'unité ont été correctement mises en œuvre.
- *Test du sous-système*: test d'un programme logiciel de sous-système composé d'une ou plusieurs unités logicielles comme élément de configuration logiciel pour vérifier les exigences de performances fonctionnelles.
- *Test d'intégration*: test d'un système logiciel dans un hôte matériel en tant qu'ensemble composé de sous-systèmes intégrés pour vérifier le fonctionnement, exposer les problèmes dans les interfaces logicielles, interfaces matérielles et interactions entre le matériel et le logiciel, et valider la performance de la fiabilité.
- *Test de la croissance de la fiabilité*: test du logiciel dans un processus itératif pour améliorer la fiabilité au moyen de tests jusqu'à la défaillance, en analysant les défaillances, en mettant en œuvre des actions correctives sur la version logicielle existante pour la mise à niveau et en continuant l'essai avec la nouvelle version logicielle. La fin du test de croissance de la fiabilité est basée sur la réalisation de l'objectif de fiabilité logicielle établi.
- *Test de qualification*: test pour prouver que le logiciel satisfait à ses spécifications lorsqu'il est intégré dans son système matériel hôte et prêt à être utilisé dans son environnement cible. Avant la validation de la version finale pour la distribution du logiciel, les tests alpha et beta sont souvent réalisés à des fins d'assurance de la qualité. Le *test alpha* est un test interne effectué par un développeur logiciel avant publication pour les utilisateurs externes. Le *test beta* est un test pratique effectué par un nombre limité d'utilisateurs dans son application prévue afin d'obtenir un retour d'informations des utilisateurs.
- *Test de réception*: test d'un système logiciel pour confirmer que les exigences du client sont satisfaites. Concernant le test de réception de systèmes matériels/logiciels complexes où aucune information préalable n'existe sur des systèmes similaires, il convient de considérer l'essai de croissance de la fiabilité et de stress [28] en tant que partie des exigences de l'essai de réception.
- *Test de non-régression*: test du logiciel qui a été précédemment testé dans le but de détecter les erreurs de maintenance introduites, le nouveau code développé, la configuration incorrecte ou le contrôle de source inadéquat.

c) Testabilité du logiciel

La testabilité est la capacité du logiciel à subir des tests avec un minimum de temps et de ressources. La testabilité est une caractéristique de conception permettant de déterminer de façon efficace le statut opérationnel du logiciel. La caractéristique de conception orientée testabilité permet également de réaliser efficacement le processus de détection, d'isolation et de diagnostic des défauts. Il convient que la conception orientée testabilité soit axée sur une conception structurée de la fonction logicielle pour permettre le test. Une approche de conception modulaire, dans laquelle chaque fonction logicielle est indépendante des autres fonctions, faciliterait la testabilité dans le cadre de la détection et de l'isolation des défauts.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Cette approche améliorerait la maintenabilité de la fonction logicielle en simplifiant le processus de mise à jour ou de modification logicielle.

Des programmes d'auto-test, des procédures de surveillance et de contrôle peuvent être conçus et intégrés dans un système logiciel pour réaliser des tests automatiques du système. La fonction d'auto-test peut être activée sur demande ou automatiquement par le biais de programmes afin de faciliter la maintenance et le diagnostic du système en indiquant son statut de performance opérationnelle. Il convient que parmi les caractéristiques de conception des auto-tests figurent la capacité de détection des fausses alarmes ainsi qu'une indication telle qu'une erreur transitoire de l'opérateur. Une fausse alarme est un avertissement donné par la fonction de gestion du diagnostic d'auto-test indiquant l'existence d'un défaut opérationnel alors qu'aucun défaut n'existe. Les fausses alarmes peuvent être réduites ou éliminées par une analyse diagnostique exhaustive et précise et peuvent être validées par le processus de gestion du diagnostic pendant la durée de fonctionnement.

L'approche de conception structurée orientée testabilité implique un processus de rationalisation des objectifs d'essai. Le processus analyse les attributs logiciels et prévoit la probabilité de bogues dans le logiciel pouvant être révélés au moyen de tests. L'analyse permet d'optimiser le processus de test pour déterminer le nombre suffisant de tests. Elle offre un moyen de gérer les ressources d'essai et de déterminer la plus-value ou les bénéfices d'une approche spécifique de test.

d) Cas-tests

Les cas-tests sont développés sur la base des spécifications logicielles. Les cas-tests sont utilisés pour simuler les conditions d'exploitation réelles du logiciel dans lesquelles des zones d'intérêt spécifiques ou problèmes potentiels pourraient survenir. Un cas-test est un ensemble d'entrées d'essais, de conditions d'exécution et de résultats attendus développés pour un objectif de test particulier, par exemple emprunter un chemin de programme logiciel particulier ou vérifier la conformité avec une exigence spécifique. Une spécification de cas-test est la documentation permettant de spécifier les entrées, d'identifier les résultats de test prévus et d'établir les conditions d'exécution pour l'élément testé. Un processus de test efficace inclut des cas-tests générés manuellement et automatiquement. Les tests manuels couvrent l'étendue de détection des défauts logiciels reflétant la compréhension, par le développeur, du domaine et de la structure de données du problème. Les tests automatiques couvrent l'étendue des études de défaut en exécutant toute la plage de valeurs de test, y compris les extrêmes qui peuvent échapper aux personnes en charge des tests. Le processus de test automatique implique l'utilisation d'un générateur de cas-tests pour accepter le code source, les critères de test, les spécifications ou les définitions de structure de données comme entrées pour générer les données de test et déterminer les résultats attendus. Le test d'insertion de défauts peut être considéré comme l'un des cas-tests dans lequel un défaut est délibérément introduit dans une partie du système logiciel pour vérifier qu'une autre partie réagit de manière appropriée. Les résultats de test permettent de déterminer les conditions de défaut probables et de faciliter la conception de logiciels à tolérance de pannes. La technique d'insertion de défauts est également utilisée pour tester la couverture du programme de test en comptant la fraction de défauts insérés trouvés.

Tester un programme logiciel est une tentative de faire tomber en panne le logiciel. Il est important de noter que toute exécution ratée doit être associée à un cas-test pour être incluse dans la séquence de test du projet logiciel. L'aspect le plus important d'une stratégie de test est le nombre de défauts que le test a découvert en fonction du temps. Ceci fournit une indication de l'efficacité du test.

e) Mesure et métriques du logiciel pour la gestion de projet

La mesure est le processus de détermination ou d'estimation des valeurs quantitatives ou métriques permettant de faciliter une gestion de projet efficace. Les données métriques sont obtenues au moyen de diverses méthodes décrites comme suit à partir de différentes

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

perspectives pour la prédiction de la fiabilité logicielle et l'amélioration de la performance de sûreté de fonctionnement du système.

- *Métriques de structure de conception*: mesure de l'approche de conception, de la complexité et de l'indépendance de la conception de logiciels.
- *Métriques d'exécution de la conception*: mesure de l'étendue selon laquelle le logiciel effectue complètement les fonctions spécifiées.
- *Métriques de gestion de processus*: mesure de la gestion, de la rentabilité et de l'amélioration de la conception du logiciel sur la base des résultats d'analyse des métriques de données de processus et des métriques de données de défaut correspondantes, obtenues au cours du processus de collecte des données.
- *Métriques de gestion de produit*: mesure des caractéristiques du logiciel qui sont spécifiques au produit logiciel développé sur la base des résultats d'analyse des métriques de données de produit et des métriques de données de défaut correspondantes, obtenues au cours du processus de collecte des données.

Un grand nombre de ces métriques est utilisé comme entrées dans les paramètres des modèles de fiabilité logicielle pour la prédiction et l'estimation dans le cas où des valeurs quantitatives sont nécessaires.

L'Annexe G présente une synthèse des métriques de modèles de fiabilité logicielle généralement utilisées dans l'industrie.

6.3.8 Croissance et prévision de la fiabilité logicielle

La croissance de la fiabilité logicielle est la condition caractérisée par une amélioration progressive d'une mesure de performance de la fiabilité du système logiciel dans le temps. L'amélioration de la fiabilité logicielle est possible au moyen de la conception et l'amélioration progressive de la fiabilité est vérifiée au moyen de l'essai de croissance de la fiabilité. Le logiciel ne tombe pas en panne s'il n'est pas exécuté en étant exposé aux défaillances. Un programme logiciel peut uniquement tomber en panne lorsqu'il est exécuté. Les défaillances logicielles révèlent les défauts et l'élimination de ces défauts entraîne une amélioration de la fiabilité. Les tendances à la croissance de la fiabilité logicielle sont basées sur les taux d'élimination des défauts eu égard au temps d'exécution logiciel cumulé. À des fins de planification, le temps d'exécution peut être converti en temps calendaire pour établir les taux de défaillance logicielle en vue de l'estimation de la fiabilité. Un programme de croissance de la fiabilité [29] peut être établi pour le système matériel/logiciel combiné. Les modèles de croissance de la fiabilité et les méthodes d'estimation pour les évaluations, basés sur les données de défaillance obtenues via le programme de croissance de la fiabilité, sont décrits dans les méthodes statistiques pour la croissance de la fiabilité [30]. Les méthodes d'amélioration-types de la conception de logiciel figurent au 6.4. L'essai de croissance de la fiabilité logicielle est présenté comme suit.

a) Essai de croissance de la fiabilité logicielle

L'essai de croissance de la fiabilité est réalisé pour évaluer la fiabilité actuelle, identifier et éliminer les bogues et prévoir la fiabilité future. Les valeurs de fiabilité basées sur le nombre de bogues découverts et éliminés pendant le temps d'exécution sont comparées aux objectifs de fiabilité logicielle intermédiaires. Ceci a pour but de mesurer l'évolution de la fiabilité dans le processus d'essai pour atteindre les objectifs de fiabilité logicielle.

L'essai accéléré [31] a été utilisé avec succès pour réduire le temps d'essai du produit. Ceci a lieu via l'application de niveaux de stress accrus ou en augmentant la vitesse d'application des stress répétés pour évaluer ou montrer la croissance de la fiabilité du produit. Pour les systèmes matériels/logiciels combinés, l'application de stress sur le logiciel peut impliquer des excursions d'essai avec divers scénarios opérationnels. L'objectif est de vérifier l'adéquation de la performance du système dans des environnements d'exploitation simulés avec les temps d'essai et conditions pratiques.

b) Environnement d'exécution du logiciel

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

L'environnement d'exécution du logiciel inclut la plate-forme matérielle, telle le système d'hôte matériel, le logiciel du système d'exploitation, les paramètres de génération du système, la charge de travail et le profil opérationnel. Le profil opérationnel est décrit dans 6.3.3.

Une *exécution* est le résultat d'exécution d'un programme logiciel. Une exécution a des variables d'entrée et de sortie identifiables. L'essai de fiabilité logicielle est basé sur la sélection d'un ensemble de valeurs de variables d'entrée pour une exécution particulière. Chaque variable d'entrée a un type de données déclaré représentant une plage et classant les valeurs admissibles. Un profil opérationnel est une fonction associée à la probabilité de la variable d'entrée, qui est utilisée dans l'estimation statique pour la croissance de la fiabilité.

c) Copies logicielles multiples

Le temps d'essai pendant l'essai de croissance de la fiabilité peut être accumulé sur plusieurs copies du logiciel. Les copies peuvent être exécutées simultanément pour accélérer l'essai. Cette procédure permet l'accumulation d'exécutions de multiples copies pour accélérer le processus d'essai, en particulier pour montrer la réalisation des objectifs de fiabilité élevés. A cet égard, le temps calendaire total de l'essai est réduit. L'utilisation de multiples copies peut fournir des avantages en termes d'économie et de planification.

d) Prévision de la fiabilité logicielle

La croissance de fiabilité du logiciel est l'amélioration positive de la fiabilité logicielle dans le temps, accomplie via l'élimination systématique de bogues logiciels. La vitesse de la croissance de la fiabilité dépend de la rapidité à laquelle les bogues peuvent être découverts et éliminés. Un modèle de fiabilité logicielle applicable aux conditions de croissance permet la gestion de projet pour suivre l'évolution de la fiabilité logicielle via l'inférence statistique pour établir les tendances et prévoir les objectifs de fiabilité futurs. Des actions de gestion appropriées peuvent également être prises si la tendance s'avère négative.

e) Modèles de fiabilité logicielle

La mesure et la projection de la croissance de la fiabilité logicielle nécessitent l'utilisation d'un modèle de fiabilité logicielle approprié qui décrit la variation de la fiabilité logicielle dans le temps. Les paramètres du modèle de fiabilité peuvent être obtenus à partir des données historiques basées sur la prédiction, ou de l'estimation des données d'essai collectées pendant l'essai du système. Il convient de valider la sélection et l'utilisation du modèle de fiabilité logicielle. Le processus d'estimation est basé sur les moments auxquels les défaillances se produisent avec un échantillon de données suffisant pour l'accumulation significative du temps d'exécution. Ceci a pour but d'établir un degré raisonnable de confiance statistique pour valider les tendances de la croissance de la fiabilité. L'approche a pour but de prévoir les objectifs de maturité et d'édition de logiciels.

L'Annexe H présente des exemples-types de modèles de fiabilité logicielle utilisés dans la pratique industrielle.

6.3.9 Retour d'informations sur la sûreté de fonctionnement du logiciel

La collecte de données de la sûreté de fonctionnement du logiciel est traitée dans 6.2. L'activité de collecte de données est effectuée pour le suivi sur le terrain afin d'évaluer la sûreté de fonctionnement opérationnelle du logiciel dans les locaux du client. Ceci a pour but de garantir et confirmer que le niveau accepté de performance de sûreté de fonctionnement pendant le fonctionnement est supporté pour le déploiement logiciel. Les informations de sûreté de fonctionnement en service sont collectées avec les informations de retour client correspondantes. Ces informations permettent de justifier les changements pour de nouvelles exigences logicielles et d'initier le développement d'une nouvelle édition de logiciels.

Souvent en raison de la dynamique d'environnements d'application et de l'évolution technologique, les décisions concernant les nouvelles éditions de logiciels sont influencées par la concurrence du marché et motivées par les stratégies économiques.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

6.4 Amélioration de la sûreté de fonctionnement du logiciel

6.4.1 Présentation de l'amélioration de la sûreté de fonctionnement du logiciel

L'amélioration de la sûreté de fonctionnement du logiciel peut être effectuée avec l'amélioration de la conception de logiciels, l'amélioration via l'essai de croissance de fiabilité, et l'amélioration de la performance du support de maintenance logicielle pour les services d'assistance à la clientèle, y compris l'effort d'amélioration du logiciel.

La fiabilité est un attribut-clé de la sûreté de fonctionnement du logiciel. L'amélioration de la fiabilité logicielle au moyen de l'essai de croissance de la fiabilité est décrite dans 6.3.8. Les paragraphes suivants fournissent des approches pratiques concernant la conception de la fiabilité logicielle et les techniques recommandées pour l'amélioration et la mise en œuvre du logiciel. Les objectifs de conception mettent l'accent sur la *testabilité* pour faciliter la vérification des fonctions logicielles, sur la *modularité* pour l'indépendance de chaque fonction logicielle pour faciliter l'isolation et le confinement des défauts, et sur la *maintenabilité* pour faciliter la modification dans le cycle de vie du logiciel.

6.4.2 Simplification de la complexité logicielle

a) Complexité structurelle

La complexité structurelle décrit les chemins logiques pour la connexion du module logiciel dans le cadre de la conception de logiciels. Chaque unité de module peut être programmée (par codage) pour fournir une unité exécutable de fonction logicielle dans la structure logicielle. La complexité structurelle est associée à la testabilité des codes de programme, qui influe sur la détection des défauts, impactant ainsi la fiabilité et la maintenabilité de l'architecture logicielle. Plus la structure est complexe, plus il est difficile de tester le logiciel. Il convient que les règles de conception de logiciels établissent un niveau de complexité pour faciliter la conception pour la sûreté de fonctionnement.

b) Complexité fonctionnelle

La *complexité fonctionnelle* décrit les fonctions requises que le module logiciel ou le segment de code dans l'unité doivent exécuter. Dans l'idéal, il convient de concevoir une unité de module pour exécuter une fonction afin d'atteindre la simplicité avec un ensemble d'entrées et de sorties cohésives pour faciliter l'isolation et l'élimination des défauts logiciels. En pratique, il convient de considérer la complexité structurelle et la complexité fonctionnelle pour l'évaluation de la conception de logiciels. La stratégie de conception de logiciels en matière de complexité est directement associée au nombre de cas-tests nécessaires pour la vérification logicielle complète.

6.4.3 Tolérance aux pannes du logiciel

La tolérance aux pannes est la capacité du logiciel à poursuivre le fonctionnement et préserver l'intégrité des données en présence de certains défauts. La conception de la tolérance aux pannes du logiciel doit empêcher les défauts logiciels de provoquer une défaillance du système pendant son exploitation. La tolérance aux pannes du logiciel est conçue de manière à avoir une faible probabilité de présenter une défaillance en mode commun à partir d'un certain nombre de diverses conceptions du système, y compris les pratiques recommandées suivantes.

- *Confinement des défauts*: le logiciel est écrit de manière que lorsqu'un défaut se produit, il ne peut pas contaminer les sections du logiciel au-delà du domaine où il s'est produit.
- *Détection des défauts*: le logiciel est écrit de manière qu'il teste les défauts et y réponde lorsqu'ils se manifestent.
- *Récupération des défauts*: le logiciel est écrit de manière qu'après avoir détecté un défaut, les étapes suffisantes soient effectuées pour permettre au logiciel de continuer à fonctionner correctement.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- *Diversité de conception:* le logiciel et ses données sont créés de manière que des versions de rechange soient disponibles.

La tolérance aux pannes présente la propriété de dégradation progressive qui permet à un système logiciel de continuer à fonctionner correctement pendant une période de temps en cas de défaillances. Ceci a pour but d'empêcher les défaillances qui provoqueraient le cas échéant un arrêt soudain du système ou une rupture totale. La tolérance aux pannes a une importance particulière pour les systèmes critiques en matière de sécurité qui dépendent de la performance du système à haute disponibilité en présence de défauts ou fonctionnant dans des conditions défavorables. Un exemple de conception de tolérance aux pannes est le protocole TCP/IP. Il s'agit d'un protocole logiciel conçu pour permettre des communications bidirectionnelles fiables dans un réseau à commutation de paquets, même en présence de liaisons de communication imparfaites ou surchargées. La conception de la tolérance aux pannes prévoit que les points d'extrémité de la communication incluent la perte de paquets, la duplication, le reclassement et la corruption, de sorte que ces conditions n'altèrent pas l'intégrité des données, et réduisent uniquement le débit selon un montant proportionnel pour soutenir le fonctionnement.

La méthode de programmation multi-versions est une approche commune utilisée pour la tolérance aux pannes lors de la conception de systèmes critiques et pour l'amélioration de la fiabilité logicielle pendant le fonctionnement. La méthode engage de multiples programmes fonctionnellement équivalents qui sont générés indépendamment à partir des mêmes spécifications logicielles initiales. L'indépendance de l'effort de programmation séparé réduit fortement la probabilité de défauts logiciels identiques se produisant dans deux versions du programme ou plus. La mise en œuvre de ces programmes utilise différents algorithmes et langages de programmation. Des mécanismes spéciaux sont intégrés dans le logiciel pour permettre à ces programmes séparés d'être contrôlés par un système de vote dans l'algorithme de décision pour l'exécution du programme pendant l'application. Le concept est basé sur l'hypothèse selon laquelle il est plus probable que la sortie de multiples versions indépendantes soit correcte par rapport à la sortie d'une seule version d'un point de vue de la redondance. En pratique, les avantages de l'effort de multi-programmation en termes d'amélioration nécessitent la justification d'exigences de temps et de ressources supplémentaires pour garantir la mise en œuvre rentable. L'efficacité de cette méthode dépend également de la garantie de diverses caractéristiques de défaut entre les versions de conception et de mises en œuvre du logiciel.

6.4.4 Interopérabilité logicielle

L'interopérabilité logicielle est la capacité de divers systèmes logiciels à fonctionner ensemble pour échanger des informations et à utiliser les informations qui ont été échangées. Dans un système ouvert (par exemple, réseau IP), il est important d'atteindre l'interopérabilité de divers systèmes logiciels pour établir des liaisons de communication. La défaillance de la liaison de communication affecterait la sûreté de fonctionnement pendant la réalisation de la performance. Une approche pratique recommandée pour améliorer l'interopérabilité dans le réseau de communication consiste à intégrer une fonctionnalité spécifique dans la conception du système logiciel pour surveiller la situation de la communication établie. Par exemple, la terminologie de “battement de cœur” (système de traitement et de synchronisation des signaux) intégrant la fonction de surveillance doit envoyer le signal de “battement de cœur” à l'autre lorsque la liaison de communication est établie. Si la liaison est rompue ou interrompue en raison de changements dans l'environnement ou d'autres causes, le système logiciel tente automatiquement de maintenir la poursuite de la communication de manière à ne pas dégrader la sûreté de fonctionnement pendant l'opération de performance du réseau de communication.

6.4.5 Réutilisation du logiciel

La réutilisation du logiciel est motivée par diverses raisons, parmi lesquelles une expérience avérée au niveau de l'exploitation, une économie de temps et de coût, ainsi que l'utilisation de produits propriétaires dans les décisions commerciales.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

La réutilisation du logiciel est l'utilisation de logiciels existants pour former un nouveau logiciel. Un logiciel réutilisable est un bien réutilisable. Le bien réutilisable le plus connu est le code. Le code de programmation écrit à un moment peut être utilisé dans un autre programme écrit à un moment ultérieur. La réutilisation du code de programmation est une technique courante qui tente d'économiser du temps et des efforts en réduisant la quantité de travail répété. La *réutilisabilité* du logiciel est le degré selon lequel un bien logiciel peut être utilisé dans plusieurs systèmes logiciels ou pour la formation d'autres biens.

D'un point de vue de la sûreté de fonctionnement, il convient de contrôler l'application de la réutilisation du logiciel dans les projets et ses attributs de *réutilisabilité* pour arriver à l'amélioration de la sûreté de fonctionnement. La réutilisabilité dépend directement de la structure du logiciel et de la conception modulaire. Pour qu'une unité logicielle soit réutilisable, il convient de la confiner pour exécuter une seule fonction complètement. Cette restriction est essentielle car si la réutilisation prévue de l'unité logicielle exécute moins d'une fonction, ou si elle peut exécuter plusieurs fonctions, elle serait difficile à mettre en œuvre ou à maintenir pour son objet de réutilisation prévu. Tout écart par rapport à une telle restriction diminuerait l'utilité du logiciel réutilisable. La réutilisation du logiciel qui n'exécute pas exactement une fonction pourrait avoir un effet défavorable sur la sûreté de fonctionnement en raison de la possibilité d'erreurs introduites dans le logiciel pendant la mise en œuvre ou la maintenance.

Il convient de mettre en œuvre la réutilisation du logiciel uniquement si les exigences fonctionnelles de la nouvelle unité logicielle sont en accord avec celles du logiciel réutilisable pour une application et un environnement opérationnel très similaires. Sinon, cela diminue la rentabilité de l'objectif de réutilisation du logiciel, et réduit éventuellement la fiabilité lors de la mise en œuvre.

Il convient de bien documenter le logiciel réutilisable pour la traçabilité afin de faciliter la gestion de la configuration des biens logiciels. Il convient de traiter les produits et systèmes logiciels commerciaux (COTS) comme des logiciels réutilisables pour des applications multiples et variées. Il convient de mettre en œuvre l'essai de qualification pour valider la performance et l'adaptabilité du produit/système logiciel COTS pour satisfaire aux besoins d'application du projet.

6.4.6 Maintenance et amélioration du logiciel

La maintenance du logiciel est la modification d'un produit logiciel après livraison pour corriger les défauts, pour améliorer la performance ou autre attribut de performance logiciel, ou pour adapter le produit à un environnement modifié. Il existe quatre catégories principales de maintenance du logiciel.

- *Maintenance corrective*: modification réactive d'un produit logiciel effectuée après livraison pour corriger les problèmes détectés.
- *Maintenance adaptative*: modification d'un produit logiciel effectuée après livraison pour rendre un produit logiciel utilisable dans un environnement modifié ou changeant.
- *Maintenance pour l'amélioration*: modification d'un produit logiciel après livraison pour améliorer la performance ou la maintenabilité.
- *Maintenance préventive: modification d'un produit logiciel après livraison pour détecter et corriger les bogues dans le produit logiciel avant de permettre la* propagation dans les occurrences de défaillances réelles.

Les principaux problèmes de maintenance du logiciel sont d'ordre managérial et technique. Les problèmes de gestion incluent l'alignement avec les priorités du client, la planification et l'allocation de ressources de maintenance, la formation du personnel de maintenance, les travaux de maintenance sous-traités et les commentaires du client par le biais de l'enquête de satisfaction. Les problèmes techniques incluent le signalement des incidents, la résolution des problèmes techniques, l'analyse des impacts, la standardisation des procédures d'application et les pratiques d'essai, l'évaluation de la maintenabilité du logiciel et les mesures d'efficacité de l'essai.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

L'amélioration du logiciel fait partie du processus d'évolution du logiciel. Le système logiciel sur le terrain est noté pour sa complexité croissante en raison des travaux de modification et d'amélioration effectués pour satisfaire aux besoins du client, aux changements continus dans les stratégies de support de maintenance en raison d'offres de services compétitives, et de la nécessité de développer les connaissances et techniques pour s'adapter aux environnements commerciaux changeants. Il convient de vérifier, valider et documenter l'étendue et la réalisation de l'effort de maintenance et d'amélioration du logiciel. Il convient que les ressources spécifiques nécessaires pour la maintenance du logiciel fassent partie de la stratégie de garantie de la sûreté de fonctionnement.

6.4.7 Documentation relative au logiciel

La documentation relative au logiciel est le texte écrit et les informations des documents de conception et d'application associés au produit logiciel. La documentation est une partie importante du génie logiciel. Les principales catégories de documentations relatives au logiciel incluent ce qui suit.

a) Documentation relative à l'architecture et à la conception

La documentation présente une vue d'ensemble du produit logiciel et de ses relations avec l'environnement d'application et les principes de construction utilisés lors de la conception. Le document relatif à la conception du logiciel est un modèle de conception de logiciels complet fournissant des informations détaillées.

- La *conception de données* décrit les structures du logiciel. Les attributs et relations entre les objets de données dictent le choix des structures de données, qui ont un impact sur la complexité structurelle du logiciel, affectant la sûreté de fonctionnement pendant l'exploitation.
- La *conception architecturale* utilise les caractéristiques du flux d'informations et les schémas dans la structure du programme, ce qui a un impact sur la modularité de la conception de logiciels affectant la fiabilité de la conception du module logiciel. Il convient de mettre en œuvre la pratique recommandée pour la description architecturale [32].
- La *conception de l'interface* décrit les interfaces des programmes internes et externes ainsi que la conception d'interfaces humaines, y compris les interfaces matérielles et les pilotes matériels. Les conceptions des interfaces internes et externes sont basées sur les informations obtenues à partir de l'analyse de la conception, ce qui affecte les systèmes de redondance et les exigences de fiabilité des conceptions et configurations de systèmes, logiciels et matériels.
- La *conception procédurale* décrit les concepts de programmation structurée à l'aide de notations graphiques, tabulaires et textuelles. Ces moyens de conception permettent au concepteur de représenter les détails procéduraux qui facilitent la conversion en code qui affecte la consistance des pratiques de programmation et la réduction de l'introduction d'erreurs de codage.

b) Documentation technique

La documentation technique inclut le code, les algorithmes, les interfaces et le texte supplémentaire pour décrire les divers aspects des produits logiciels destinés au fonctionnement. Il convient que la documentation soit exhaustive mais concise dans l'écriture du code source pour faciliter la maintenance et la mise à jour logicielles. Le commentaire dans le code est une forme de documentation technique, où le commentaire inclut de brèves lignes de commentaires d'explication ajoutées au code qui sont reconnues par le compilateur comme étant des commentaires uniquement et ne faisant pas partie du code pour exécution. Le commentaire dans le code comme pratique de documentation logicielle a pour but d'accroître la réutilisabilité et la maintenabilité. Ceci facilitera l'examen et l'inspection du code, la mise à jour et la modification du code, et améliorera l'intégrité et la fiabilité logicielles. La documentation technique peut également inclure, lorsque cela est nécessaire et applicable au projet, des spécifications et des exigences, des plans et des procédures d'essai, des rapports techniques et des données appropriées.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

c) Documentation utilisateur

La documentation utilisateur [33] inclut les manuels pour les utilisateurs finals, les administrateurs système et le personnel de support. Elle est destinée à assister l'application des produits logiciels par l'utilisateur final. La documentation décrit comment le logiciel peut être utilisé dans son environnement d'application. La documentation utilisateur décrit également les fonctionnalités du produit logiciel et aide l'utilisateur à réaliser ces fonctionnalités pour l'application, y compris l'édition de logiciels et les informations de contrôle de version, les instructions de dépannage, les instructions de sécurité, les avertissements et restrictions concernant l'utilisation du logiciel, et les instructions d'aide. Parfois, l'aide en ligne est disponible pour promouvoir un accès convivial et le contact du service pour arriver à la satisfaction du client.

d) Documentation commerciale

La documentation commerciale inclut le matériel de promotion pour encourager les simples observateurs à en savoir plus sur le produit logiciel. L'accès Web et les centres d'assistance à la clientèle sont des services courants dans les environnements de marchés concurrentiels actuels pour obtenir les produits logiciels et les informations d'application. L'orientation vis-à-vis de la clientèle est essentielle pour développer une stratégie commerciale. La documentation commerciale est l'une des nombreuses approches pour la diffusion des informations. Elle peut inclure des informations traitant les valeurs spécifiques relatives à la sûreté de fonctionnement et les problèmes associés pour les applications logicielles appropriées, telles la réutilisation du logiciel et sa modification pour des applications spécifiques.

6.4.8 Outils automatisés

Les outils automatisés sont utiles pour le traitement des données périodiques, l'analyse informatique et la comparaison des résultats d'évaluation. Il existe de grandes gammes d'outils automatisés disponibles sur le marché pour satisfaire à la plupart des besoins d'application pour la modélisation, l'analyse, les données de base de connaissances pour supporter toutes les formes d'évaluation de la sûreté de fonctionnement. La sélection et l'application d'outils appropriés pour des tâches de projet spécifiques nécessitent la connaissance et l'expérience du spécialiste en sûreté de fonctionnement. Les outils automatisés sont des systèmes d'habilitation qui peuvent aider le travail informatique de routine à améliorer la productivité. Il convient d'étudier la validité et l'exactitude de ces outils automatisés avant tout engagement pour l'application du projet. Il convient de déterminer la supportabilité de ces outils avant l'acquisition d'outils. Les outils automatisés sont utilisés pour le développement et l'essai de logiciels applicables à l'amélioration de la sûreté de fonctionnement et de la croissance de la fiabilité. Ils représentent une partie essentielle du système d'habilitation utilisé dans la vérification du logiciel et la validation du système.

6.4.9 Support technique et formation des utilisateurs

Le support technique est une gamme de services d'assistance en matière d'utilisation de produits logiciels. L'objectif est d'aider l'utilisateur à résoudre des problèmes spécifiques liés à l'utilisation ou à l'application des produits. Le support technique revêt diverses formes, telles l'enquête téléphonique, le service en ligne, l'e-mail, la réparation d'accès à distance et la visite sur site pour la résolution des problèmes. Il existe une croissance et une utilisation accrues de centres d'appel externalisés par les organisations de développement de produits technologiques pour des raisons commerciales, économiques et géographiques afin de faciliter la réponse en temps réel aux services de support technique. Ces centres d'appel servent de support technique centralisé pour une vaste gamme de produits technologiques, tels les systèmes informatiques, y compris les logiciels nécessitant une assistance technique 24h/24 avec un accès utilisateur gratuit dans le monde. Les services de support technique font partie du support de maintenance pour soutenir l'opérabilité et la fiabilité des produits contribuant à l'amélioration de la sûreté de fonctionnement.

La formation des utilisateurs de logiciels est un aspect important de l'amélioration de la sûreté de fonctionnement du logiciel. L'objectif est d'améliorer ou de familiariser le niveau de

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

compétence ou la compréhension des applications logicielles du point de vue de l'utilisateur. La formation des utilisateurs de logiciels revêt diverses formes, y compris l'accès en ligne de la base de données didactique du fournisseur de produits, l'assistance du centre d'appel, le service d'un expert technique dédié pour résoudre les problèmes inhabituels rencontrés.

7 Assurance logicielle

7.1 Présentation de l'assurance logicielle

L'assurance logicielle est l'ensemble des activités planifiées et systématiques qui garantit que les processus du cycle de vie du logiciel et les produits sont conformes aux exigences, normes et procédures. Les modèles d'évolution des capacités (CMM) [8, 9] sont des outils de gestion courants recommandés pour la mise en œuvre de programmes d'assurance logicielle dans les organisations de développement de logiciels. Il existe également une méthodologie et des procédures détaillées et documentées en matière d'assurance logicielle pour le développement et les applications logiciels [34].

L'assurance logicielle implique généralement les disciplines techniques de qualité, fiabilité, sûreté et sécurité associées au développement de produits logiciels et à l'exploitation du système. Le processus d'assurance logicielle consiste à planifier, développer, maintenir et fournir des moyens de confiance et de prise de décision. Le cycle de vie de l'assurance [35] est dirigé à des fins d'évaluation de la conformité via le cycle de vie du système concernant les produits logiciels pour satisfaire à la sûreté, à la sécurité, à la sûreté de fonctionnement et autres objectifs applicables. Les études de cas d'assurance [36] sont des dossiers de réclamation concernant la performance du processus et les propriétés physiques et caractéristiques fonctionnelles du système logiciel audité pour la preuve de la conformité avec les spécifications du système. L'assurance logicielle intervient dans l'évaluation des risques, le test de vérification et de validation, la documentation et maintenance des dossiers d'audit comme preuve objective. L'assurance logicielle utilise les données de mesure pertinentes basées sur le projet pour surveiller le produit logiciel et le processus correspondant pour les améliorations possibles.

La sûreté de fonctionnement du logiciel met l'accent sur la fiabilité logicielle en tant que partie intrinsèque de l'assurance logicielle via la mise en œuvre du processus d'ingénierie de la fiabilité logicielle [37]. La sûreté de fonctionnement et la qualité du logiciel sont les conditions préalables pour garantir la sûreté et la sécurité pendant l'exploitation du système.

7.2 Processus de personnalisation

La personnalisation est une activité de gestion de projets pour garantir la synchronisation et l'action, ainsi que l'allocation appropriée de ressources pour satisfaire aux besoins du projet. Le processus de personnalisation peut être utilisé efficacement pour la mise en œuvre des activités d'assurance logicielle. La personnalisation est souvent utilisée dans les projets à court terme pour améliorer ou soutenir l'exploitation du système lorsque les exigences et contraintes du projet sont plus restrictives que pour le démarrage d'un nouveau projet de développement. Les tâches suivantes sont recommandées pour la mise en œuvre du processus de personnalisation.

- Identifier et documenter les circonstances qui influencent la personnalisation, telles l'environnement d'exploitation, la taille du projet et sa complexité, la planification du projet et son budget, la disponibilité des ressources, les problèmes de sûreté, de sécurité et d'intégrité, les problèmes d'héritage, et les exigences en matière de conformité aux normes.
- Identifier les exigences d'entrée pour la prise de décision.
- Définir les objectifs du projet et planifier le processus de personnalisation en vue de la mise en œuvre.
- Sélectionner les étapes du cycle de vie appropriées applicables à la personnalisation pour atteindre les résultats visés.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- Documenter les résultats de personnalisation pour faciliter l'examen d'efficacité et d'amélioration.

7.3 Influence technologique sur l'assurance logicielle

La technologie logicielle a conduit à de nombreuses avancées en matière de développement de logiciels efficace, entraînant une polyvalence et des avantages économiques pour les applications logicielles. Jusqu'à récemment, l'assurance logicielle mettait l'accent sur les améliorations de la qualité et de la fiabilité logicielles du point de vue du développement de produits. Les récentes cyber-attaques dans les opérations logicielles sont devenues plus fréquentes, plus importantes et de plus en plus sophistiquées. Elles affectent non seulement les développeurs de logiciels à l'aide de logiciels COTS mais provoquent également des problèmes importants de perte de temps pour les opérateurs du système logiciel et les utilisateurs finals. La situation dans son ensemble est devenue une réaction en chaîne propagée par des virus inconnus et des cyber-attaques furtives créant un environnement à risques complexe et dynamique pour les opérations informatiques qui dépendent des logiciels.

L'assurance logicielle est critique pour les organisations impliquées dans des transactions de sûreté, de sécurité et d'ordre financier eu égard à la vulnérabilité dans les applications logicielles. L'assurance logicielle inclut le développement et la mise en œuvre de méthodes et de processus pour garantir que les fonctions logicielles sont telles que prévues tout en réduisant les risques de vulnérabilités, codes malveillants, défauts ou erreurs qui pourraient porter atteinte à l'utilisateur final. L'assurance logicielle est vitale pour garantir la sécurité des ressources informatiques critiques. En raison du caractère très changeant de la menace, même un logiciel de qualité supérieure n'est pas invulnérable aux cyber-intrusions s'il est mal configuré et maintenu. La gestion des menaces dans le cyberespace nécessite une approche à plusieurs niveaux concernant la prévention et la collaboration en matière de sécurité. Les développeurs conçoivent des logiciels de plus en plus sécurisés et robustes, les intégrateurs systèmes garantissent que le logiciel est correctement installé, les opérateurs maintiennent le système correctement et les utilisateurs finals utilisent le logiciel de manière sûre et sécurisée.

Ceci conduit les organisations concernées par les logiciels à redéfinir l'assurance logicielle pour leurs opérations. Par exemple, *l'assurance logicielle* peut être interprétée comme le "niveau de confiance à partir duquel le logiciel est exempt de vulnérabilités, soit intentionnellement soit non intentionnellement conçu dans le logiciel ou inséré accidentellement à n'importe quel moment du cycle de vie du logiciel, et que le logiciel fonctionne de la manière prévue" [38]. Il convient que l'assurance logicielle fournisse un niveau raisonnable de confiance justifiable selon lequel le logiciel fonctionnera correctement et de manière prévisible et cohérente avec ses exigences documentées. L'objectif de l'assurance est de garantir que la fonction logicielle n'est pas compromise par une attaque directe ou un sabotage par le code implanté de façon malveillante.

Selon les applications spécifiques, le niveau de confiance dans l'assurance logicielle traite ce qui suit:

a) *fiabilité* – qu'aucune vulnérabilité exploitable n'existe, insérée de façon malveillante ou non intentionnelle;

b) *exécution prévisible* – que le logiciel fonctionne, lorsqu'il est exécuté comme prévu, permette une confiance justifiable;

c) *conformité* – que l'ensemble des activités multi-disciplinaires planifiées et systématiques pour la garantie des processus et produits logiciels est conforme aux exigences, normes et procédures.

Les défis identifiés pour l'assurance logicielle incluent:

1) les erreurs de conception accidentelles ou erreurs d'implémentation qui conduisent à des vulnérabilités de code exploitables;

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

2) un environnement technologique changeant qui expose à de nouvelles vulnérabilités et fournit aux cyber-attaquants de nouveaux outils d'exploitation;

3) des initiés et tiers malveillants qui cherchent à porter atteinte aux développeurs ou utilisateurs finals.

Le premier défi est représenté par une action accidentelle et non intentionnelle. Le deuxième et le troisième défi sont représentés par des actions intentionnelles et délibérées commises par les cyber-attaquants. La contremesure consiste à gérer les risques associés à ces défis au moyen de pratiques d'excellence en matière d'assurance logicielle.

7.4 Pratiques d'excellence en matière d'assurance logicielle

Il existe des forums de technologie logicielle et d'assurance logicielle [39] qui impliquent la participation du gouvernement, de l'industrie, des universitaires et des utilisateurs lors de la mise en œuvre de pratiques d'excellence en matière d'assurance logicielle. Les pratiques recommandées en matière de développement de logiciels sont identifiées au 6.1. Les pratiques d'excellence recommandées en matière d'assurance logicielle sont présentées comme suit:

a) établissement d'une politique d'assurance logicielle pour guider le développement du logiciel et la mise en œuvre du processus;

b) formation sur les applications technologiques associées au produit logiciel et sur l'utilisation de ressources de référence;

c) utilisation d'une plateforme de conception d'architecture logicielle commune pour faciliter le développement de divers produits logiciels;

d) mise en œuvre des processus du cycle de vie du logiciel;

e) initiation d'études de cas d'assurance logicielle pour l'évaluation des risques si garanti et approprié;

f) établissement de critères communs pour la vérification et la validation pour la qualification et conformité logicielle;

g) contrôle de gestion de la configuration de l'édition d'une version logicielle;

h) performance logicielle établie et système de suivi des défauts et de collecte des données pour l'amélioration de conception et du processus du logiciel;

i) centre d'assistance à la clientèle établi pour faciliter le support du service utilisateur et l'application du produit logiciel.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annexe A
(informative)

Classement des logiciels et applications logicielles

A.1 Classement des logiciels

A.1.1 Catégories de logiciels

Les catégories de logiciels incluent les produits et données de développement de logiciels qui sont produits par le processus de génie logiciel. Les caractéristiques du logiciel et les environnements d'application constituent des facteurs qui influencent les processus de sûreté de fonctionnement au niveau de la conception et de la mise en œuvre du logiciel. Le système de classement présente une combinaison ordonnée de vues et catégories associées au logiciel [40].

La catégorie est représentée par le groupement de logiciels en fonction d'attributs ou de caractéristiques. L'accent est mis sur les questions liées à la sûreté de fonctionnement du logiciel pour faciliter le développement et les applications. Les exemples-types de ces groupements incluent ce qui suit.

A.1.2 Caractéristiques

- *Mode de fonctionnement* – catégories définies par la technique de traitement spécifique ou le type adopté par le système logiciel, telle le traitement en temps réel, en lot, en temps partagé, parallèle et concurrent. Il convient qu'un système en temps réel mette l'accent sur le temps de réponse. Il convient que le logiciel en temps partagé mette l'accent sur les spécifications d'interface.
- *Echelle du logiciel* – catégories définies par la taille (par exemple, KSLOC) ou la complexité (par exemple, flux de données) du logiciel et interprétées comme petit, moyen ou grand, simple ou complexe. Il convient de décomposer un logiciel complexe ou grand ou de le ventiler en plus petites tailles pour faciliter le contrôle de projets, le test progressif ou l'intégration.
- *Stabilité* – catégories définies par l'aspect évolutionnaire intrinsèque ou la stabilité en termes de caractéristiques du système logiciel: changement continu, changement graduel ou improbabilité au changement. Les changements continus ou incrémentiels de logiciels nécessitent des spécifications d'interface qui permettent la flexibilité et la stabilité après chaque changement. Des modèles de développement spéciaux, tels les modèles en spirale et en chute d'eau, sont souvent utilisés.
- *Fonction logicielle* – catégories définies par le type de fonction: compilateur, traitement des transactions financières, traitement de texte, systèmes de contrôle... Il convient que les transactions financières mettent l'accent sur la sécurité et la disponibilité. Il convient que les systèmes de contrôle mettent l'accent sur la disponibilité, la sûreté et la sécurité.
- *Sécurité* – catégories définies par le niveau de protection contre les accès non autorisés, piste d'audit, programme et protection de données. Il convient que l'accent soit mis sur la robustesse et la disponibilité.
- *Fiabilité* – catégories définies par le niveau de fiabilité requis, tel la maturité, la tolérance des défauts et la récupérabilité. Il convient de mettre l'accent sur la croissance de la fiabilité et le contrôle de configuration pour, par exemple, la réalisation de la fiabilité.
- *Performance* – catégories définies par la performance logicielle en termes de capacité, rendement, rotation ou temps de réponse. Il convient de mettre l'accent sur le temps de réponse qui varie avec la charge et la capacité.
- *Langage* – catégories définies par le type de langage de programmation principalement utilisé pour le logiciel: traditionnel (par exemple COBOL, FORTRAN), procédural (par exemple C), fonctionnel (par exemple Lisp), orienté objet (ex.: C++). Il convient de mettre

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

l'accent sur la formation des programmateurs et la familiarité des utilisateurs avec les fonctions et limitations du langage de programmation.

A.1.3 Environnement

- *Champ d'application* – catégories définies par le type ou la classe de système externe dans lequel le logiciel est utilisé, par ex.: e-business, contrôle des processus et système de réseautage. La sécurité et l'intégrité des données sont les principaux facteurs d'influence pour le commerce électronique. La sécurité et la fiabilité sont des préoccupations importantes dans le contrôle des processus. Le temps de réponse et la disponibilité sont critiques pour l'exploitation des systèmes de réseau.
- *Système informatique* – catégories définies par le système informatique de destination spécifique dans lequel le logiciel fonctionne, comme par exemple les systèmes d'exploitation commandés par microprocesseur, d'ordinateur central et temps réel. Les limitations de taille de mémoire et de code de programmation sont importantes pour les systèmes de microprocesseur. Il convient de considérer la compatibilité de l'opérabilité logicielle dans la configuration matérielle de l'ordinateur central et le temps de réponse pour l'exploitation en temps réel.
- *Classe d'utilisateur* – catégories définies par le niveau de compétence ou les caractéristiques de sa classe d'utilisateur prévue: novice, intermédiaire ou expert. L'identification de la classe utilisateur est essentielle pour la conception de l'interface et le développement d'instructions d'utilisation pour contribuer à la facilité d'application.
- *Ressources informatiques* – catégories définies par les limitations des ressources informatiques, par exemple exigence de mémoire, espace disque et exigence de réseau local. Les limitations des ressources informatiques affectent le développement des besoins de support logiciel ainsi que la capacité d'application.
- *Criticité logicielle* – catégories définies par les exigences de niveau d'intégrité du produit, comme par exemple la sécurité nationale, la sécurité organisationnelle et la confidentialité. Il convient de considérer les exigences réglementaires et les besoins sociétaux.
- *Disponibilité du produit logiciel* – catégories définies par la disponibilité du produit logiciel, comme par exemple, un logiciel commercial (COTS), personnalisé ou propriétaire. La synchronisation de l'acquisition et de la disponibilité du produit logiciel est un facteur de décision pour la conception interne ou l'externalisation dans la gestion de projets.

A.1.4 Données

- *Représentation des données* – catégories définies par un élément, un type et une structure de données tels que fichier relationnel, indexé, formaté. Il convient de considérer la compatibilité des données.
- *Utilisation de données logicielles* – catégories définies par le type d'utilisation des données logicielles prévues, comme par exemple, un utilisateur unique, de multiples utilisateurs séquentiels. L'utilisation des données affecterait les critères de conception du fichier de données et de support de maintenance des données.

A.2 Applications logicielles

Le logiciel est utilisé dans une grande variété d'applications. En général, les applications logicielles informatiques peuvent être résumées comme suit.

- *Le logiciel de base* fournit l'infrastructure permettant le contrôle du matériel informatique pour que l'exécution du logiciel d'application puisse être faite. Exemples: systèmes d'exploitation comme Microsoft Windows, Mac OS et Linux.
- *Le logiciel d'application* est le logiciel informatique conçu pour aider l'utilisateur à exécuter une tâche particulière, par exemple les traitements de texte, feuilles de calcul et applications de base de données.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

- *Le micro-logiciel* est le logiciel résidant dans les dispositifs de mémoire programmables des produits utilisateurs finals pour le contrôle interne des divers appareils électroniques comme les télécommandes, calculatrices, téléphones mobiles et caméras numériques.
- *L'intergiciel* est le logiciel informatique qui connecte les éléments logiciels pour de multiples applications ou pour la fourniture de services, tels le multitraitement dans les systèmes distribués et les services Web.
- *Le logiciel de test* est un sous-ensemble de logiciels destiné spécialement au test du logiciel et à l'automatisation du test.
- *Le logiciel de programmation* est l'outil de développement de logiciels aidant les concepteurs de logiciels à créer, déboguer, maintenir ou supporter d'autres programmes et applications. Exemples: outils CASE.
- *Le maliciel* est le logiciel malveillant destiné à infiltrer un ordinateur sans l'autorisation du propriétaire.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annexe B
(informative)

Exigences relatives au système logiciel et activités de sûreté de fonctionnement associées

B.1 Généralités

Les exigences-types relatives au système logiciel et les activités de sûreté de fonctionnement associées sont résumées pour chaque étape du cycle de vie du logiciel. Les informations peuvent être utilisées comme référence pour la personnalisation des projets de sûreté de fonctionnement du logiciel.

B.2 Définition des exigences

Exigences relatives au système logiciel	*Activités de sûreté de fonctionnement associées*
• Informations commerciales sur les produits logiciels • Exigences relatives à l'application du système et aux besoins des utilisateurs • Domaine et plate-forme du système d'exploitation	• Identifier les exigences logicielles • Identifier les besoins de performance • Identifier les besoins de support

B.3 Analyse des exigences

Exigences relatives au système logiciel	*Activités de sûreté de fonctionnement associées*
• Exigences en matière de performance fonctionnelle et de capacité • Scénarios d'application • Exigences spécifiques à l'application relatives à la sûreté, à la sécurité et à l'intégrité le cas échéant • Exigences en matière d'interface • Exigences en matière de qualification • Faisabilité de la conception et de la testabilité des logiciels • Faisabilité de l'exploitation et de la maintenance • Exigences en matière d'installation et d'acceptation • Exigences en matière de documentation	• Développer un profil opérationnel • Développer un plan de sûreté de fonctionnement du projet • Développer un plan d'assurance de sûreté de fonctionnement • Identifier les métriques de sûreté de fonctionnement du logiciel • Déterminer les exigences d'intégrité du flux de données • Déterminer les exigences de sûreté et de sécurité • Etablir les règles de conception en matière d'ingénierie (ergonomie) des facteurs humains • Etablir les critères de support logiciel • Identifier les contraintes affectant la conception et la mise en œuvre de la sûreté de fonctionnement, y compris les exigences spécifiques à l'application pour l'intégration dans la conception • Etablir les critères de réutilisation logicielle • Etablir les critères de croissance de la fiabilité logicielle et d'acceptation de qualification • Déterminer les exigences en matière de documentation et d'enregistrements pour les tests de fiabilité

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

B.4 Conception architecturale

Exigences relatives au système logiciel	*Activités de sûreté de fonctionnement associées*
• Une architecture décrivant la structure de niveau supérieur et identifiant les éléments logiciels constitutifs • Transformation et allocation des exigences pour faciliter la configuration des éléments logiciels • Intégration d'exigences spécifiques à l'application relatives à la sûreté, à la sécurité et à l'intégrité si nécessaire dans l'architecture du système • Interfaces internes et externes pour l'intégration et la vérification du système • Documentation préliminaire pour les exigences de base de données et de test • Méthodes de conception recommandées et normes pour satisfaire aux objectifs du projet et spécifications de conception • Traçabilité des exigences de l'élément logiciel • Faisabilité de la conception détaillée • Conditions de fonctionnement et de maintenance	• Effectuer une analyse du scénario d'application • Déterminer la complexité structurelle et fonctionnelle du logiciel • Intégrer les exigences spécifiques à l'application dans les performances de sûreté de fonctionnement du système de modélisation • Effectuer l'analyse du modèle fonctionnel de disponibilité/fiabilité • Effectuer l'allocation de la disponibilité/fiabilité logicielle • Etablir la base de données des métriques de sûreté de fonctionnement • Effectuer une prévision de la disponibilité/fiabilité préliminaire • Etablir le plan de croissance de fiabilité et d'acceptation de qualification du logiciel • Etablir les dossiers de données et le système de notification • Etablir le plan de support logiciel • Etudier la conception architecturale pour la mise en œuvre

B.5 Conception détaillée

Exigences relatives au système logiciel	*Activités de sûreté de fonctionnement associées*
• Une structure de niveau inférieur améliorée pour le codage des unités logicielles en vue de l'inclusion dans les éléments de configuration logiciels • Spécifications de conception détaillée des unités logicielles et description des éléments de configuration logiciels • Cohérence et traçabilité des spécifications de conception détaillée et de conception architecturale • Etablissement de méthodes de conception et de normes pour satisfaire aux exigences du projet • Etablissement de méthodes de conception spéciales pour traiter des questions de sûreté, de sécurité et d'intégrité le cas échéant • Toutes les exigences d'interface pour la compilation et le test des unités logicielles et éléments de configuration • Documentation de la base de données et exigences de test détaillées et plannings de test • Examens de gestion de projet pour surveiller les objectifs d'avancement et de livraison • Référence pour la configuration logicielle et communications des changements de conception	• Mettre en œuvre les règles de conception de logiciels • Etablir les normes de mesure et critères d'évaluation de métriques • Intégrer les conceptions spécifiques pour satisfaire aux exigences de sûreté, de sécurité et d'intégrité • Promouvoir la conception tolérante aux pannes • Appliquer les normes de sûreté de fonctionnement du logiciel • Effectuer un examen et une inspection du code logiciel • Affiner l'allocation de la disponibilité/fiabilité logicielle • Effectuer l'évaluation de la complexité logicielle • Prévoir la fiabilité de l'unité logicielle • Prévoir la disponibilité/fiabilité du sous-système logiciel • Effectuer une analyse de l'optimisation de la conception • Affiner la prévision de la fiabilité logicielle • Mettre à jour la base de données des métriques de sûreté de fonctionnement • Mettre en œuvre la gestion de la configuration • Effectuer un examen formel de la conception • Effectuer un examen du projet

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

B.6 Réalisation

Exigences relatives au système logiciel	*Activités de sûreté de fonctionnement associées*
• Méthodes et normes de conception et de codage de l'unité logicielle • Elément de configuration logiciel avec unités logicielles spécifiques • Critères de vérification pour le test de l'unité • Couverture de test des unités logicielles • Vérification des fonctions logicielles, incluant les spécifications d'application pour les exigences de sûreté, de sécurité et d'intégrité • Faisabilité de l'intégration et de test du logiciel	• Mettre en œuvre les normes de mesure et critères d'évaluation de métriques • Déterminer la couverture de code des unités logicielles • Effectuer le test de l'unité • Déterminer la couverture des défauts et l'exhaustivité du test • Catégoriser les données de défaut pour la classification • Mettre en œuvre le processus d'assurance de la sûreté de fonctionnement pour le test de l'unité et le test fonctionnel • Vérifier que les unités et fonctions logicielles respectent les spécifications de performance et d'application • Etablir l'analyse de notification des défaillances et le système d'actions correctives • Mettre en œuvre un programme d'assurance logicielle incluant l'externalisation et la chaîne de distribution si nécessaire • Effectuer un examen du projet

B.7 Intégration

Exigences relatives au système logiciel	*Activités de sûreté de fonctionnement associées*
• Stratégie d'intégration pour unités logicielles et configuration • Critères de vérification pour l'essai d'élément de configuration logiciel • Vérification des sous-systèmes logiciels, incluant les spécifications d'application pour les exigences de sûreté, de sécurité et d'intégrité • Documentation des résultats d'essai d'intégration • Documentation des changements de conception • Stratégie de régression pour la revérification des éléments modifiés • Système de collecte de données d'essai	• Mettre en œuvre la procédure de suivi des défauts • Mettre en œuvre la procédure d'analyse des défauts • Initier le programme de croissance de la fiabilité • Mettre en œuvre la notification des défaillances, l'analyse et le système d'actions correctives • Mettre en œuvre le système de collecte de données • Vérifier les sous-systèmes logiciels pour l'intégration • Effectuer le test d'intégration • Evaluer les données du test de disponibilité/fiabilité • Identifier les points à problème • Effectuer les actions correctives • Contrôler le changement de conception et l'édition de version • Effectuer un examen du projet

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

B.8 Acceptation

Exigences relatives au système logiciel	*Activités de sûreté de fonctionnement associées*
• Critères pour l'acceptation du système logiciel • Mise en évidence de la conformité du test • Validation de résultats du test d'intégration en conformité avec les exigences • Validation du système logiciel pour l'acceptation des clients • Stratégie de régression pour le nouveau test prenant en compte l'intégration de modifications du logiciel • Documentation de résultats d'acceptation de qualification	• Effectuer le test de croissance de fiabilité et l'essai accéléré, si nécessaire • Contrôler la tendance de la fiabilité et le statut d'amélioration • Effectuer le test de qualification • Examiner les résultats du test pour acceptation • Initier l'acceptation des clients • Valider le système logiciel en conformité avec les exigences du client, incluant la mise en évidence de la performance de sûreté de fonctionnement et les caractéristiques de sûreté, de sécurité et d'intégrité le cas échéant • Documenter le statut d'édition de la version logicielle

B.9 Fonctionnement et maintenance

Exigences relatives au système logiciel	*Activités de sûreté de fonctionnement associées*
• Procédures et conditions de fonctionnement • Stratégie de support de maintenance • Support logistique • Collecte de données sur le terrain • Formation des utilisateurs • Programme d'assurance logicielle pour garantir la sûreté de fonctionnement de l'exploitation du système	• Surveiller les tendances de performance sur le terrain • Mettre à jour les dossiers de performance sur le terrain et de support de maintenance • Effectuer les enquêtes de satisfaction des clients • Examiner les données sur le terrain pour identifier les points d'amélioration de la fiabilité • Etablir le profil opérationnel de la performance sur le terrain • Maintenir l'historique de performance de la sûreté de fonctionnement du système et la base de données d'expérience • Collecter les métriques de la sûreté de fonctionnement appropriées pour la prévision de la fiabilité • Mettre en œuvre les pratiques d'excellence de l'assurance logicielle

B.10 Mise à jour/amélioration logicielle

Exigences relatives au système logiciel	*Activités de sûreté de fonctionnement associées*
• Mises à niveau logicielles • Mise en œuvre de la stratégie de maintenance d'amélioration • Introduction d'un nouveau service et évaluation des impacts • Effets de l'amélioration sur la performance logicielle	• Surveiller les mises à niveau logicielles • Effectuer la maintenance d'amélioration • Mettre en œuvre le changement de conception et le contrôle de la configuration • Evaluer l'impact de l'introduction d'un nouveau service • Gérer l'édition d'une nouvelle version logicielle

B.11 Résiliation

Exigences relatives au système logiciel	*Activités de sûreté de fonctionnement associées*
• Fin du service logiciel spécifique • Consultation utilisateur de la fin de l'ancien service et remplacement du nouveau service	• Identifier le logiciel retiré et la fin du service de support • Conseiller le service d'assistance à la clientèle des actions de sûreté de fonctionnement requises

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annexe C
(informative)

Processus d'intégration du modèle d'évolution des capacités

L'intégration du modèle d'évolution des capacités (CMMI) est un modèle d'évolution de l'amélioration des processus pour le développement de produits et services. Elle inclut les pratiques d'excellence qui traitent des activités de développement et de maintenance couvrant le cycle de vie du produit de la conception à la livraison et maintenance. Les modèles de *CMMI pour le développement* [9] comportent les pratiques qui couvrent la gestion de projets, la gestion de processus, l'ingénierie des systèmes, l'ingénierie matérielle, le génie logiciel et autre processus de support utilisé dans le développement et la maintenance. Le processus CMMI est mis en relation avec la mise en œuvre des exigences du système logiciel et des activités de sûreté de fonctionnement associées comme indiqué en Annexe B. Le processus CMMI est utilisé pour les activités d'étalonnage et d'évaluation, ainsi que pour guider les efforts d'amélioration d'une organisation. Le processus CMMI est conçu par niveau.

- Les niveaux de capacité, qui appartiennent à une représentation continue, s'appliquent à l'amélioration du processus d'une organisation dans les zones de traitement individuelles. Ces niveaux sont un moyen de guider le processus d'amélioration progressive correspondant à une zone de traitement donnée. Il existe six niveaux de capacité, numérotés de 0 à 5.
- Les niveaux de maturité, qui appartiennent à une représentation échelonnée, s'appliquent à l'amélioration du processus d'une organisation dans de multiples zones de traitement. Ces niveaux sont utilisés pour prévoir le résultat général du projet suivant. Il existe cinq niveaux de maturité, numérotés de 1 à 5.

Le Tableau C.1 compare les six niveaux de capacité avec les cinq niveaux de maturité.

Tableau C.1 – Comparaison des niveaux de capacité et de maturité

Niveau	Représentation continue des niveaux de capacité	Représentation échelonnée des niveaux de maturité
0	Un *processus incomplet* est un processus qui n'est pas exécuté, ou exécuté partiellement. Un ou plusieurs objectifs spécifiques de la zone de traitement ne sont pas satisfaits, et aucun objectif générique n'existe pour ce niveau car il n'y a aucune raison d'institutionnaliser un processus exécuté partiellement.	N/A
1	Un *processus exécuté* est un processus qui répond aux objectifs spécifiques de la zone de traitement. Il supporte et permet le travail nécessaire pour produire des produits. Bien que le niveau de capacité 1 entraîne des améliorations importantes, ces améliorations peuvent être perdues dans le temps si elles ne sont pas institutionnalisées. L'application de l'institutionnalisation aide à garantir que les améliorations sont maintenues.	Au niveau de maturité 1, les processus sont généralement ad hoc et chaotiques. L'organisation ne fournit généralement pas un environnement stable pour supporter les processus. Le succès de ces organisations dépend de la compétence des personnes dans l'organisation et pas de l'utilisation de processus éprouvés. Les résultats des organisations du niveau de maturité 1 donnent souvent des produits et services qui fonctionnent; mais, ils dépassent fréquemment leurs budgets et ne satisfont pas à leurs plannings. Il y a une tendance au surengagement, à l'abandon des processus en temps de crise et à l'incapacité de répéter les succès.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Niveau	Représentation continue des niveaux de capacité	Représentation échelonnée des niveaux de maturité
2	Un *processus géré* est un processus exécuté qui a l'infrastructure de base en place pour supporter le processus. Il est planifié et exécuté conformément à la politique; emploie des personnes qualifiées qui ont les ressources adéquates pour produire des rendements contrôlés; implique les acteurs correspondants; est surveillé, contrôlé et examiné; et est évalué en termes de cohérence avec sa description de processus. La discipline de processus reflétée par le niveau de capacité 2 aide à garantir que les pratiques existantes sont conservées pendant les périodes de stress.	Au niveau de maturité 2, les projets de l'organisation ont garanti que les processus sont planifiés et exécutés conformément à la politique; les projets emploient des personnes qualifiées qui ont les ressources adéquates pour produire des rendements contrôlés; impliquent les acteurs correspondants; sont surveillés, contrôlés et examinés; et sont évalués en termes de cohérence avec leur description de processus. Les résultats des organisations du niveau de maturité 2 garantissent que les pratiques existantes sont conservées pendant les périodes de stress; les projets sont exécutés et gérés conformément à leurs plans documentés; le statut des produits et la livraison des services sont visibles pour la direction à des points définis aux principales étapes-clés et lors de l'exécution des tâches principales. Des engagements sont établis parmi les acteurs correspondants et sont révisés, si nécessaire. Les produits sont contrôlés de manière appropriée. Les produits et services répondent à leurs descriptions de processus, normes et procédures spécifiées.
3	Un *processus défini* est un processus géré qui est personnalisé à partir de l'ensemble de processus normalisés de l'organisation conformément aux instructions de personnalisation de l'organisation, et contribue aux actifs associés aux processus organisationnels au moyen de produits, mesures et autres informations relatives à l'amélioration des processus. Les normes, descriptions de processus et procédures correspondantes sont cohérentes et personnalisées pour répondre à un projet particulier ou une unité organisationnelle donnée. Un processus défini indique clairement l'objet, les entrées, critères d'entrée, activités, rôles, mesures, étapes de vérification, sorties et critères de sortie. Les processus sont gérés de manière proactive par la compréhension des interrelations des activités de processus et des mesures détaillées du processus, de ses produits et services.	Au niveau de maturité 3, les processus de l'organisation sont bien caractérisés et compris, et sont décrits dans les normes, procédures, outils et méthodes. Ces processus normalisés sont utilisés pour établir leur cohérence en termes de mise en œuvre dans toute l'organisation. Les projets établissent leurs processus définis en personnalisant l'ensemble des processus normalisés de l'organisation conformément aux instructions de personnalisation. Les résultats des organisations du niveau de maturité 3 montrent la cohérence en termes de performance.
4	Un *processus géré quantitativement* est un processus défini qui est contrôlé à l'aide de techniques statiques et autres techniques quantitatives. Les objectifs quantitatifs de performance de qualité et de processus sont établis et utilisés comme critères de gestion du processus. La performance de qualité et de processus est comprise en termes statistiques et gérée au cours de la vie du processus.	Au niveau de maturité 4, l'organisation et les projets établissent des objectifs quantitatifs pour la performance de qualité et de processus et les utilisent comme critères de gestion des processus. Les objectifs quantitatifs sont basés sur les besoins du client, des utilisateurs finals, de l'organisation et des exécutants du processus. La performance de qualité et de processus est comprise en termes statistiques et est gérée au cours de la vie des processus. Pour les sous-processus sélectionnés, des mesures détaillées de performance de processus sont collectées et analysées statistiquement. Les mesures de performance de qualité et de processus sont intégrées dans le répertoire de mesure de l'organisation pour supporter la prise de décision basée sur les faits. Les causes spéciales de variation de processus sont identifiées et, si appropriées, les sources des causes spéciales sont corrigées pour empêcher les occurrences futures. Les résultats des organisations du niveau de maturité 4 montrent le contrôle adéquat de performance des processus à l'aide de techniques statistiques et autres techniques quantitatives pour garantir que les résultats de performance sont quantitativement prévisibles.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Niveau	Représentation continue des niveaux de capacité	Représentation échelonnée des niveaux de maturité
5	Un *processus d'optimisation* est un processus géré quantitativement qui est amélioré en fonction d'une compréhension des causes communes de variation inhérentes au processus. L'accent d'un processus d'optimisation est mis sur l'amélioration continue de la plage de performances de processus via des améliorations progressives et innovantes.	Au niveau de maturité 5, l'organisation améliore en continu ses processus en fonction d'une compréhension quantitative des causes communes de variation inhérentes aux processus; met l'accent sur l'amélioration continue de la performance de processus via le processus progressif et innovant et les améliorations technologiques. Les objectifs quantitatifs d'amélioration des processus pour l'organisation sont établis, révisés en continu pour refléter les objectifs commerciaux changeants et utilisés comme critères pour gérer l'amélioration des processus. Les effets des améliorations de processus déployées sont mesurés et évalués par rapport aux objectifs d'amélioration quantitatifs des processus. Les processus définis et l'ensemble de processus normalisés de l'organisation sont les objectifs des activités d'amélioration mesurables. Les résultats des organisations de niveau de maturité 5 montrent une amélioration continue des processus pour atteindre les objectifs quantitatifs établis en termes d'amélioration des processus.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annexe D
(informative)

Classification des attributs des défauts logiciels

D.1 Généralités

La *classification orthogonale des défauts* (ODC) est une méthode utilisée pour enregistrer et grouper les informations relatives aux défauts logiciels en termes d'attributs de défauts logiciels. Le processus ODC permet d'extraire les signaux de défauts et de déduire l'intégrité du processus de développement de logiciels. La classification se fonde sur ce que l'on sait du défaut. Lorsqu'un défaut ou une défaillance est révélé(e), la manière dont le défaut a été détecté et exposé et l'impact pour l'utilisateur sont normalement connus. Par conséquent, les attributs ODC d'*Activité*, *Déclencheur* et *Impact* peuvent être classés. De manière similaire, lorsqu'un défaut est diagnostiqué et résolu, les détails de la résolution sont connus. Les attributs ODC de *Destination du défaut*, *Type de défaut*, *Qualificatif*, *Source* et *Age* peuvent être classés. D'autres attributs non-ODC comme la phase de projet planifié trouvée, la gravité et le composant, qui sont enregistrés dans un système de suivi de défauts ou de défaillances, peuvent être utilisés en relation avec l'analyse basée sur ODC. L'ODC n'impose pas de structure spécifique. Les articles suivants résument la classification des attributs de défaut logiciel avec pour titres *section d'ouverture*, *section de fermeture* et *allocation de l'activité sur le déclencheur*.

D.2 Section d'ouverture

Une section d'ouverture permet de classer les attributs lorsqu'un défaut est détecté.

Lorsqu'un défaut est découvert et est ouvert pour le diagnostic pendant le développement de logiciels, les informations sur l'exposition aux défauts, l'impact probable et la gravité peuvent être évaluées. Les attributs-types de données de défaut enregistrées lorsqu'un défaut est observé peuvent être résumés dans le Tableau D.1.

Tableau D.1 – Classification des attributs de défauts logiciels lorsqu'un défaut est détecté

Activité d'élimination des défauts [a] **(si détecté)**	**Déclencheur** [b] **(moyen de détection)**	**Impact** [c] **(effet et gravité)**
• Examen de conception • Inspection de code • Essai de l'unité • Essai de fonction • Essai du système • Essai de réception • Essai de qualification • Essai de croissance de fiabilité	• Conformité de la conception • Compatibilité • Concurrence • Couverture • Séquençage • Interaction • Configuration	• Conditions d'installation • Aptitude à l'entretien • Intégrité/Sécurité/Sûreté • Fiabilité • Maintenance • Accessibilité • Aptitude à l'utilisation
NOTE 1 Les défauts logiciels sont souvent difficiles à répliquer. Il est important d'enregistrer les circonstances conduisant à et entourant l'incidence de la détection des défauts. NOTE 2 La traçabilité des exigences est nécessaire; soit traçable pour les exigences logicielles soit traçable pour un cas-test spécifique.		
[a] *Activité*: activité réelle qui a été exécutée au moment où le défaut a été détecté. [b] *Déclencheur*: environnement ou condition qui devait exister pour l'exposition au défaut. [c] *Impact*: pour les défauts de développement, l'impact est évalué sous forme d'effet potentiel et de gravité pour l'utilisateur; pour un défaut signalé sur le terrain, l'impact est la défaillance pour l'utilisateur.		

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

D.3 Section de fermeture

Une section de fermeture permet de classer les attributs lorsqu'un défaut est corrigé.

Si un défaut est éliminé après avoir été réparé, la nature exacte du défaut et l'étendue de la réparation sont connues. Les attributs-types de données de défaut enregistrées lorsqu'un défaut est corrigé peuvent être résumés dans le Tableau D.2.

Tableau D.2 – Classification des attributs de défauts logiciels lorsqu'un défaut est corrigé

Destination [a]	Type [b]	Qualificatif [c]	Age [d]	Source [e]
• Conception/Code	• Initiation • Contrôle • Fonction • Synchronisation • Interface	• Manquant • Incorrect • Externe	• Base • Nouvelle • Réécrit	• Développée en interne • Externalisée • Réutilisée à partir de la bibliothèque • Portée

a *Destination:* l'identité de niveau supérieur de l'entité qui a été corrigée.

b *Type:* la nature de la correction réelle qui a été effectuée.

c *Qualificatif: (*s'applique au défaut *Type)*: décrit l'élément de mise en œuvre inexistante, incorrecte ou inappropriée.

d *Age:* identifie l'historique de la *Destination*, par exemple *Conception/Code*, qui a rencontré le défaut.

e *Source:* identifie l'origine de la *Destination*, par exemple *Conception/Code*, qui a rencontré le défaut.

D.4 Allocation de l'activité sur le déclencheur

Les allocations d'activité sur le déclencheur groupent les déclencheurs applicables en fonction des activités d'examen de la conception de logiciel, d'inspection et de test. Les Tableaux D.3, D.4, D.5 et D.6 illustrent certains exemples génériques des allocations d'activité sur le déclencheur.

Tableau D.3 – Allocation de l'activité «examen de conception/inspection de code» sur les déclencheurs

Activité	Déclencheurs
Examen de conception/ inspection de code *Examen de la conception ou comparaison de la conception documentée par rapport aux exigences connues*	• *Conformité de la conception* L'examinateur de document ou l'inspecteur de code détecte le défaut en comparant l'élément de conception ou le segment de code inspecté avec sa spécification à l'étape précédente. Ceci inclut les documents de conception, le code, les pratiques et normes de développement, ou de garantir que les exigences de conception ne manquent pas ou ne sont pas ambigües. • *Logique/Flux* L'inspecteur utilise les connaissances des pratiques et normes de programmation de base pour examiner le flux de logique ou de données afin de garantir qu'elles sont correctes et complètes. • *Rétrocompatibilité* L'inspecteur utilise sa solide expérience du produit/composant pour identifier une incompatibilité entre la fonction décrite par le document de conception ou le code, et celle des versions précédentes du même produit ou composant. D'un point de vue de terrain, l'application du client, qui a été exécutée avec succès sur la version précédente, ne s'exécute pas sur la version actuelle. • *Compatibilité latérale* L'inspecteur avec une solide expérience détecte une incompatibilité entre la fonction décrite par le document de conception ou le code, et les autres systèmes, produits, services, composants ou modules avec lesquels elle doit être en interface. • *Concurrence* L'inspecteur considère la sérialisation comme nécessaire pour contrôler une ressource partagée lorsque le défaut est détecté. Ceci inclut la sérialisation de fonctions multiples, fils,

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Activité	Déclencheurs
	processus ou contextes ainsi que l'obtention et la libération de verrous. • *Document interne* Il y a des informations incorrectes, une incohérence ou une omission dans la documentation interne. Les prologues, commentaires de code et plans de test représentent certains exemples de documentation qui tomberaient dans cette catégorie. • *Dépendance du langage* Le développeur détecte le défaut en contrôlant les détails spécifiques au langage de la mise en œuvre d'un composant ou d'une fonction. Les normes linguistiques, problèmes de compilation et rendements spécifiques au langage sont des exemples de problèmes potentiels. • *Effets secondaires* L'inspecteur utilise sa solide expérience ou connaissance du produit pour prévoir certains comportements de système, produit, fonction ou composant pouvant résulter de la conception ou du code examiné. Les effets secondaires sont caractérisés comme le résultat d'un usage commun ou de configurations communes, mais en dehors de l'étendue du composant ou de la fonction avec laquelle la conception ou le code examiné est associé. • *Situation rare* L'inspecteur utilise sa solide expérience et connaissance du produit pour prévoir certains comportements du système qui ne sont pas considérés ou traités par la conception documentée ou le code examiné, et sont généralement associés à des configurations ou utilisations inhabituelles. Une reprise sur erreur manquante ou incomplète n'est en général pas classée avec un déclencheur de *situation rare*, mais relève très probablement de la *Conformité de la conception* en cas de détection pendant l'*examen/inspection*.

Tableau D.4 – Allocation de l'activité «Test de l'unité» sur les déclencheurs

Activité	Déclencheurs
Test de l'unité *Test structurel ou exécution basée sur les connaissances détaillées des éléments internes de code*	• *Chemin simple* Le cas-test a été motivé par la connaissance de branches spécifiques dans le code et pas par la connaissance externe de la fonctionnalité. Ce déclencheur n'est généralement pas sélectionné pour les défauts signalés sur le terrain à moins que le client n'ait de solides connaissances des éléments internes du code et de la conception et invoque spécialement un chemin spécifique (comme c'est parfois le cas lorsque le client est un partenaire commercial ou fournisseur). • *Chemin complexe* Lors de l'essai structurel, le cas-test a découvert que le défaut exécutait certaines combinaisons artificielles de chemins de code. La personne en charge des tests a essayé d'invoquer l'exécution de plusieurs branches dans plusieurs conditions différentes. Ce déclencheur est uniquement sélectionné pour les défauts signalés sur le terrain dans les mêmes circonstances que celles décrites dans *chemin simple.*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau D.5 – Allocation de l'activité «test de fonction» sur les déclencheurs

Activité	Déclencheurs
Test de fonction *Exécution fonctionnelle basée sur les spécifications externes de la fonctionnalité*	• *Couverture* Pendant le test fonctionnel, le cas-test qui a découvert le défaut était une tentative d'exercer le code pour une simple fonction, n'utilisant aucun paramètre ou un simple ensemble de paramètres • *Variation* Pendant le test fonctionnel, le cas-test qui a découvert le défaut était une tentative directe d'exercer le code pour une simple fonction, mais en utilisant une variété d'entrées et de paramètres. Ceci peut inclure les paramètres incorrects, les valeurs extrêmes, les conditions limites ou combinaisons de paramètres • *Séquençage* Pendant le test fonctionnel, le cas-test qui a détecté le défaut a exécuté plusieurs fonctions dans une séquence très spécifique. Ce déclencheur est uniquement sélectionné lorsque chaque fonction est exécutée avec succès de manière indépendante, mais échoue dans cette séquence spécifique. Une séquence différente peut également être exécutée avec succès • *Interaction* Pendant le test fonctionnel, le cas-test qui a détecté le défaut a initié une interaction parmi deux corps de code ou plus. Ce déclencheur est uniquement sélectionné lorsque chaque fonction est exécutée avec succès de manière indépendante, mais échoue dans cette combinaison spécifique. L'interaction implique plusieurs séquences en série simples des exécutions.

Tableau D.6 – Allocation de l'activité «test du système» sur les déclencheurs

Activité	Déclencheurs
Test du système *Test ou exécution du système complet, dans l'environnement réel, nécessitant toutes les ressources*	• *Charge de travail/stress* Le système fonctionne à ou quasiment à la limite de ressources, supérieure ou inférieure. Ces limites de ressources peuvent être créées au moyen d'une grande variété de mécanismes, y compris l'exécution de petites ou grandes charges, l'exécution de quelques ou nombreux produits en même temps, laissant fonctionner le système pendant une longue période. • *Récupération/exception* Le système est testé dans le but d'invoquer un système de traitement d'exceptions ou un certain type de code de récupération. Le défaut ne serait pas apparu si une exception antérieure n'avait pas provoqué l'invocation du traitement de l'exception ou de la récupération. D'un point de vue de terrain, ce déclencheur serait sélectionné si le défaut est dans la capacité du système ou du produit à récupérer d'une défaillance, pas la défaillance elle-même. • *Démarrage/redémarrage* Le système ou sous-système a été initialisé ou redémarré à la suite d'un arrêt précédent, d'une défaillance du système complet ou d'un sous-système. • *Configuration matérielle* Le système est testé pour garantir que les fonctions sont correctement exécutées dans des configurations matérielles spécifiques. • *Configuration logicielle* Le système est testé pour garantir que les fonctions sont correctement exécutées dans des configurations logicielles spécifiques. • *Test bloqué (mode normal)* Le produit fonctionne bien dans les limites de ressources et le défaut est apparu tout en essayant d'exécuter un scénario de test du système. Ce déclencheur est utilisé si les scénarios n'ont pas pu être exécutés parce qu'il existe des problèmes de base qui empêchent leur exécution. Ce déclencheur ne doit pas être utilisé dans les défauts signalés par le client.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annexe E
(informative)

Exemples de métriques de données logicielles obtenues à partir de la collecte de données

E.1 Métriques de données de défaut

Métrique	Application
Données signalant un problème • Date et heure de détection du défaut • Description du défaut détecté • Défaut détecté dans la zone de programme • Personne qui a détecté le défaut • Symptôme du défaut et statut • Gravité et priorité	Il convient d'utiliser les données collectées sur les projets logiciels pour signaler un problème concernant l'identification et l'occurrence d'un défaut.
Données d'actions correctives • Date de correction du défaut • Personne qui a corrigé le défaut • Action de maintenance effectuée • Description de la modification • Identification des modules modifiés • Informations de contrôle de version • Temps nécessaire pour corriger le défaut • Date de vérification du défaut corrigé • Personne qui a vérifié la correction	Il convient d'utiliser pour signaler la résolution du problème les données collectées sur les actions correctives et vérifiées lorsque le défaut a été corrigé
Défauts cumulés détectés	Il convient d'utiliser les données relatives aux défauts cumulés détectés pour déterminer le taux de défaillance et la tendance de la fiabilité sur une période
Défauts cumulés corrigés	Il convient d'utiliser les données sur les défauts cumulés corrigés pour déterminer les défauts connus qui nécessitent des actions correctives et le suivi de l'efficacité des actions de maintenance
Taux de détection des défauts	Le taux de détection des défauts est utilisé pour indiquer la tendance afin de faciliter la planification de la stratégie de maintenance et la gestion des ressources
Taux de correction des défauts	Le taux de correction des défauts est utilisé pour indiquer la tendance afin de faciliter la planification de la stratégie de maintenance et la gestion des ressources. Le réglage de la priorité pour l'action de maintenance est basé sur la gravité du défaut
Défauts par emplacement	Le suivi des défauts d'après les fonctions logicielles pour identifier la zone spécifique du code est plus exposé aux erreurs
Criticité des défauts	Classement du degré d'impact des défauts pour définir la priorité pour les actions de maintenance
Nombre et pourcentage de défauts graves	Indications pour planifier la stratégie de maintenance
Complexité structurelle par emplacement	Utilisation avec d'autres métriques pour déterminer l'impact des défauts générés en fonction de la complexité de la structure logicielle et de l'emplacement
Complexité fonctionnelle par emplacement	Utilisation avec d'autres métriques pour déterminer l'impact des défauts générés en fonction de la complexité des fonctions logicielles et de l'emplacement

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

E.2 Métriques des données de produit

Métrique	Application
Nombre et pourcentage de modules qui exécutent plusieurs fonctions	Indication de la cohésion de la conception logicielle générale sur la complexité fonctionnelle. Plus la complexité est élevée, plus la cohésion est faible et nécessite une reconception
Nombre et pourcentage de modules qui ont une complexité structurelle élevée	Indication de la conception générale du logiciel nécessitant une reconception pour réduire la complexité
Nombre et pourcentage de modules qui ont exactement une entrée et une sortie	Indication d'une conception cohésive qu'il convient d'utiliser comme base pour la pratique de conception structurée
Nombre et pourcentage de modules qui sont documentés d'après les normes	Indication de l'intégrité du code qu'il convient d'utiliser pour déterminer si le code comporte toutes es exigences et traite complètement les exigences.
Nombre et pourcentage de défauts qui figurent dans le code réutilisé	Indication de la non-fiabilité du code réutilisé

E.3 Métriques des données de processus

Métrique	Application
Défauts introduits par étape du cycle de vie	Indication du moment et de l'étape où les défauts ont été introduits et où des actions appropriées doivent être prises
Défauts détectés par étape du cycle de vie	Indication du moment et de l'étape où les défauts ont été détectés et justification du retard des actions correctives pour l'élimination du défaut
Temps total consacré à l'analyse	Indication du temps consacré à l'analyse pour l'identification et l'isolation du problème pour une action corrective, et les ressources associées requises
Temps total consacré à la conception	Indication du temps consacré à la conception du logiciel et ressources associées requises
Temps total consacré au codage	Indication du temps consacré au codage et à la programmation et ressources associées requises
Temps total consacré au test de l'unité	Indication du temps consacré au test de l'unité et ressources associées requises
Temps total consacré à l'essai du système	Indication du temps consacré à l'essai du système et ressources associées requises
Temps de maintenance total	Indication du temps consacré aux activités de maintenance et ressources associées requises
Temps moyen d'administration de la maintenance	Indication du temps consacré à l'administration de la maintenance et ressources associées requises. Les tâches administratives relatives à la maintenance incluent la période qui précède et celle qui suit la correction du défaut, par exemple le temps consacré à l'assignation du personnel de maintenance, à l'édition de la correction dans une nouvelle version
Temps moyen de l'action corrective	Indication du temps consacré aux actions correctives et ressources associées requises. Ceci reflète la rentabilité des activités de maintenance.
Raison de l'action corrective	Permet de déterminer la source des défauts. Raisons types: • action de maintenance précédente • nouvelle exigence • changement d'exigence • exigence mal interprétée • exigence manquante • exigence ambiguë • changement dans l'environnement logiciel • changement dans l'environnement matériel • erreur de code/erreur logique • erreur de performance

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Métrique	Application
Coût de l'action corrective	Indication du coût total de l'action corrective, y compris l'isolation du défaut, la résolution du problème et l'administration pour une action de maintenance efficace
Pourcentage de fonctions testées et vérifiées	Indication de la couverture, de l'efficacité et de l'intégrité du test
Pourcentage de chemins indépendants testés et vérifiés	Indication de la couverture et de l'intégrité du test structurel
Pourcentage de lignes sources de code testées et vérifiées	Indication de la couverture et de l'intégrité du test du code logiciel
Données historiques	Fourniture de l'historique de données concernant les problèmes de conception, processus et produit

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

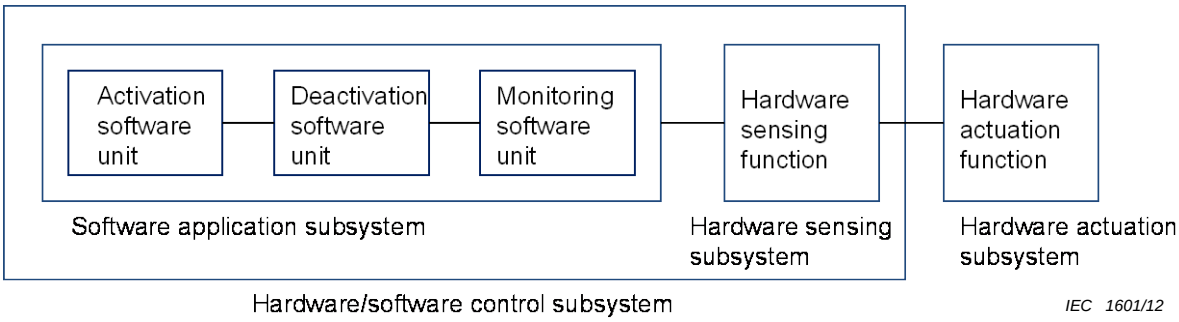
Annexe F
(informative)

Exemple de fonctions de fiabilité matérielle/logicielle combinées

Un système de contrôle de surveillance est illustré comme système matériel/logiciel combiné incluant les fonctions de commande et d'application électroniques représentées par la hiérarchie du système. Descriptions matérielles et logicielles du système de base:

- fonction de commande matérielle pour actionner le mécanisme de contrôle de surveillance;
- commande des composants du circuit électronique;
- fonction de détection matérielle pour démarrer ou arrêter l'activation;
- détection des composants du circuit électronique;
- fonctions d'application logicielles pour le contrôle de l'initiation rapide et de la validation de la fonction de détection matérielle, y compris
  - unité logicielle pour le contrôle de l'initiation de l'activation;
  - unité logicielle pour le contrôle de la validation et de la désactivation;
  - unité logicielle pour le contrôle de la vitesse d'activation et de désactivation.

Ces fonctions sont représentées au moyen d'un schéma de principe comme illustré dans la Figure F.1.



Légende

Anglais	Français
Activation software unit	Unité logicielle d'activation
Deactivation software unit	Unité logicielle de désactivation
Monitoring software unit	Unité logicielle de surveillance
Hardware sensing function	Fonction de détection matérielle
Hardware actuation function	Fonction de commande matérielle
Software application subsystem	Sous-système d'application logicielle
Hardware sensing subsystem	Sous-système de détection matérielle
Hardware actuation subsystem	Sous-système de commande matérielle
Hardware/software control subsystem	Sous-système de contrôle matériel/logiciel

Figure F.1 – Schéma de principe pour un système de contrôle de surveillance

La *fonction de commande matérielle* peut être considérée comme un sous-système matériel avec le circuit électronique de commande complet incluant les composants électroniques.

La *fonction de détection matérielle* peut être considérée comme un sous-système matériel avec le circuit électronique de détection complet incluant les composants électroniques.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Les *fonctions d'application logicielles* incluent trois unités logicielles séparées conçues pour fonctionner avec le sous-système de détection matérielle hôte. Chaque unité logicielle est conçue pour exécuter exactement une fonction comme prévu. Pour que le sous-système de détection matérielle fonctionne, toutes les unités logicielles doivent être intégrées dans le sous-système d'application logiciel pour fonctionner comme un sous-système matériel/logiciel combiné.

Pour l'analyse de la fiabilité, il convient de noter ce qui suit lors de la détermination des taux de défaillance du système et du sous-système. On part de l'hypothèse de taux de défaillance constants des composants électroniques qui incluent toutes les fonctions matérielles.

- Le taux de défaillance de la fonction de commande matérielle peut être déterminé par la somme des taux de défaillance des composants individuels du circuit de commande en supposant l'absence de redondance concernant la conception et le fonctionnement en continu à temps complet (λ_1).
- Le taux de défaillance de la fonction de détection matérielle peut être déterminé par la somme des taux de défaillance des composants individuels du circuit de détection en supposant l'absence de redondance concernant la conception et le fonctionnement en continu à temps complet (λ_2).
- La fiabilité des fonctions d'application logicielles est déterminée par la fiabilité combinée des trois unités logicielles en raison de leur dépendance associée avec le profil opérationnel du système de contrôle de surveillance. Le profil opérationnel ne nécessite pas le temps calendaire total pour l'exécution de ces unités logicielles, mais est uniquement nécessaire pour leurs applications à temps partiel sur demande. Il convient d'intégrer le temps d'exécution associé à chacune des unités logicielles dans le calcul de la densité de défaut relative à chaque unité logicielle respective. La densité de défaut est alors convertie en temps calendaire pour déterminer le taux de défaillance effectif. La densité de défaut est utilisée pour les unités réparables et le taux de défaillance pour les unités non réparables. Comme un logiciel est non réparable, le taux de défaillance s'applique ici. A cet égard, toutes les fonctions d'application logicielles sont traitées comme un élément de configuration logiciel individuel avec taux de défaillance (λ_3).
- La fiabilité du système de contrôle de surveillance en termes de taux de défaillance est déterminée par la somme de $\lambda_1 + \lambda_2 + \lambda_3 = \lambda$.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annexe G
(informative)

Résumé des métriques du modèle de fiabilité logicielle

Il existe de nombreux modèles de fiabilité logicielle utilisés dans la pratique industrielle. La plupart des modèles ont été développés pour des applications spécifiques. La plupart d'entre eux ont été informatisés et automatisés pour faciliter le traitement, l'analyse et l'évaluation des données. Il n'existe pas de modèle unique adapté à toutes les applications. Il incombe à l'utilisateur de sélectionner et d'appliquer des modèles de fiabilité logicielle appropriés pour satisfaire à ses propres besoins de projet. Ci-après figure une liste des métriques courantes utilisées dans la plupart des modèles de fiabilité logicielle.

a) Nombre total de défauts inhérents dans le logiciel. Cette métrique est supposée fixe et finie.

b) Nombre total de défauts latents (bogues) dans le logiciel. Cette métrique est supposée variable en raison de la possibilité d'insérer de nouveaux défauts dans le code dans le temps.

c) Nombre total de défauts corrigés à un moment donné, ou après écoulement d'un temps d'utilisation ou de test.

d) Nombre total de défauts détectés à un moment donné, ou après écoulement d'un temps d'utilisation ou de test.

e) Nombre de périodes ou intervalles de test. Il s'agit du nombre d'intervalles entre les activités de correction de défaut. Certains modèles supposent que les défauts sont corrigés dès qu'ils sont détectés.

f) Nombre total de défauts corrigés jusqu'à une période de test donnée.

g) Changement du taux de défaillance.

h) Temps de test ou d'utilisation accumulé jusqu'au moment présent ou nombre présent de défauts détectés.

i) Temps d'exécution accumulé.

j) Taux de défaillance initial.

k) Taux de défaillance présent.

l) Taux de croissance.

m) Nombre estimé de lignes de code exécutable au moment du test ou de l'utilisation.

n) Nombre total de cas-tests exécutés.

o) Nombre total de cas-tests exécutés avec succès.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Annexe H
(informative)

Sélection et application de modèles de fiabilité logicielle

Il existe de nombreux modèles et métriques disponibles actuellement pour estimer la fiabilité logicielle et mesurer les caractéristiques des logiciels. Tous les modèles de fiabilité sont développés pour des exercices de lissage à l'aide des données métriques collectées pour les entrées de modèle. La validité et la précision de l'application du modèle et la sortie qui en résulte dépendent des hypothèses émises dans la formulation du modèle et de la pertinence de l'entrée de données dans le modèle pour générer la sortie. La plupart des modèles sont développés pour satisfaire à un besoin spécifique pendant le cycle de vie du logiciel. Les exemples incluent le modèle de précision pendant la conception de logiciels et le modèle d'estimation pour déterminer le temps de test supplémentaire nécessaire avant l'édition du logiciel. Certains modèles sont développés pour prévoir la fiabilité du logiciel avant que le code ne soit écrit. L'entrée de données pour la fiabilité de la prévision dans ce cas est souvent basée sur les données historiques d'un système logiciel et d'une application similaires. D'autres modèles sont déployés pour estimer les tendances concernant la croissance de la fiabilité du logiciel à partir de l'entrée de données de test provisoires. Il n'existe pas de modèle unique capable de couvrir tout le cycle de vie du logiciel. En pratique, plusieurs modèles sont souvent essayés et utilisés pour déterminer la fiabilité du logiciel. Les techniques statistiques, par exemple essai d'ajustement pour vérifier comment un modèle ajuste un ensemble d'observations, sont souvent utilisées pour la sélection du modèle. La plupart des modèles de fiabilité logicielle sont automatisés en raison de leurs besoins informatiques itératifs. Les interprétations des résultats de modélisation de la fiabilité nécessitent une expérience pratique et une expertise en modélisation de la fiabilité.

Le Tableau H.1 présente des exemples-types de modèles de fiabilité logicielle utilisés dans la pratique industrielle. La présente norme n'a pas pour but de fournir une formulation de modèle détaillée et des applications paramétriques. Les références aux modèles de fiabilité logicielle et à leurs applications spécifiques sont bien documentées dans les ouvrages de référence [14, 37].

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Tableau H.1 – Exemples de modèles de fiabilité logicielle

	Nom du modèle	Hypothèses	Exigences relatives aux données	Limites d'application
1	Musa basic	• Nombre fini d'erreurs inhérentes (défauts latents) •Taux d'erreur constant dans le temps •Distribution exponentielle	• Nombre de défauts détectés à un moment donné • Estimation du taux de défaillance initial • Taux de défaillance actuel du système logiciel	• Le logiciel est opérationnel • Utilisation après intégration du système • Hypothèse qu'aucun nouveau défaut n'est introduit dans la correction • Hypothèse que le nombre de défauts résiduels diminue linéairement dans le temps
2	Musa-Okumoto	• Nombre infini d'erreurs inhérentes (défauts latents) •Taux d'erreur variable dans le temps •Distribution logarithmique	• Nombre de défauts détectés à un moment donné • Estimation du taux de défaillance initial • Changement relatif du taux de défaillance dans le temps • Taux de défaillance actuel du système logiciel	• Le logiciel est opérationnel • Utilisation pour les essais unité-système • Hypothèse qu'aucun nouveau défaut n'est introduit dans la correction • Hypothèse que le nombre de défauts résiduels diminue exponentiellement dans le temps
3	Jelinski-Moranda	• Nombre fini et constant d'erreurs inhérentes (défauts latents) •Taux d'erreur constant dans le temps •Erreurs corrigées dès leur détection •Distribution exponentielle binomiale	• Nombre de défauts corrigés à un moment donné • Estimation du taux de défaillance initial • Taux de défaillance actuel du système logiciel	• Le logiciel est opérationnel • Utilisation après intégration du système • Hypothèse qu'aucun nouveau défaut n'est introduit dans la correction • Hypothèse que le nombre de défauts résiduels diminue linéairement dans le temps
4	Littlewood-Verrall	• Incertitude dans le processus de correction	• Estimation du nombre de défaillances • Estimation du taux de croissance de la fiabilité • Temps entre défaillances détectées ou temps d'occurrence de la défaillance	• Le logiciel est opérationnel
5	Schneidewind	• Aucun nouveau défaut n'est introduit dans la correction	• Estimation du taux de défaillance au début du premier intervalle • Estimation de la constante de proportionnalité du taux de défaillance dans le temps • Défauts détectés à intervalle de temps égal	• Le logiciel est opérationnel • Le taux de détection des défauts diminue exponentiellement dans le temps

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

	Nom du modèle	Hypothèses	Exigences relatives aux données	Limites d'application
6	Géométrique	• Nombre inhérent de défauts considéré comme infini	• Baisse de la fonction de progression géométrique lorsque des défaillances sont détectées • Temps entre occurrences de défaillance ou temps d'occurrence de défaillance	• Le logiciel est opérationnel • Les défauts sont indépendants et inégaux en termes de probabilité d'occurrence et de gravité
7	Brooks-Motley	• Le taux de détection des défauts est constant dans le temps	• Effort de test de chaque test • Probabilité de détection des défauts dans le $i^{ème}$ test • Probabilité de correction des défauts sans en introduire de nouveaux • Nombre de défauts restant au début du $i^{ème}$ test Nombre total de défauts trouvés dans chaque test	• Logiciel développé de manière incrémentielle • Certains modules logiciels ont un effort de test différent des autres
8	Bayesian	• Le logiciel est relativement exempt de défauts	• Temps de test pour chaque intervalle • Nombre de défaillances détectées dans chaque intervalle	• Le logiciel est opérationnel • Le logiciel est corrigé à la fin de l'intervalle de test
9	Keene	• Met en corrélation le contenu du défaut latent fourni avec la capacité du processus de développement et la taille du logiciel (KSLOCs)	• Niveau de maturité CMM pour évaluer la capacité de processus, KSLOC estimé du code livrable, nombre estimé de mois pour atteindre la maturité après édition, latence des défauts, pourcentage de gravité 1 et 2 des défauts, temps de récupération, heures d'utilisation par semaine du code, et pourcentage d'activation des défauts	• Le pourcentage d'activation des défauts est un paramètre estimé qui représente le pourcentage moyen de sièges d'utilisateurs système qui sont susceptibles de rencontrer un défaut particulier; il convient d'évaluer le niveau de capacité CMMI pour l'organisation de développement ainsi que pour l'organisation de maintenance

Il convient d'utiliser les critères suivants pour faciliter la sélection du modèle:

- profils de défaillance;
- maturité du produit logiciel;
- caractéristiques du développement logiciel;
- caractéristiques de test du logiciel;
- métriques et données existantes.

Pour l'exécution du modèle, l'utilisation d'outils informatiques automatisés est recommandée. Il existe des outils disponibles dans le commerce qui couvrent totalement ou partiellement les modèles de fiabilité logicielle identifiés dans le Tableau H.1. Le principal avantage de l'utilisation d'outils informatiques automatisés réside dans les économies de temps et de coût de mise en œuvre pour les applications de modèles. En sélectionnant un outil approprié, les résultats d'un ensemble de données sur plusieurs modèles peuvent être comparés pour déterminer le meilleur ajustement.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Il convient de considérer les critères suivants lors de la sélection d'un ou plusieurs outils pour une organisation:

- disponibilité de l'outil compatible avec les systèmes informatiques de l'organisation;
- coût d'installation et de maintenance du programme;
- nombre d'études susceptibles d'être exécutées pour les applications d'outils;
- types de systèmes logiciels à étudier;
- qualité de la documentation des outils;
- facilité d'apprentissage de l'outil;
- flexibilité et puissance de l'outil;
- support technique de l'outil.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Bibliographie

[1] CEI 62508, *Lignes directrices relatives aux facteurs humains dans la sûreté de fonctionnement*

[2] ISO/CEI 12207, *Ingénierie des systèmes et du logiciel – Processus du cycle de vie du logiciel*

[3] CEI 60300-1, *Gestion de la sûreté de fonctionnement – Partie 1: Gestion du programme de sûreté de fonctionnement*

[4] CEI 60300-2, *Gestion de la sûreté de fonctionnement – Partie 2: Lignes directrices pour la gestion de la sûreté de fonctionnement*

[5] KLINE, M.B., *Software and hardware R&M: What are the differences?* Proceedings of the Annual Reliability and Maintainability Symposium, 1980

[6] LIPOW, M., SHOOMAN, M. L., *Software reliability, Tutorial Session*, Annual Reliability and Maintainability Symposium, 1986

[7] ISO/CEI 15288, *Ingénierie des systèmes et du logiciel – Processus de cycle de vie du système*

[8] *Capability Maturity Model® (CMM®)*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA USA

[9] *Capability Maturity Model Integration® (CMMI®) for Development, Version 1.2*; Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA USA 2006

[10] CEI 60300-3-3, *Gestion de la sûreté de fonctionnement – Partie 3-3: Guide d'application – Evaluation du coût du cycle de vie*

[11] CEI 62347, *Lignes directrices pour les spécifications de sûreté de fonctionnement des systèmes*

[12] CEI 61160, *Revue de conception*

[13] CHILLAREGE, Ram, *Orthogonal Defect Classification – A concept for in process Measurements*, IEEE Transactions on Software Engineering, 1992

[14] LYU, M.R. (Ed.): *The Handbook of Software Reliability Engineering*, IEEE Computer Society Press and McGraw-Hill Book Company, 1996

[15] IEEE-1633: *Recommended Practice on Software Reliability, 2009*

[16] ISO/CEI 20926, *Ingénierie du logiciel et des mesures — Mesurage du logiciel — Méthode IFPUG 2009 de mesurage de la taille fonctionnelle*

[17] CEI 61078, *Techniques d'analyse pour la sûreté de fonctionnement – Bloc-diagramme de fiabilité et méthodes booléennes*

[18] CEI 61025, *Analyse par arbre de panne (AAP)*

[19] CEI 61165, *Application des techniques de Markov*

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

[20] CEI 62551, *Techniques d'analyse de la sûreté de fonctionnement – Techniques des réseaux de Pétri*[2]

[21] DUGAN, J.B., *Fault Tree Analysis for Computer-based Systems, Tutorial Session*, Annual Reliability and Maintainability Symposium, 2000

[22] CEI 62198, *Gestion des risques liés à un projet – Lignes directrices pour l'application*

[23] CEI 60812, *Techniques d'analyses de la fiabilité du système – Procédure d'analyse des modes de défaillance et de leurs effets (AMDE)*

[24] CEI 60300-3-1, *Gestion de la sûreté de fonctionnement – Partie 3-1: Guide d'application – Techniques d'analyse de la sûreté de fonctionnement – Guide méthodologique*

[25] ISO/CEI 15026-3, *Ingénierie du logiciel et des systèmes – Assurance du logiciel et des systèmes – Partie 3: Niveaux d'intégrité du système*

[26] CEI 61508-3,*Sécurité fonctionnelle des systèmes électriques/électroniques/ électroniques programmables relatifs à la sécurité – Partie 3: Exigences concernant les logiciels*

[27] ISO/IEC 13335-1, *Information technology – Security techniques – Management of information and communications technology security – Part 1: Concepts and models for information and communications technology security management* (disponible en anglais seulement)
(retirée)

[28] CEI 62429, *Croissance de fiabilité – Essais de contraintes pour révéler les défaillances précoces d'un système complexe et unique*

[29] CEI 61014, *Programmes de croissance de fiabilité*

[30] CEI 61164, *Croissance de fiabilité – Tests et méthodes d'estimation statistiques*

[31] CEI 62506, *Méthodes d'essais accélérés de produits*[2]

[32] ISO/CEI/IEEE 42010, *Ingénierie des systèmes et des logiciels – Description de l'architecture*

[33] ISO/IEC 18019, *Ingénierie du logiciel et du système – Lignes directrices pour la conception et la préparation de la documentation de l'utilisateur de logiciels d'application*
(retirée)

[34] *Software Assurance Standard*, NASA-STD-8739.8 w/Change 1, May 2005

[35] ISO/CEI 15026-4, *Ingénierie du logiciel et des systèmes – Assurance du logiciel et des systèmes – Partie 4: Assurance du cycle de vie*[2]

[36] ISO/CEI 15026-2, *Ingénierie du logiciel et des systèmes – Assurance du logiciel et des systèmes – Partie 2: Cas d'assurance*

[37] LAKEY, P.B., NEUFELDER, A.M., *System and Software Reliability Assurance Notebook*, Rome Laboratory, 1996

2 A publier.

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

[38] *National Information Assurance (IA) Glossary*, (CNSS Instruction No. 4009), National Security Telecommunications and Information Systems Security Committee (NSTISSC), published by the United States federal government (unclassified), June 2006

[39] *Software Assurance: An Overview of Current Industry Best Practices*, Software Assurance Forum for Excellence in Code, February 2008

[40] ISO/CEI TR 12182, *Technologie de l'information – Classement des logiciels*

———————

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when print

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch

Copyrighted material licensed to BR Demo by Thomson Reuters (Scientific), Inc., subscriptions.techstreet.com, downloaded on Nov-27-2014 by James Madison. No further reproduction or distribution is permitted. Uncontrolled when printe