NORME INTERNATIONALE

INTERNATIONAL STANDARD

CEI
IEC
62279

Première édition
First edition
2002-09

Applications ferroviaires – Systèmes de signalisation, de télécommunication et de traitement – Logiciels pour systèmes de commande et de protection ferroviaire

Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems

Numéro de référence
Reference number
CEI/IEC 62279:2002

Numérotation des publications

Depuis le 1er janvier 1997, les publications de la CEI sont numérotées à partir de 60000. Ainsi, la CEI 34-1 devient la CEI 60034-1.

Editions consolidées

Les versions consolidées de certaines publications de la CEI incorporant les amendements sont disponibles. Par exemple, les numéros d'édition 1.0, 1.1 et 1.2 indiquent respectivement la publication de base, la publication de base incorporant l'amendement 1, et la publication de base incorporant les amendements 1 et 2.

Informations supplémentaires sur les publications de la CEI

Le contenu technique des publications de la CEI est constamment revu par la CEI afin qu'il reflète l'état actuel de la technique. Des renseignements relatifs à cette publication, y compris sa validité, sont disponibles dans le Catalogue des publications de la CEI (voir ci-dessous) en plus des nouvelles éditions, amendements et corrigenda. Des informations sur les sujets à l'étude et l'avancement des travaux entrepris par le comité d'études qui a élaboré cette publication, ainsi que la liste des publications parues, sont également disponibles par l'intermédiaire de:

- **Site web de la CEI (www.iec.ch)**
- **Catalogue des publications de la CEI**

  Le catalogue en ligne sur le site web de la CEI (http://www.iec.ch/searchpub/cur_fut.htm) vous permet de faire des recherches en utilisant de nombreux critères, comprenant des recherches textuelles, par comité d'études ou date de publication. Des informations en ligne sont également disponibles sur les nouvelles publications, les publications remplacées ou retirées, ainsi que sur les corrigenda.
- **IEC Just Published**

  Ce résumé des dernières publications parues (http://www.iec.ch/online_news/justpub/jp_entry.htm) est aussi disponible par courrier électronique. Veuillez prendre contact avec le Service client (voir ci-dessous) pour plus d'informations.
- **Service clients**

  Si vous avez des questions au sujet de cette publication ou avez besoin de renseignements supplémentaires, prenez contact avec le Service clients:

  Email: custserv@iec.ch
  Tél: +41 22 919 02 11
  Fax: +41 22 919 03 00

Publication numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series. For example, IEC 34-1 is now referred to as IEC 60034-1.

Consolidated editions

The IEC is now publishing consolidated versions of its publications. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Further information on IEC publications

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology. Information relating to this publication, including its validity, is available in the IEC Catalogue of publications (see below) in addition to new editions, amendments and corrigenda. Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is also available from the following:

- **IEC Web Site (www.iec.ch)**
- **Catalogue of IEC publications**

  The on-line catalogue on the IEC web site (http://www.iec.ch/searchpub/cur_fut.htm) enables you to search by a variety of criteria including text searches, technical committees and date of publication. On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.
- **IEC Just Published**

  This summary of recently issued publications (http://www.iec.ch/online_news/justpub/jp_entry.htm) is also available by email. Please contact the Customer Service Centre (see below) for further information.
- **Customer Service Centre**

  If you have any questions regarding this publication or need further assistance, please contact the Customer Service Centre:

  Email: custserv@iec.ch
  Tel: +41 22 919 02 11
  Fax: +41 22 919 03 00

.

NORME INTERNATIONALE

INTERNATIONAL STANDARD

CEI
IEC
62279

Première édition
First edition
2002-09

Applications ferroviaires – Systèmes de signalisation, de télécommunication et de traitement – Logiciels pour systèmes de commande et de protection ferroviaire

Railway applications – Communications, signalling and processing systems – Software for railway control and protection systems

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

CODE PRIX
PRICE CODE XE

Pour prix, voir catalogue en vigueur
For price, see current catalogue

SOMMAIRE

CONTENTS

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

APPLICATIONS FERROVIAIRES – SYSTÈMES DE SIGNALISATION, DE TÉLÉCOMMUNICATION ET DE TRAITEMENT – LOGICIELS POUR SYSTÈMES DE COMMANDE ET DE PROTECTION FERROVIAIRE

AVANT-PROPOS

1) La CEI (Commission Électrotechnique Internationale) 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. 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 intéressés sont représentés dans chaque comité d'études.
3) Les documents produits se présentent sous la forme de recommandations internationales. Ils sont publiés comme normes, spécifications techniques, rapports techniques ou guides et agréés comme tels par les Comités nationaux.
4) Dans le but d'encourager l'unification internationale, les Comités nationaux de la CEI s'engagent à appliquer de façon transparente, dans toute la mesure possible, les Normes internationales de la CEI dans leurs normes nationales et régionales. Toute divergence entre la norme de la CEI et la norme nationale ou régionale correspondante doit être indiquée en termes clairs dans cette dernière.
5) La CEI n'a fixé aucune procédure concernant le marquage comme indication d'approbation et sa responsabilité n'est pas engagée quand un matériel est déclaré conforme à l'une de ses normes.
6) L'attention est attirée sur le fait que certains des éléments de la présente Norme internationale peuvent faire l'objet de droits de propriété intellectuelle ou de droits analogues. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de propriété et de ne pas avoir signalé leur existence.

La Norme internationale CEI 62279 a été établie par le comité d'études 9 de la CEI: Matériel électrique ferroviaire.

La présente norme, basée sur la norme européenne EN 50128 (2001), a été préparée par le sous-comité 9XA: Systèmes de signalisation de télécommunications et de traitement, du Comité Technique 9X du CENELEC: Applications électriques et électroniques dans le domaine ferroviaire. Elle a été soumise aux Comités Nationaux pour vote suivant la procédure par voie express, par les documents suivants:

FDIS	Rapport de vote
9/687/FDIS	9/704/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 norme doit être lue conjointement avec la CEI 62278 et la norme ENV 50129:1998.

La présente norme ne suit pas les règles de structure des normes internationales comme le spécifie la Partie 2 des Directives ISO/CEI.

NOTE Cette norme a été reproduite sans modifications importantes de son contenu original ou de ses règles structurelles.

INTERNATIONAL ELECTROTECHNICAL COMMISSION

RAILWAY APPLICATIONS – COMMUNICATIONS, SIGNALLING AND PROCESSING SYSTEMS – SOFTWARE FOR RAILWAY CONTROL AND PROTECTION SYSTEMS

FOREWORD

1) The IEC (International Electrotechnical Commission) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of the 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, the IEC publishes International Standards. 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. The 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 the 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 National Committees.
3) The documents produced have the form of recommendations for international use and are published in the form of standards, technical specifications, technical reports or guides and they are accepted by the National Committees in that sense.
4) In order to promote international unification, IEC National Committees undertake to apply IEC International Standards transparently to the maximum extent possible in their national and regional standards. Any divergence between the IEC Standard and the corresponding national or regional standard shall be clearly indicated in the latter.
5) The IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with one of its standards.
6) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. The IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 62279 has been prepared by IEC technical committee 9: Electric railway equipment.

This standard, based on the European Norm EN 50128 (2001), was prepared by subcommittee 9XA: Communication, signalling and processing systems, of Technical Committee CENELEC TC 9X: Electrical and electronic applications for railways. It was submitted to the National Committees for voting under the Fast Track Procedure as the following documents:

FDIS	Report on voting
9/687/FDIS	9/704/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 standard shall be read in conjunction with IEC 62278 and ENV 50129:1998.

This standard does not follow the rules for structuring International Standards as given in Part 2 of the ISO/IEC Directives.

NOTE This standard has been reproduced without significant modification to its original content or drafting.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant 2008. A cette date, la publication sera

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

The committee has decided that the contents of this publication will remain unchanged until 2008. At this date, the publication will be

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

INTRODUCTION

La présente Norme internationale fait partie intégrante d'un groupe de normes connexes. Les autres éléments de ce groupe sont la Norme internationale CEI 62278 et la norme européenne ENV 50129. La CEI 62278 traite des problèmes liés aux systèmes de façon très générale, tandis que la norme ENV 50129 aborde le processus d'approbation des systèmes individuels qui peuvent exister dans le cadre du système global de commande et de protection ferroviaire. La présente norme traite en particulier des méthodes qu'il est nécessaire d'utiliser pour fournir des logiciels répondant aux exigences d'intégrité de la sécurité imposées par ces considérations plus larges.

L'orientation de la présente norme doit beaucoup au travail préalable effectué par le groupe de travail (GT) 9 du comité d'études 65 de la CEI. Le travail du GT 9 a abouti à une norme générique, destinée aux logiciels des systèmes de sécurité, qui est désormais intégrée dans la série de normes CEI 61508*. Un aspect spécifique du travail effectué par le GT 9 est l'inclusion d'une intégrité logicielle de niveau 0, applicable aux logiciels non liés à la sécurité, ainsi que d'une intégrité logicielle de niveaux 1 à 4, applicable aux logiciels liés à la sécurité et critiques pour celle-ci. La présente norme couvre également les cinq niveaux d'intégrité logicielle.

Le travail de l'IRSE (Institution of Railway Signal Engineers) a également été pris en compte, notamment son rapport technique numéro 1 qui traite du même sujet.

Le concept-clé de la présente norme est celui des niveaux d'intégrité de la sécurité logicielle. Plus les conséquences d'une défaillance logicielle sont dangereuses, plus le niveau d'intégrité de la sécurité logicielle est élevé.

La présente norme a identifié des techniques et mesures applicables aux cinq niveaux d'intégrité de la sécurité logicielle dans lesquels 0 correspond au niveau minimum et 4 au niveau le plus élevé. Les niveaux 1 à 4 font référence aux logiciels liés à la sécurité, alors que le niveau 0 s'applique aux logiciels non liés à la sécurité. Ce niveau a été inclus dans la norme de manière normative, de façon à permettre une transition progressive entre les développements du logiciel des systèmes non liés à la sécurité et ceux des systèmes liés à la sécurité. Les techniques et mesures requises pour chaque niveau d'intégrité de la sécurité logicielle et pour le niveau non lié à la sécurité sont indiquées dans les tableaux. Dans la présente version, les techniques requises pour le niveau 1 sont identiques à celles du niveau 2, et les techniques requises pour le niveau 3 sont identiques à celles du niveau 4. La présente norme ne fournit aucune ligne directrice sur le niveau d'intégrité logicielle approprié pour un risque donné. Cette décision sera tributaire de nombreux facteurs, notamment de la nature de l'application, de la limite dans laquelle les autres systèmes assurent des fonctions de sécurité, ainsi que de facteurs socio-économiques.

C'est la fonction de la CEI 62278 et de la norme ENV 50129 de spécifier les fonctions de sécurité affectées au logiciel.

La présente norme spécifie les mesures nécessaires au respect de ces exigences. Le processus est illustré par la Figure 1.

La CEI 62278 et la norme ENV 50129 exigent qu'une approche systématique soit adoptée pour ce qui concerne

i) l'identification des dangers, des risques et des critères de sécurité;

ii) l'identification de la réduction des risques nécessaire pour répondre aux critères de sécurité;

iii) la définition d'une Spécification des Exigences de Sécurité du Système, globale, qui décrit les protections indispensables en vue d'atteindre la réduction des risques requise;

---

* CEI 61508 (toutes les parties), *Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité*.

INTRODUCTION

This International Standard is part of a group of related standards. The others are the International Standard IEC 62278 and the European Norm ENV 50129. IEC 62278 addresses system issues on the widest scale, while ENV 50129 addresses the approval process for individual systems which may exist within the overall railway control and protection system. This standard concentrates on the methods which need to be used in order to provide software which meets the demands for safety integrity which are placed upon it by these wider considerations.

This standard owes much of its direction to earlier work done by Working Group (WG) 9 of IEC/TC 65. The work of WG 9 resulted in a generic standard for software for safety systems which is now part of standards of IEC 61508 series*. A particular aspect of the work by WG 9 is its inclusion of Software Integrity Level 0, which covers non-safety software, as well as Software Integrity Levels 1 to 4, which cover safety-related and safety-critical software. This standard also covers all five Software Integrity Levels.

Account has also been taken of the work of the Institution of Railway Signal Engineers (the IRSE), in particular its Technical Report Number 1, which addressed the same topic.

The key concept of this standard is that of levels of software safety integrity. The more dangerous the consequences of a software failure, the higher the software safety integrity level will be.

This standard has identified techniques and measures for 5 levels of software safety integrity where 0 is the minimum level and 4 the highest level. Four of these levels, 1 to 4, refer to safety-related software, whilst level 0 refers to non-safety-related software. This level has been included as normative in order to allow a smooth transition between software developments for non-safety-related systems and those for safety-related systems. The required techniques and measures for each software safety integrity level and for the non-safety-related level are shown in the tables. In this version, the required techniques for level 1 are the same as for level 2, and the required techniques for level 3 are the same as for level 4. This standard does not give guidance on which level of software integrity is appropriate for a given risk. This decision will depend upon the many factors including the nature of the application, the extent to which other systems carry out safety functions and social and economic factors.

It is the function of IEC 62278 and ENV 50129 to specify the safety functions allocated to software.

This standard specifies those measures necessary to achieve these requirements. The process is illustrated in Figure 1.

IEC 62278 and ENV 50129 require that a systematic approach be taken to

i) identify hazards, risks and risk criteria;

ii) identify the necessary risk reduction to meet the risk criteria;

iii) define an overall System Safety Requirements Specification for the safeguards necessary to achieve the required risk reduction;

---

* IEC 61508 (all parts), *Functional safety of electrical/electronic/programmable electronic safety-related systems.*

iv) le choix d'une architecture de système adaptée;

v) la planification, le contrôle et la maîtrise des activités techniques et de management nécessaires pour transformer la Spécification des Exigences de Sécurité du Système en un système de sécurité dont le niveau de sécurité (ou intégrité de sécurité) est validé.

Au fur et à mesure que la spécification se décompose en une conception comprenant des composants et des systèmes liés à la sécurité, l'affectation des niveaux d'intégrité de la sécurité est effectuée. Finalement cela conduit aux niveaux d'intégrité de sécurité logicielle requis.

L'état actuel de la technique est tel que ni l'application des méthodes d'assurance qualité (mesures dites d'évitement de «fautes»), ni l'application d'approches logicielles à tolérance aux «fautes» ne peuvent garantir la sécurité absolue du système. Il n'existe aucun moyen connu de prouver l'absence de défauts dans un logiciel lié à la sécurité raisonnablement complexe, en particulier l'absence de défauts de spécification et de conception.

Les principes appliqués dans le développement de logiciels à haute intégrité, incluent, sans s'y limiter

– des méthodes de conception descendante;
– la modularité;
– la vérification de chaque phase du cycle de vie du développement;
– des modules vérifiés et des bibliothèques de modules;
– une documentation claire;
– des documents aptes à être audités; et
– des tests de validation.

Il est indispensable que ces principes et ceux associés soient correctement appliqués. La présente norme spécifie le niveau d'assurance requis afin de le démontrer pour chaque niveau d'intégrité de la sécurité logicielle.

Une fois que la Spécification des Exigences de Sécurité du Système identifiant toutes les fonctions de sécurité affectées au logiciel et déterminant le niveau d'intégrité de la sécurité du système a été obtenue ou produite, les étapes fonctionnelles de l'application de la présente norme, décrites à la Figure 2, sont les suivantes:

i) définir la Spécification des Exigences du Logiciel et examiner parallèlement l'architecture du logiciel. C'est au travers de l'architecture logicielle qu'est développée la stratégie de base en matière de sécurité pour le logiciel et le niveau d'intégrité de la sécurité logicielle (articles 5, 8 et 9);

ii) concevoir, développer et tester le logiciel selon le Plan d'Assurance Qualité du Logiciel, le niveau d'intégrité de la sécurité logicielle et le cycle de vie du logiciel (article 10);

iii) intégrer le logiciel sur le matériel cible (article 12);

iv) valider le logiciel (article 13);

v) si la maintenance du logiciel est requise pendant la vie opérationnelle, le cas échéant réactiver la présente norme (article 16).

Un certain nombre d'activités se déroulent au cours du développement du logiciel, parmi lesquelles la vérification (article 11), l'évaluation (article 14) et l'assurance qualité (article 15).

Des exigences sont fournies en ce qui concerne les systèmes configurés par les données d'application (article 17).

Des exigences sont également fournies en ce qui concerne la compétence du personnel impliqué dans le développement du logiciel (article 6).

iv) select a suitable system architecture;

v) plan, monitor and control the technical and managerial activities necessary to translate the System Safety Requirements Specification into a Safety-Related System of a validated safety performance (or safety integrity).

As decomposition of the specification into a design comprising safety-related systems and components takes place, further allocation of safety integrity levels is performed. Ultimately this leads to the required software safety integrity levels.

The current state of the art is such that neither the application of quality assurance methods (so-called fault-avoiding measures) nor the application of software fault-tolerant approaches can guarantee the absolute safety of the system. There is no known way to prove the absence of faults in reasonably complex safety-related software, especially the absence of specification and design faults.

The principles applied in developing high-integrity software include, but are not restricted to

– top-down design methods;
– modularity;
– verification of each phase of the development life cycle;
– verified modules and module libraries;
– clear documentation;
– auditable documents; and
– validation testing.

These and related principles must be correctly applied. This standard specifies the level of assurance required to demonstrate this at each software safety integrity level.

After the System Safety Requirements Specification, which identifies all safety functions allocated to software and determines the system safety integrity level, has been obtained or produced, the functional steps in the application of this standard are shown in Figure 2 and are as follows:

i) define the Software Requirements Specification and in parallel consider the software architecture. The software architecture is where the basic safety strategy is developed for the software and the software safety integrity level (clauses 5, 8 and 9);

ii) design, develop and test the software according to the Software Quality Assurance Plan, software safety integrity level and the software life cycle (clause 10);

iii) integrate the software on the target hardware (clause 12);

iv) validate the software (clause 13);

v) if software maintenance is required during operational life then re-activate this standard as appropriate (clause 16).

A number of activities run across the software development. These include verification (clause 11), assessment (clause 14) and quality assurance (clause 15).

Requirements are given for systems which are configured by application data (clause 17).

Requirements are also given for the competency of staff involved in software development (clause 6).

La norme n'impose pas l'utilisation d'un cycle de vie spécifique de développement du logiciel. Cependant un cycle de vie recommandé et un ensemble de documents sont fournis (article 7 et Figures 3 et 4).

Les tableaux ont été établis pour classer diverses techniques/mesures par rapport aux cinq niveaux d'intégrité de la sécurité logicielle. Les tableaux se trouvent à l'annexe A. En référence croisée avec les tableaux, la bibliographie fournit une brève description de chaque technique/mesure avec des références à des sources complémentaires d'informations. La bibliographie se trouve à l'annexe B.

The standard does not mandate the use of a particular software development life cycle. However, a recommended life cycle and documentation set are given (clause 7 and Figures 3 and 4).

Tables have been formulated ranking various techniques/measures against the 5 software safety integrity levels. The tables are in annex A. Cross-referenced to the tables is a bibliography giving a brief description of each technique/measure with references to further sources of information. The bibliography is in annex B.

APPLICATIONS FERROVIAIRES – SYSTÈMES DE SIGNALISATION, DE TÉLÉCOMMUNICATION ET DE TRAITEMENT – LOGICIELS POUR SYSTÈMES DE COMMANDE ET DE PROTECTION FERROVIAIRE

1 Domaine d'application

1.1 La présente Norme internationale spécifie les procédures et les exigences techniques applicables au développement des systèmes électroniques programmables utilisés dans les applications de commande et de protection ferroviaires. Elle est destinée à être utilisée dans tout domaine comportant des implications de sécurité. Ces applications sont susceptibles d'aller du très critique tel que la signalisation de sécurité au non critique comme les systèmes de gestion de l'information. Il est permis de mettre en oeuvre ces systèmes à l'aide de microprocesseurs dédiés, de contrôleurs logiques programmables, de systèmes multi-processeurs distribués, de grands systèmes dotés d'un calculateur central ou à l'aide d'autres architectures.

1.2 La présente norme est exclusivement applicable au logiciel et à l'interaction entre le logiciel et le système auquel il appartient.

1.3 Les niveaux d'intégrité de la sécurité logicielle supérieurs à 0 sont destinés à être utilisés pour des systèmes dans lesquels les conséquences d'une défaillance pourraient provoquer la mort. Des considérations économiques ou environnementales, toutefois, peuvent également justifier l'utilisation de niveaux supérieurs d'intégrité de la sécurité logicielle.

1.4 La présente norme s'applique à tous les logiciels utilisés dans le développement et l'implémentation des systèmes de commande et de protection ferroviaires, y compris

– la programmation de l'application;
– les systèmes d'exploitation;
– les outils d'aide;
– le microprogramme.

La programmation de l'application comprend une programmation de haut niveau, une programmation de bas niveau et une programmation spécifique personnalisée (par exemple: la logique à contact du contrôleur logique programmable).

1.5 L'utilisation de logiciel et d'outils standards disponibles sur le marché est également abordée dans la présente norme.

1.6 La présente norme traite également des exigences applicables aux systèmes configurés par des données d'application.

1.7 La présente norme ne vise pas à traiter des problèmes commerciaux. Il convient de les traiter comme une partie essentielle de tout accord contractuel. Tous les articles de la présente norme font l'objet d'une étude soignée dans toute situation commerciale.

1.8 La présente norme n'a pas d'effet rétroactif. Elle s'applique donc principalement aux nouveaux développements et n'est applicable intégralement aux systèmes existants que s'ils font l'objet de modifications importantes. Seul l'article 16 s'applique pour les modifications mineures.

RAILWAY APPLICATIONS – COMMUNICATIONS, SIGNALLING AND PROCESSING SYSTEMS – SOFTWARE FOR RAILWAY CONTROL AND PROTECTION SYSTEMS

1 Scope

1.1 This International Standard specifies procedures and technical requirements for the development of programmable electronic systems for use in railway control and protection applications. It is aimed at use in any area where there are safety implications. These may range from the very critical, such as safety signalling to the non-critical, such as management information systems. These systems may be implemented using dedicated microprocessors, programmable logic controllers, multiprocessor distributed systems, larger scale central processor systems or other architectures.

1.2 This standard is applicable exclusively to software and the interaction between software and the system of which it is part.

1.3 Software safety integrity levels above zero are for use in systems in which the consequences of failure could include loss of life. Economic or environmental considerations, however, may also justify the use of higher software safety integrity levels.

1.4 This standard applies to all software used in development and implementation of railway control and protection systems including

– application programming;
– operating systems;
– support tools;
– firmware.

Application programming comprises high-level programming, low-level programming and special-purpose programming (for example, Programmable Logic Controller ladder logic).

1.5 The use of standard, commercially available software and tools is also addressed in this standard.

1.6 This standard also addresses the requirements for systems configured by application data.

1.7 This standard is not intended to address commercial issues. These should be addressed as an essential part of any contractual agreement. All the clauses of this standard will need careful consideration in any commercial situation.

1.8 This standard is not intended to be retrospective. It therefore applies primarily to new developments and only applies in its entirety to existing systems if these are subjected to major modifications. For minor changes, only clause 16 applies.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 62278, *Applications ferroviaires – Spécification et démonstration de la fiabilité, de la disponibilité, de la maintenabilité et de la sécurité (FDMS)* [1]

CEI 62280-1, *Applications ferroviaires – Systèmes de signalisation, de télécommunication et de traitement – Partie 1: Communication de sécurité sur des systèmes de transmission fermés* [1]

CEI 62280-2, *Applications ferroviaires – Systèmes de signalisation, de télécommunication et de traitement – Partie 2: Communication de sécurité sur des systèmes de transmission ouverts* [1]

ISO 9000:2000, *Systèmes de management de la qualité – Principes essentiels et vocabulaire*

ISO 9000-3:1997, *Normes pour le management de la qualité et l'assurance de la qualité – Partie 3 – Lignes directrices pour l'application de l'ISO 9001:1994 au développement, à la mise à disposition et à la maintenance du logiciel*

ISO 9001:1994, *Systèmes qualité – Modèle pour l'assurance de la qualité en conception, développement, production, installation et prestations associées*

ENV 50129:1998, *Applications ferroviaires – Systèmes électroniques de sécurité pour la signalisation*

3 Définitions

Pour les besoins de la présente Norme internationale, les définitions suivantes s'appliquent. Quant aux termes ne figurant pas dans la liste ci-après, il convient de consulter les références indiquées ci-dessous dans l'ordre de priorité.

ISO 9000:2000, *Systèmes de management de la qualité– Principes essentiels et vocabulaire*

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

IEEE STD 610.12, *IEEE Standard Glossary of Software Engineering Terminology* (publié en anglais seulement)

ISO/CEI 2382 (toutes les parties), *Technologies de l'information – Vocabulaire*

ISO/CEI 9126 (toutes les parties), *Génie du logiciel – Qualité des produits*

---

[1] A publier.

2 Normative references

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

IEC 62278, *Railway applications – Specification and demonstration of reliability, availability, maintainability and safety (RAMS)* [1]

IEC 62280-1, *Railway applications – Communication, signalling and processing systems – Part 1: Safety-related communication in closed transmission systems* [1]

IEC 62280-2, *Railway applications – Communication, signalling and processing systems – Part 2: Safety-related communication in open transmission systems* [1]

ISO 9000:2000, *Quality management systems – Fundamentals and vocabulary*

ISO 9000-3:1997, *Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software*

ISO 9001:1994, *Quality systems – Model for quality assurance in design, development, production, installation and servicing*

ENV 50129:1998, *Railway applications – Safety related electronic systems for signalling*

3 Definitions

For the purposes of this International Standard, the following definitions apply. For terms not defined here, the following references should be consulted in order of priority:

ISO 9000:2000, *Quality management systems – Fundamentals and vocabulary*

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

IEEE STD 610.12, *IEEE standard glossary of software engineering terminology*

ISO/IEC 2382 (all parts), *Information technology – Vocabulary*

ISO/IEC 9126 (all parts), *Software engineering – Product quality*

---

1 To be published.

3.1
évaluation
processus d'analyse afin de déterminer si l'autorité de conception et le chargé de validation ont réussi à obtenir un produit qui satisfait les exigences spécifiées et afin de formuler un jugement sur le fait que le produit répond à l'objectif attendu

3.2
chargé d'évaluation
personne ou agent désigné pour mener à bien l'évaluation

3.3
disponibilité
aptitude d'un produit à être en état d'accomplir une fonction requise dans des conditions données, à un instant donné ou pendant un intervalle de temps donné, en supposant que la fourniture des moyens extérieurs nécessaires est assurée

3.4
logiciel standard disponible dans le commerce (COTS)
logiciel défini par les besoins du marché, disponible dans le commerce et dont l'adéquation aux besoins a été démontrée par un large éventail d'utilisateurs

3.5
autorité de conception
organisme responsable de la formulation d'un choix de conception pour remplir les exigences spécifiées ainsi que de la supervision des développements ultérieurs et de la mise en marche d'un système dans l'environnement prévu

3.6
concepteur
une ou plusieurs personnes désignées par l'autorité de conception pour analyser et transformer les exigences spécifiées en choix de conception acceptables et présentant l'intégrité de la sécurité requise

3.7
élément
partie d'un produit identifié comme étant une unité de base ou un bloc. Un élément peut être simple ou complexe

3.8
erreur
écart par rapport à la conception prévue pouvant conduire à un comportement ou à une défaillance inattendu du système

3.9
défaillance
écart par rapport aux performances spécifiées d'un système. Une défaillance est la conséquence d'un défaut ou d'une erreur dans un système

3.10
défaut
état anormal pouvant conduire à une erreur ou à une défaillance dans un système. Un défaut peut être aléatoire ou systématique

3.11
évitement de «fautes»
utilisation de techniques de conception visant à éviter l'introduction de défauts pendant la conception et la construction du système

3.1
assessment
process of analysis to determine whether the Design Authority and the Validator have achieved a product that meets the specified requirements and to form a judgement as to whether the product is fit for its intended purpose

3.2
Assessor
person or agent appointed to carry out the assessment

3.3
availability
ability of a product to be in a state to perform a required function under given conditions at a given instant of time or over a given time interval, assuming the required external resources are provided

3.4
commercial off-the-shelf (COTS) software
software defined by market-driven need, commercially available and whose fitness for purpose has been demonstrated by a broad spectrum of commercial users

3.5
design authority
body responsible for the formulation of a design solution to fulfil the specified requirements and for overseeing the subsequent development and setting to work of a system in its intended environment

3.6
designer
one or more persons assigned by the Design Authority to analyse and transform specified requirements into acceptable design solutions which have the required safety integrity

3.7
element
part of a product that has been determined to be a basic unit or building block. An element may be simple or complex

3.8
error
deviation from the intended design which could result in unintended system behaviour or failure

3.9
failure
deviation from the specified performance of a system. A failure is the consequence of a fault or error in a system

3.10
fault
abnormal condition that could lead to an error or a failure in a system. A fault can be random or systematic

3.11
fault avoidance
use of design techniques which aim to avoid the introduction of faults during the design and construction of the system

3.12
tolérance aux «fautes»
capacité intégrée d'un système à délivrer d'une manière correcte et continue un service tel qu'il est spécifié, en présence d'un nombre limité de défauts matériels ou logiciels

3.13
microprogramme
ensemble ordonné d'instructions et de données associées, enregistré de manière à ce qu'il soit fonctionnellement indépendant du stockage principal et résidant en principe dans une mémoire morte

3.14
logiciel générique
logiciel pouvant être utilisé pour une grande variété d'installations simplement en fournissant des données propres à l'application

3.15
réalisateur
une ou plusieurs personnes désignées par l'autorité de conception en vue de transformer des choix de conception en leur réalisation physique

3.16
produit
ensemble d'éléments interconnectés pour former un système, un sous-système ou une pièce d'un équipement, de manière à satisfaire aux exigences spécifiées. Dans la présente norme, un produit peut être considéré comme étant entièrement constitué d'éléments logiciels ou de documentation

3.17
contrôleur logique programmable (PLC)
système de commande informatisé, doté d'une mémoire programmable par l'utilisateur pour le stockage des instructions, pour réaliser des fonctions spécifiques

3.18
fiabilité
aptitude d'une entité à réaliser une fonction dans des conditions données, pendant une période de temps donnée

3.19
traçabilité des exigences
objectif de la traçabilité des exigences est d'assurer qu'il est possible de montrer que toutes les exigences ont été correctement remplies

3.20
risque
combinaison de la fréquence ou de la probabilité, et de la conséquence d'un événement dangereux spécifié

3.21
sécurité
absence de niveaux de risque inacceptables

3.22
autorité de tutelle
organisme chargé de certifier que le système lié à la sécurité est apte au service et qu'il est conforme à toutes les exigences de sécurité statutaires et réglementaires applicables

3.12
fault tolerance
built-in capability of a system to provide continued correct provision of service as specified, in the presence of a limited number of hardware or software faults

3.13
firmware
ordered set of instructions and associated data stored in a way that is functionally independent of main storage, usually in a ROM

3.14
generic software
generic software is software which can be used for a variety of installations purely by the provision of application-specific data

3.15
implementer
one or more persons assigned by the Design Authority to transform specified designs into their physical realisation

3.16
product
collection of elements, interconnected to form a system, subsystem or item of equipment, in a manner which meets the specified requirements. In this standard, a product may be considered to consist entirely of elements of software or documentation

3.17
programmable logic controller (PLC)
solid-state control system which has a user programmable memory for storage of instructions to implement specific functions

3.18
reliability
ability of an item to perform a required function under given conditions for a given period of time

3.19
requirements traceability
objective of requirements traceability is to ensure that all requirements can be shown to have been properly met

3.20
risk
combination of the frequency, or probability, and the consequence of a specified hazardous event

3.21
safety
freedom from unacceptable levels of risk

3.22
safety authority
body responsible for certifying that the safety-related system is fit for service and complies with relevant statutory and regulatory safety requirements

3.23
logiciel lié à la sécurité
logiciel qui porte la responsabilité de la sécurité

3.24
logiciel
création intellectuelle comprenant les programmes, les procédures, les règles et toute documentation associée en rapport avec le fonctionnement d'un système

NOTE Le logiciel est indépendant de son support.

3.25
cycle de vie du logiciel
activités survenant pendant une période qui commence à la spécification du logiciel et se termine quand le logiciel ne peut plus être utilisé. Généralement, le cycle de vie du logiciel inclut une phase de spécifications, une phase de développement, une phase de tests, une phase d'intégration, une phase d'installation et une phase de maintenance

3.26
maintenabilité du logiciel
capacité d'un logiciel à être modifié pour corriger les défauts, pour améliorer la performance ou d'autres attributs, ou pour l'adapter à un environnement différent

3.27
maintenance du logiciel
action ou ensemble d'actions effectuée sur un logiciel après acceptation de celui-ci par l'utilisateur final dans le but d'améliorer, d'étendre et/ou de corriger ses fonctions

3.28
niveau d'intégrité de la sécurité logicielle
numéro de classification déterminant les techniques et mesures à appliquer de manière à réduire à un niveau convenable, les défauts résiduels du logiciel

3.29
niveau d'intégrité de la sécurité du système
numéro indiquant le degré de confiance requis sur le fait qu'un système respectera ses caractéristiques de sécurité spécifiées

3.30
traçabilité
facilité avec laquelle un lien peut être établi entre deux ou plusieurs produits d'un processus de développement, et plus particulièrement, ceux possédant entre eux une relation de prédécesseur à successeur ou de maître à esclave

3.31
validation
activité visant à démontrer, par l'analyse et des tests, que le produit satisfait totalement aux exigences spécifiées

3.32
chargé de validation
personne ou agent désigné pour procéder à la validation

3.33
vérification
activité visant à déterminer, par l'analyse et des tests, que le résultat en sortie de chaque phase du cycle de vie satisfait aux exigences de la phase précédente

3.23
safety-related software
software which carries responsibility for safety

3.24
software
intellectual creation comprising the programs, procedures, rules and any associated documentation pertaining to the operation of a system

NOTE Software is independent of the media used for transport.

3.25
software life cycle
activities occurring during a period of time that starts when software is conceived and ends when the software is no longer available for use. The software life cycle typically includes a requirements phase, development phase, test phase, integration phase, installation phase and a maintenance phase

3.26
software maintainability
capability of a system to be modified to correct faults, improve performance or other attributes, or adapt it to a different environment

3.27
software maintenance
action, or set of actions, carried out on software after its acceptance by the final user. The aim is to improve, increase and/or correct its functionality

3.28
software safety integrity level
classification number which determines the techniques and measures that have to be applied in order to reduce residual software faults to an appropriate level

3.29
system safety integrity level
number which indicates the required degree of confidence so that a system will meet its specified safety features

3.30
traceability
degree to which a relationship can be established between two or more products of a development process, especially those having a predecessor/successor or master/subordinate relationship to one another

3.31
validation
activity of demonstration, by analysis and test, that the product meets, in all respects, its specified requirements

3.32
Validator
person or agent appointed to carry out validation

3.33
verification
activity of determination, by analysis and test, that the output of each phase of the life cycle fulfils the requirements of the previous phase

3.34
chargé de vérification
personne ou agent désigné pour procéder à la vérification

4 Objets et conformité

4.1 Dans chacun des paragraphes suivants, les objets et les exigences de l'article sont décrits de façon détaillée.

4.2 Pour se conformer à la présente norme, on doit montrer que chacune des exigences a été satisfaite vis-à-vis du niveau d'intégrité de la sécurité logicielle défini et que l'objet de l'article a donc été atteint.

4.3 Là où une exigence est qualifiée par les termes «Dans la limite requise par le niveau d'intégrité de la sécurité logicielle», cela indique qu'une panoplie de techniques et de mesures peuvent être utilisées pour satisfaire à cette exigence.

4.4 Là où 4.3 s'applique, il convient d'utiliser les tableaux détaillés dans la présente norme pour faciliter la sélection des techniques et des mesures appropriées au niveau d'intégrité de la sécurité logicielle.

4.5 Si une technique ou une mesure comptant parmi celles hautement recommandées (HR) dans les tableaux n'est pas utilisée, il convient de détailler et d'enregistrer le raisonnement sous-jacent, soit dans le Plan d'Assurance Qualité du Logiciel, soit dans un autre document mentionné par ce dernier. Cette formalité n'est pas nécessaire si l'on a recours à une combinaison approuvée des techniques fournies dans le tableau correspondant.

4.6 Si l'utilisation d'une technique ou d'une mesure non contenue dans les tableaux est proposée, son efficacité et sa pertinence par rapport au respect de l'exigence particulière et de l'objet global de l'article doivent être justifiées et enregistrées, soit dans le Plan d'Assurance Qualité du Logiciel, soit dans un autre document mentionné par le Plan d'Assurance Qualité du Logiciel.

4.7 La conformité aux exigences d'un article particulier et à leurs techniques et mesures respectives détaillées dans les tableaux doit être évaluée par l'inspection des documents requis par la présente norme, par d'autres preuves tangibles, l'audit et par l'observation des tests.

4.8 La présente norme nécessite l'utilisation d'un ensemble de techniques et leur application correcte. Ces techniques sont exigées par les tableaux et détaillées dans la bibliographie.

5 Niveaux d'intégrité de la sécurité logicielle

5.1 Objet

Décrire l'affectation au logiciel, de niveaux d'intégrité de la sécurité logicielle.

3.34
Verifier
person or agent appointed to carry out verification

4 Objectives and conformance

4.1 In each of the following subclauses, the objectives and requirements of the clause are detailed.

4.2 To conform to this standard, it shall be shown that each of the requirements have been satisfied to the software safety integrity level defined and therefore the clause objective has been met.

4.3 Where a requirement is qualified by the words "To the extent required by the software safety integrity level", this indicates that a range of techniques and measures can be used to satisfy that requirement.

4.4 Where 4.3 applies, the tables detailed in this standard should be used to assist in the selection of techniques and measures appropriate to the software safety integrity level.

4.5 If a technique or measure is ranked as highly recommended (HR) in the tables then the rationale for not using that technique should be detailed and recorded either in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan. This is not necessary if an approved combination of techniques given in the corresponding table is used.

4.6 If a technique or measure is proposed to be used that is not contained in the tables then its effectiveness and suitability in meeting the particular requirement and overall objective of the clause shall be justified and recorded in either the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan.

4.7 Compliance with the requirements of a particular clause and their respective techniques and measures detailed in the tables shall be assessed by the inspection of documents required by this standard, other objective evidence, auditing and the witnessing of tests.

4.8 This standard requires the use of a package of techniques and their correct application. These techniques are required from the tables and detailed in the bibliography.

5 Software safety integrity levels

5.1 Objective

To describe the assignment of software safety integrity levels to the software.

5.2 Exigences

5.2.1 Conformément à la CEI 62278 et à la norme européenne ENV 50129, on doit produire

– la Spécification des Exigences du Système,
– la Spécification des Exigences de Sécurité du Système,
– la Description de l'Architecture du Système,
– le Plan de Sécurité du Système,

comprenant

– les fonctions de sécurité,
– la configuration ou l'architecture du système,
– les exigences de fiabilité matérielle,
– les exigences d'intégrité de la sécurité.

Le niveau d'intégrité de la sécurité logicielle doit être spécifié par le suivi du processus général pour obtenir un niveau d'intégrité de la sécurité identifié dans la CEI 62278.

5.2.2 Le niveau d'intégrité de la sécurité logicielle requis doit être décidé sur la base du niveau de risque associé à l'utilisation du logiciel dans le système et du niveau d'intégrité de la sécurité du système.

5.2.3 Sans plus de précautions, le niveau d'intégrité de la sécurité logicielle doit être au minimum identique au niveau d'intégrité de la sécurité du système. Cependant, s'il existe des mécanismes empêchant le système d'atteindre un état non sûr, suite à la défaillance d'un module du logiciel, le niveau d'intégrité de la sécurité de ce module peut être réduit.

5.2.4 Les risques qui doivent être pris en compte sont ceux aboutissant à

– la perte d'une vie ou de vies humaines;
– des blessures ou des maladies pour les personnes;
– la pollution de l'environnement; et
– la perte ou les dommages de biens.

5.2.5 Il est permis de quantifier un risque mais il n'est pas possible de spécifier l'intégrité de la sécurité logicielle de cette façon. En conséquence, dans le cadre de la présente norme, l'intégrité de la sécurité logicielle doit être spécifiée comme appartenant à l'un des cinq niveaux suivants:

Niveau d'intégrité de la sécurité logicielle	Description de l'intégrité de la sécurité logicielle
4	Très élevée
3	Elevée
2	Moyenne
1	Faible
0	Non liée à la sécurité

5.2.6 Le niveau d'intégrité de la sécurité logicielle doit être spécifié dans la Spécification des Exigences du Logiciel (article 8). Si divers composants du logiciel ont des niveaux d'intégrité de la sécurité du logiciel différents, ceux-ci doivent être spécifiés dans la Spécification d'Architecture du Logiciel (article 9).

5.2 Requirements

5.2.1 There shall be produced, in accordance with IEC 62278 and ENV 50129,

– the System Requirements Specification,
– the System Safety Requirements Specification,
– the System Architecture Description,
– the System Safety Plan,

which include

– safety functions;
– configuration or architecture of the system;
– hardware reliability requirements;
– safety integrity requirements.

The software safety integrity level shall be specified through following the general process for obtaining a safety integrity level identified in IEC 62278.

5.2.2 The required software safety integrity level shall be decided on the basis of the level of risk associated with the use of the software in the system and the system safety integrity level.

5.2.3 Without further precautions, the software safety integrity level shall be, as a minimum, identical to the system safety integrity level. However, if mechanisms exist to prevent the failure of a software module from causing the system to go to an unsafe state, the software safety integrity level of the module may be reduced.

5.2.4 Risks which shall be taken into account are those associated with the following hazard consequences:

– loss of human life or lives;
– injuries to, or illness, of persons;
– environmental pollution; and
– loss of, or damage, to property.

5.2.5 Risk may be quantified but it is not possible to specify the software safety integrity in the same manner. Therefore, for this standard, the software safety integrity shall be specified as one of the following five levels:

Software safety integrity level	Description of software safety integrity
4	Very high
3	High
2	Medium
1	Low
0	Non-safety-related

5.2.6 The software safety integrity level shall be specified in the Software Requirements Specification (clause 8). If different software components have different software safety integrity levels, these shall be specified in the Software Architecture Specification (clause 9).

6 Personnel et responsabilités

6.1 Objet

S'assurer que l'ensemble du personnel responsable du logiciel a la compétence nécessaire pour s'acquitter de ces responsabilités.

6.2 Exigences

6.2.1 Pour le moins, le fournisseur et/ou le développeur et le client doivent mettre en oeuvre les parties applicables de l'ISO 9001, conformément aux directives contenues dans l'ISO 9000-3.

6.2.2 A l'exception de l'intégrité de la sécurité logicielle de niveau 0, le processus de sécurité doit être mis en oeuvre sous le contrôle d'une organisation de sécurité appropriée et conforme au paragraphe «Organisation de la sécurité» de l'article «Preuve de la gestion de la sécurité» de la norme ENV 50129.

6.2.3 L'ensemble du personnel impliqué dans toutes les phases du cycle de vie du logiciel, y compris dans les activités de gestion, doit avoir la formation, l'expérience et les qualifications appropriées.

6.2.4 Il est hautement recommandé que la formation, l'expérience et les qualifications de l'ensemble du personnel impliqué dans toutes les phases du cycle de vie du logiciel, y compris dans les activités de gestion, soient justifiées par rapport à l'application particulière, excepté pour ce qui concerne l'intégrité de la sécurité logicielle de niveau 0.

6.2.5 La justification contenue en 6.2.4 doit être enregistrée dans le Plan d'Assurance Qualité du Logiciel et doit inclure la preuve des compétences dans les secteurs suivants, selon le cas:

i) ingénierie appropriée au secteur d'application;

ii) ingénierie du logiciel;

iii) ingénierie en systèmes informatiques;

iv) ingénierie en matière de sécurité;

i) cadre juridique et réglementaire.

6.2.6 Un chargé d'évaluation du logiciel indépendant doit être nommé. Voir aussi 6.2.10 et 14.4.1.

6.2.7 L'autorité doit être donnée au chargé d'évaluation pour accomplir l'évaluation du logiciel.

6.2.8 Tout au long du cycle de vie du logiciel, les parties impliquées doivent être indépendantes, dans la limite requise par le niveau d'intégrité de la sécurité logicielle, conformément à la Figure 5, qui doit être interprétée de la façon suivante.

A tous les niveaux d'intégrité de la sécurité logicielle, le chargé d'évaluation doit être agréé par l'autorité de tutelle et être indépendant du fournisseur sauf dans les conditions exposées en 6.2.10.

Le concepteur/réalisateur, le chargé de vérification et le chargé de validation peuvent tous appartenir à la même entreprise mais les règles suivantes garantissant un minimum d'indépendance doivent être respectées.

6 Personnel and responsibilities

6.1 Objective

To ensure that all personnel who have responsibilities for the software are competent to discharge those responsibilities.

6.2 Requirements

6.2.1 As a minimum, the supplier and/or developer and the customer shall implement the relevant parts of ISO 9001, in accordance with the guidelines contained in ISO 9000-3.

6.2.2 Except at software safety integrity level zero, the safety process shall be implemented under the control of an appropriate safety organisation which is compliant with the "Safety organisation" subclause in the "Evidence of Safety Management" clause of ENV 50129.

6.2.3 All personnel involved in all the phases of the Software Life Cycle, including management activities, shall have the appropriate training, experience and qualifications.

6.2.4 It is highly recommended that the training, experience and qualifications of all personnel involved in all the phases of the Software Life Cycle, including management activities, be justified with respect to the particular application, except at software safety integrity level zero.

6.2.5 The justification contained in 6.2.4 shall be recorded in the Software Quality Assurance Plan and shall include evidence of competency in the following areas, as appropriate:

i) engineering appropriate to the application area;

ii) software engineering;

iii) computer-systems engineering;

iv) safety engineering;

v) legal and regulatory framework.

6.2.6 An independent Assessor for the software shall be appointed. See also 6.2.10 and 14.4.1.

6.2.7 The Assessor shall be given authority to perform the assessment of the software.

6.2.8 Throughout the Software Life Cycle, the parties involved shall be independent, to the extent required by the software safety integrity level, in accordance with Figure 5, which shall be interpreted as follows.

At all software safety integrity levels, the Assessor shall be approved by the Safety Authority and independent from the supplier except in the circumstances defined in 6.2.10.

The Designer/Implementer, Verifier and Validator can all belong to the same company but the following rules for minimum independence shall be complied with.

Au niveau 0 de l'intégrité de la sécurité logicielle:

Il n'existe aucune contrainte; le concepteur/réalisateur, le chargé de vérification et le chargé de validation peuvent tous être la même personne.

Aux niveaux 1 et 2 de l'intégrité de la sécurité logicielle:

Le chargé de vérification et le chargé de validation peuvent être la même personne mais ils ne doivent pas être le concepteur/réalisateur. Toutefois, le concepteur/réalisateur, le chargé de vérification et le chargé de validation peuvent tous rendre compte par l'intermédiaire du chef de projet.

Aux niveaux 3 et 4 de l'intégrité de la sécurité logicielle, il existe deux arrangements acceptables.

a) Le chargé de vérification et le chargé de validation peuvent être la même personne mais ils ne doivent pas être également le concepteur/réalisateur. En outre, le chargé de vérification et le chargé de validation ne doivent pas rendre compte par l'intermédiaire du chef de projet comme le fait le concepteur/réalisateur et ils doivent avoir autorité pour empêcher la sortie du produit.
b) Le concepteur/réalisateur, le chargé de vérification et le chargé de validation doivent tous être des personnes distinctes. Le concepteur/réalisateur et le chargé de vérification peuvent rendre compte par l'intermédiaire du chef de projet, alors que le chargé de validation ne le doit pas. Le chargé de validation doit avoir autorité pour empêcher la sortie du produit.

6.2.9 Les responsables des différents articles sont les suivants:

Spécification des Exigences du Logiciel (article 8)	Concepteur
Spécification des Tests des Exigences du Logiciel (article 8)	Chargé de validation
Architecture du logiciel (article 9)	Concepteur
Conception et développement du logiciel (article 10)	Concepteur
Vérification et tests du logiciel (article 11)	Chargé de vérification
Intégration logiciel/matériel (article 12)	Concepteur
Validation du logiciel (article 13)	Chargé de validation
Evaluation du logiciel (article 14)	Chargé d'évaluation

6.2.10 A la discrétion de l'autorité de tutelle, le chargé d'évaluation peut appartenir à l'organisation du fournisseur ou à celle du client mais, dans de tels cas, le chargé d'évaluation doit

– être autorisé par l'autorité de tutelle,
– être totalement indépendant de l'équipe projet,
– rendre compte directement à l'autorité de tutelle.

7 Problèmes liés au cycle de vie et documentation

7.1 Objets

7.1.1 Structurer le développement du logiciel en phases et activités définies.

7.1.2 Enregistrer toutes les informations en rapport avec le logiciel tout au long de son cycle de vie.

At software safety integrity level 0:

There are no constraints; the Designer/Implementer, Verifier and Validator can all be the same person.

At software safety integrity levels 1 and 2:

The Verifier and Validator can be the same person but they shall not be the Designer/Implementer. However, the Designer/Implementer, Verifier and Validator can all report through the Project Manager.

At software safety integrity levels 3 and 4 there are two permissible arrangements.

a) The Verifier and Validator can be the same person but they shall not also be the Designer/Implementer. In addition, the Verifier and Validator shall not all report through the Project Manager as the Designer/Implementer does and they shall have the authority to prevent the release of the product.
b) The Designer/Implementer, Verifier and Validator must all be separate persons. The Designer/Implementer and Verifier can report through the Project Manager, whereas the Validator shall not, and the Validator shall have the authority to prevent the release of the product.

6.2.9 The parties responsible for the various clauses are as follows:

Software Requirements Specification (clause 8)	Designer
Software Requirements Test Specification (clause 8)	Validator
Software Architecture (clause 9)	Designer
Software Design and Development (clause 10)	Designer
Software Verification and Testing (clause 11)	Verifier
Software/Hardware Integration (clause 12)	Designer
Software Validation (clause 13)	Validator
Software Assessment (clause 14)	Assessor

6.2.10 At the discretion of the Safety Authority, the Assessor may be part of the supplier's organisation or of the customer's organisation but, in such cases, the Assessor shall

– be authorised by the Safety Authority,
– be totally independent from the project team,
– report direct to the Safety Authority.

7 Life cycle issues and documentation

7.1 Objectives

7.1.1 To structure the development of the software into defined phases and activities.

7.1.2 To record all information pertinent to the software throughout the life cycle of the software.

7.2 Exigences

7.2.1 Un modèle de cycle de vie pour le développement du logiciel doit être choisi. Ce modèle doit être détaillé dans le Plan d'Assurance Qualité du Logiciel conformément à l'article 15 de la présente norme. Deux modèles de cycle de vie sont décrits à titre d'exemple dans les Figures 3 et 4.

7.2.2 Les procédures d'assurance qualité doivent être menées à bien parallèlement aux activités du cycle de vie et doivent utiliser les mêmes termes.

7.2.3 Toutes les activités à exécuter pendant une phase doivent être définies avant le commencement de la phase. Chaque phase du cycle de vie du logiciel doit être divisée en tâches élémentaires avec une entrée, une sortie et une activité bien définies pour chacune d'elles.

7.2.4 Le Plan d'Assurance Qualité du Logiciel doit décrire quelles sont les étapes de la vérification et les rapports exigés.

7.2.5 Tous les documents doivent être structurés de manière à permettre un enrichissement continu parallèlement au processus de développement.

7.2.6 La traçabilité des documents doit être assurée par le fait que chaque document comporte un numéro de référence unique et qu'une relation définie et documentée existe avec les autres documents. Chaque terme, acronyme ou abréviation, doit avoir le même sens dans tous les documents. Si cela n'est pas possible pour des raisons historiques, les différents sens doivent être cités avec leurs références.

Sauf pour la documentation des logiciels COTS (voir 9.4.5) ou pour la documentation des logiciels développés antérieurement (voir 9.4.6), chaque document doit, en outre, être rédigé conformément aux règles suivantes:

a) il doit contenir ou mettre en oeuvre toutes les conditions et exigences applicables du document précédent avec lequel il a une relation hiérarchique;

b) il ne doit pas être en contradiction avec le document précédent;

c) chaque terme, acronyme ou abréviation, doit avoir la même signification dans tous les documents;

d) il doit être fait référence à chaque entité ou concept en utilisant le même nom ou la même description dans tous les documents.

Le résultat du processus de traçabilité doit faire l'objet d'une gestion de configuration formalisée.

7.2.7 Le contenu de tous les documents doit être enregistré dans un format approprié à la manipulation, le traitement et le stockage.

7.2.8 Les documents listés dans la table de correspondance des documents (voir ci-après) doivent être produits, dans la limite requise par le niveau d'intégrité de la sécurité logicielle.

7.2.9 En fonction de la taille, de la complexité et du cycle de vie du logiciel en cours de développement, le nombre de documents requis et distincts à produire variera. Il est permis de combiner certains documents (sous réserve qu'il n'existe aucune perte des détails exigés par le processus). Pour les projets de grande taille, il n’est pas exclu qu’il soit nécessaire de subdiviser la documentation répertoriée (de façon hiérarchique) en un certain nombre de sous-documents plus faciles à manier. Les documents qui ont été produits par des équipes ou des entités indépendantes ne doivent pas être regroupés en un seul document.

7.2 Requirements

7.2.1 A life cycle model for the development of software shall be selected. It shall be detailed in the Software Quality Assurance Plan in accordance with clause 15 of this standard. For example, two life cycle models are shown in Figures 3 and 4.

7.2.2 Quality Assurance procedures shall run in parallel with life cycle activities and use the same terminology.

7.2.3 All activities to be performed during a phase shall be defined prior to the phase commencing. Each phase of the software life cycle shall be divided into elementary tasks with a well defined input, output and activity for each of them.

7.2.4 The Software Quality Assurance Plan shall describe which verification steps and reports are required.

7.2.5 All documents shall be structured to allow continued expansion in parallel with the development process.

7.2.6 Traceability of documents shall be provided for by each document having a unique reference number and a defined and documented relationship with other documents. Each term, acronym or abbreviation shall have the same meaning in every document. If, for historical reasons, this is not possible, the different meanings shall be listed and the references given.

In addition, each document except documents for COTS software (see 9.4.5) or previously developed software (see 9.4.6) shall be written according to the following rules:

a) it shall contain or implement all applicable conditions and requirements of the predecessor document with which it has a hierarchical relationship;

b) it shall not contradict the predecessor document;

c) each term, acronym or abbreviation shall have the same meaning in every document;

d) each item or concept shall be referred to by the same name or description in every document.

The output of the traceability process shall be the subject of formal configuration management.

7.2.7 The contents of all documents shall be recorded in a form appropriate for manipulation, processing and storage.

7.2.8 To the extent required by the software safety integrity level, the documents listed in the Documents Cross-Reference Table (see below) shall be produced.

7.2.9 Depending upon the size, complexity and life cycle of the software being developed, the number of separate documents required to be produced will vary. Some documents may be combined (providing there is no loss of the required detail in the process). For large projects it may be necessary to sub-divide the documentation listed (in a hierarchical manner) into a number of more manageable child documents. Documents which have been produced by independent teams or entities shall not be combined into a single document.

7.2.10 La relation entre les divers documents identifiés dans l'article 7 peut également être définie par l'utilisation d'une DCRT (table de correspondance des documents). Pour chaque document listé dans la colonne «Documents», la phase et l'article associé à sa création peuvent être trouvés en lisant horizontalement et verticalement depuis la cellule contenant le symbole «■». Les phases dans lesquelles ce document est utilisé peuvent être trouvées par lecture verticale depuis les cellules marquées par le symbole «♦». L'article ou toute référence à la définition du document peut être trouvé dans la colonne «Défini en». Lorsqu'un article est cité, il convient de vérifier les articles le suivant car ils sont susceptibles de contenir des informations supplémentaires. Il y a lieu de noter que la référence pour le Plan de Gestion de la Configuration du Logiciel est indiquée entre parenthèses, simplement parce que cet article se réfère à l'ISO 9001.

7.2.10 The relationship between the various documents identified in clause 7 can also be defined by using a DCRT (a Document Cross-Reference Table). For each document listed in the "Documents" column, the phase and clause associated with its creation can be found by reading horizontally and vertically from the cell containing the symbol "■". The phases in which it is used can be found by reading vertically from the cells marked with the symbol "♦". The clause or other reference to the definition of the document can be found in the "Where defined" column. Where a clause is given, the following clauses should also be checked as they may contain further information. It should be noted that the reference for the Software Configuration Management Plan is shown in brackets because that clause simply references ISO 9001.

TABLE DE CORRESPONDANCE DES DOCUMENTS

Article	8	9	10	11	12	13	14	15	16	Documents	Défini en
Titre	SEL	AL	CDL	Ver	ILM	Val	Eva	Q	Ma		
PHASES (*) = en parallèle avec d'autres phases											
ENTRÉES SYSTÈME	♦			♦	♦					Spécification des Exigences du Système	ENV 50129 annexe B.2.3
	♦	♦		♦	♦	♦	♦			Spécification des Exigences de Sécurité du Système	ENV 50129 annexe B.2.4
	♦				♦					Description de l'Architecture du Système	ENV 50129 annexe B.2.1
										Plan de Sécurité du Système	ENV 50129 CEI 62278
PLANIFICATION DU LOGICIEL (*)	♦	♦	♦	♦		♦	♦	▮		Plan d'Assurance Qualité Lg	15.4.3
						♦	♦	▮		Plan de Gestion de la Configuration Lg	(15.4.2)
				▮		♦	♦			Plan de Vérification Lg	11.4.1
				▮		♦	♦			Plan de Tests d'Intégration Lg	11.4.5
					▮	♦	♦			Plan de Tests d'Intégration Lg/Mt	12.4.1
						▮	♦			Plan de validation du Logiciel	13.4.3
							♦		▮	Plan de Maintenance du Logiciel	16.4.3
										Plan de Préparation des Données	17.4.2.1
										Plan de Tests des Données	17.4.2.4
EXIGENCES DU LOGICIEL	▮	♦	♦	♦	♦	♦	♦			Spécification des Exigences Lg	8.4.1
										Spécification des Exigences de l'Application	17.4.1.1
	▮			♦	♦	♦	♦			Spécification des Tests des Exigences Lg	8.4.13
				▮						Rapport des Vérifications des Exigences Lg	11.4.11
CONCEPTION DU LOGICIEL		▮	♦	♦	♦	♦	♦			Spécification de l'Architecture Lg	9.4.1
			▮	♦	♦	♦	♦			Spécification de la Conception Lg	10.4.3
				▮						Rapport de Vérification de l'Architecture et de la Conception Lg	11.4.12
CONCEPTION DES MODULES DU LOGICIEL			▮	♦	♦	♦	♦			Spécification de la Conception des Modules Lg	10.4.3
			▮	♦	♦	♦	♦			Spécification des Tests des Modules Lg	10.4.14
				▮						Rapport de Vérification des Modules Lg	11.4.13
CODE			▮	♦	♦	♦	♦			Code Source Lg	
				▮		♦	♦			Rapport de Vérification du Code Source Lg	11.4.14
TEST DES MODULES			▮	♦						Rapport de Tests des Modules Lg	10.4.14
INTÉGRATION DU LOGICIEL				▮						Rapport de Tests d'Intégration Lg	11.4.15
										Rapport de Tests des Données	17.4.2.4
INTÉGRATION LG/MT					▮					Rapport de Tests d'Intégration Lg/Mt	12.4.8
VALIDATION (*)						▮				Rapport de Validation du Logiciel	13.4.10
ÉVALUATION (*)							▮			Rapport d'Evaluation du Logiciel	14.4.9
MAINTENANCE									▮	Enregistrements des Evolutions Lg	16.4.9
									▮	Enregistrements de la Maintenance du Logiciel	16.4.8

DOCUMENTS CROSS-REFERENCE TABLE

Clause	8	9	10	11	12	13	14	15	16	Documents	Where defined
Title	SRS	SA	SDD	SVer	S/H I	SVal	Ass	Q	Ma		
PHASES (*) = in parallel with other phases											
SYSTEM INPUTS	♦			♦	♦					System Requirements Specification	ENV 50129 annex B.2.3
	♦	♦		♦	♦	♦	♦			System Safety Requirements Specification	ENV 50129 annex B.2.4
	♦				♦					System Architecture Description	ENV 50129 annex B.2.1
										System Safety Plan	ENV 50129 IEC 62278
SW PLANNING (*)	♦	♦	♦	♦		♦	♦	▮		Sw Quality Assurance Plan	15.4.3
						♦	♦	▮		Sw Configuration Management Plan	(15.4.2)
				▮		♦	♦			Sw Verification Plan	11.4.1
				▮		♦	♦			Sw Integration Test Plan	11.4.5
					▮	♦	♦			Sw/Hw Integration Test Plan	12.4.1
						▮	♦			Sw Validation Plan	13.4.3
							♦		▮	Sw Maintenance Plan	16.4.3
										Data Preparation Plan	17.4.2.1
										Data Test Plan	17.4.2.4
SW REQUIREMENTS	▮	♦	♦	♦	♦	♦	♦			Sw Requirements Specification	8.4.1
										Application Requirements Specification	17.4.1.1
	▮			♦	♦	♦	♦			Sw Requirements Test Specification	8.4.13
				▮						Sw Requirements Verification Report	11.4.11
SW DESIGN		▮	♦	♦	♦	♦	♦			Sw Architecture Specification	9.4.1
			▮	♦	♦	♦	♦			Sw Design Specification	10.4.3
				▮						Sw Arch. and Design Verification Report	11.4.12
SW MODULE DESIGN			▮	♦	♦	♦	♦			Sw Module Design Specification	10.4.3
			▮	♦	♦	♦	♦			Sw Module Test Specification	10.4.14
				▮						Sw Module Verification Report	11.4.13
CODE			▮	♦	♦	♦	♦			Sw Source Code	
				▮		♦	♦			Sw Source Code Verification Report	11.4.14
MODULE TESTING			▮	♦						Sw Module Test Report	10.4.14
SW INTEGRATION				▮						Sw Integration Test Report	11.4.15
										Data Test Report	17.4.2.4
SW/HW INTEGRATION					▮					Sw/Hw Integration Test Report	12.4.8
VALIDATION (*)						▮				Sw Validation Report	13.4.10
ASSESSMENT (*)							▮			Sw Assessment Report	14.4.9
MAINTENANCE									▮	Sw Change Records	16.4.9
									▮	Sw Maintenance Records	16.4.8

8 Spécification des Exigences du Logiciel

8.1 Objets

8.1.1 Décrire un document qui définit un ensemble complet d'exigences pour le logiciel respectant toutes les exigences du système dans la limite requise par le niveau d'intégrité de la sécurité logicielle. Il sert de document complet destiné à chaque ingénieur en logiciel, qui ainsi n'a pas besoin de rechercher les exigences dans d'autres documents.

8.1.2 Décrire la Spécification des Tests des Exigences du Logiciel.

8.2 Documents en entrée

1) Spécification des Exigences du Système
2) Spécification des Exigences de Sécurité du Système
3) Description de l'Architecture du Système
4) Plan d'Assurance Qualité du Logiciel

8.3 Documents en sortie

1) Spécification des Exigences du Logiciel
2) Spécification des Tests des Exigences du Logiciel

8.4 Exigences

8.4.1 La Spécification des Exigences du Logiciel doit exprimer les propriétés requises du logiciel en cours de développement et non les procédures permettant de les développer. Ces propriétés, qui sont toutes (à l'exception de la sécurité) définies dans la série de normes ISO/CEI 9126, doivent inclure

– la fonctionnalité (y compris le dimensionnement et la performance en temps de réponse);
– la fiabilité et la maintenabilité;
– la sécurité (y compris les fonctions de sécurité et les niveaux d'intégrité de la sécurité logicielle qui leur sont associés);
– l'efficacité;
– l'utilisabilité;
– la portabilité.

Le niveau d'intégrité de la sécurité logicielle doit être déterminé comme indiqué dans l'article 5 et enregistré dans la Spécification des Exigences du Logiciel.

8.4.2 Dans la limite requise par le niveau d'intégrité de la sécurité logicielle, la Spécification des Exigences du Logiciel doit être exprimée et structurée de manière à être

i) complète, claire, précise, sans équivoque, susceptible d'être vérifiée, testée, maintenue et réalisée,
ii) traçable en remontant à tous les documents mentionnés en 8.2.

8.4.3 La Spécification des Exigences du Logiciel doit inclure des modes d'expression et des descriptions qui sont compréhensibles par le personnel responsable impliqué dans tout le cycle de vie du système.

8 Software requirements specification

8.1 Objectives

8.1.1 To describe a document which defines a complete set of requirements for the software meeting all System Requirements to the extent required by the Software Safety Integrity level. It serves the purpose of a comprehensive document for each software engineer and makes it unnecessary for him to screen for requirements in any other document.

8.1.2 To describe the Software Requirements Test Specification.

8.2 Input documents

1) System Requirements Specification
2) System Safety Requirements Specification
3) System Architecture Description
4) Software Quality Assurance Plan

8.3 Output documents

1) Software Requirements Specification
2) Software Requirements Test Specification

8.4 Requirements

8.4.1 The Software Requirements Specification shall express the required properties of the software being developed, not the procedures to develop them. These properties, which are all (except safety) defined in ISO/IEC 9126, shall include

– functionality (including capacity and response time performance);
– reliability and maintainability;
– safety (including safety functions and their associated software safety integrity levels);
– efficiency;
– usability;
– portability.

The software safety integrity level shall be derived as defined in clause 5 and recorded in the Software Requirements Specification.

8.4.2 To the extent required by the software safety integrity level the Software Requirements Specification shall be expressed and structured in such a way that it is

i) complete, clear precise, unequivocal, verifiable, testable, maintainable and feasible,
ii) traceable back to all documents mentioned in 8.2.

8.4.3 The Software Requirements Specification shall include modes of expression and descriptions which are understandable to the responsible personnel involved in the whole life cycle of the system.

8.4.4 La Spécification des Exigences du Logiciel doit identifier et documenter toutes les interfaces avec tout autre système, qu'il se trouve à l'intérieur ou à l'extérieur de l'équipement sous contrôle, y compris les opérateurs, chaque fois qu'une connexion directe existe ou qu'elle est prévue.

8.4.5 Tous les modes de fonctionnement applicables doivent être détaillés dans la Spécification des Exigences du Logiciel.

8.4.6 Tous les modes de comportement applicables des systèmes électroniques programmables, notamment le comportement de défaillance, doivent être détaillés dans la Spécification des Exigences du Logiciel.

8.4.7 Toute contrainte entre le matériel et le logiciel doit être identifiée et documentée dans la Spécification des Exigences du Logiciel.

8.4.8 La Spécification des Exigences du Logiciel doit indiquer le degré d'auto-contrôle du logiciel et le degré spécifié de contrôle du matériel par le logiciel. L'auto-contrôle logiciel consiste en la détection et le compte rendu par le logiciel de ses propres défaillances et erreurs.

8.4.9 La Spécification des Exigences du Logiciel doit inclure des exigences concernant le test périodique des fonctions, dans la limite requise par la Spécification des Exigences de Sécurité du Système.

8.4.10 La Spécification des Exigences du Logiciel doit inclure des exigences permettant la testabilité de toutes les fonctions de sécurité pendant le fonctionnement global du système, dans la limite requise par la Spécification des Exigences de Sécurité du Système.

8.4.11 Les fonctions que le logiciel est tenu d'exécuter, notamment celles liées à la réalisation du niveau requis d'intégrité de la sécurité du système, doivent être clairement identifiées dans la Spécification des Exigences du Logiciel.

8.4.12 Les fonctions non liées à la sécurité que le logiciel est tenu d'exécuter doivent être clairement identifiées dans la Spécification des Exigences du Logiciel.

8.4.13 Une Spécification des Tests des Exigences du Logiciel doit être développée à partir de la Spécification des Exigences du Logiciel. Cette Spécification des Tests doit être utilisée pour la vérification de toutes les exigences décrites dans la Spécification des Exigences du Logiciel et également comme une description des tests à effectuer sur le logiciel terminé.

8.4.14 La Spécification des Tests des Exigences du Logiciel doit identifier les scénarios de tests pour chacune des fonctions requises, y compris

i) les signaux d'entrée requis avec leurs séquences et leurs valeurs;

ii) les signaux de sortie attendus avec leurs séquences et leurs valeurs;

iii) les critères d'acceptation en incluant les aspects performance et qualité.

8.4.15 La traçabilité des exigences doit être largement prise en considération dans la validation d'un système lié à la sécurité et des moyens doivent être fournis pour lui permettre d'être démontrée d'un bout à l'autre de toutes les phases du cycle de vie.

8.4.16 On doit montrer que tout élément non traçable n'a pas d'impact sur la sécurité ou l'intégrité du système.

8.4.4 The Software Requirements Specification shall identify and document all interfaces with any other systems, either within or outside the equipment under control, including operators, wherever a direct connection exists or is planned.

8.4.5 All relevant modes of operation shall be detailed in the Software Requirements Specification.

8.4.6 All relevant modes of behaviour of the programmable electronics, in particular failure behaviour, shall be detailed in the Software Requirements Specification.

8.4.7 Any constraints between the hardware and the software shall be identified and documented in the Software Requirements Specification.

8.4.8 The Software Requirements Specification shall indicate the degree of software self-checking and the specified degree of hardware checking by the software. Software self-checking consists of the detection and reporting by the software of its own failures and errors.

8.4.9 The Software Requirements Specification shall include requirements for the periodic testing of functions to the extent required by the System Safety Requirements Specification.

8.4.10 The Software Requirements Specification shall include requirements to enable all safety functions to be testable during overall system operation to the extent required by the System Safety Requirements Specification.

8.4.11 When the software is required to perform functions especially those related to achieving the required system safety integrity level, then these shall be clearly identified in the Software Requirements Specification.

8.4.12 When the software is required to perform non-safety functions then these shall be clearly identified in the Software Requirements Specification.

8.4.13 A Software Requirements Test Specification shall be developed from the Software Requirements Specification. This test specification shall be used for verification of all the requirements as described in the Software Requirements Specification and also as a description of the tests to be performed on the completed software.

8.4.14 The Software Requirements Test Specification shall identify for each required function the test cases including

i) the required input signals with their sequences and their values;

ii) the anticipated output signals with their sequences and their values;

iii) the acceptance criteria, including performance and quality aspects.

8.4.15 Traceability to requirements shall be an important consideration in the validation of a safety-related system and means shall be provided to allow this to be demonstrated throughout all phases of the life cycle.

8.4.16 Any untraceable material shall be shown to have no bearing upon the safety or integrity of the system.

9 Architecture du logiciel

9.1 Objets

9.1.1 Développer une architecture logicielle qui respecte les exigences de la Spécification des Exigences du Logiciel dans la limite requise par le niveau d'intégrité de la sécurité logicielle.

9.1.2 Examiner les contraintes qu'impose l'architecture du système sur le logiciel.

9.1.3 Identifier et évaluer l'importance des interactions matériel/logiciel pour la sécurité.

9.1.4 Choisir une méthode de conception si aucune n'a été définie au préalable.

9.2 Documents en entrée

1) Spécification des Exigences du Logiciel
2) Spécification des Exigences de Sécurité du Système
3) Description de l'Architecture du Système
4) Plan d'Assurance Qualité du Logiciel

9.3 Documents en sortie

Spécification de l'Architecture du Logiciel

9.4 Exigences

9.4.1 L'architecture du logiciel proposée doit être établie par le fournisseur du logiciel et/ou le développeur et détaillée dans la Spécification de l'Architecture du Logiciel.

9.4.2 La Spécification de l'Architecture du Logiciel doit prendre en considération la faisabilité de la réalisation de la Spécification des Exigences du Logiciel au niveau requis d'intégrité de la sécurité logicielle.

9.4.3 La Spécification de l'Architecture du Logiciel doit identifier, évaluer et détailler l'importance de toutes les interactions matériel/logiciel. Comme cela est prescrit dans la CEI 62278 et dans la norme ENV 50129, les études préliminaires sur les interactions entre le matériel et le logiciel doivent avoir été enregistrées dans la Spécification des Exigences de Sécurité du Système.

9.4.4 La Spécification de l'Architecture du Logiciel doit identifier tous les composants logiciels et déterminer pour ces composants:

i) si ces composants sont nouveaux, existants ou standards;

ii) si ces composants ont été précédemment validés et, si tel est le cas, leurs conditions de validation;

iii) le niveau d'intégrité de la sécurité logicielle du composant.

9 Software architecture

9.1 Objectives

9.1.1 To develop a software architecture that achieves the requirements of the Software Requirements Specification to the extent required by the software safety integrity level.

9.1.2 To review the requirements placed on the software by the system architecture.

9.1.3 To identify and evaluate the significance of hardware/software interactions for safety.

9.1.4 To choose a design method if one has not been previously defined.

9.2 Input documents

1) Software Requirements Specification
2) System Safety Requirements Specification
3) System Architecture Description
4) Software Quality Assurance Plan

9.3 Output documents

Software Architecture Specification

9.4 Requirements

9.4.1 The proposed software architecture shall be established by the software supplier and/or developer and detailed in the Software Architecture Specification.

9.4.2 The Software Architecture Specification shall consider the feasibility of achieving the Software Requirements Specification at the required software safety integrity level.

9.4.3 The Software Architecture Specification shall identify, evaluate and detail the significance of all hardware/software interactions. As required by IEC 62278 and ENV 50129, the preliminary studies concerning the interactions between hardware and software shall have been recorded in the System Safety Requirements Specification.

9.4.4 The Software Architecture Specification shall identify all software components and for these components identify

i) whether these components are new, existing or proprietary;
ii) whether these components have been previously validated and if so their validation conditions;
iii) the software safety integrity level of the component.

9.4.5 L'utilisation d'un logiciel standard disponible dans le commerce doit être soumise aux restrictions suivantes:

i) pour le niveau 0 de l'intégrité de la sécurité logicielle, l'usage d'un logiciel standard disponible dans le commerce doit être accepté sans précaution supplémentaire;

ii) s'il est prévu d'utiliser des logiciels standards disponibles dans le commerce au niveau 1 ou 2 de l'intégrité de la sécurité logicielle, ils doivent être inclus dans le processus de validation des logiciels;

iii) s'il est prévu d'utiliser des logiciels standards disponibles dans le commerce au niveau 3 ou 4 de l'intégrité de la sécurité logicielle, les précautions suivantes doivent également être prises:

   a) les logiciels standards disponibles dans le commerce doivent être inclus dans les tests de validation;

   b) une analyse des défaillances possibles des logiciels standards disponibles dans le commerce doit être réalisée;

   c) une stratégie doit être définie en vue de détecter les défaillances des logiciels standards disponibles dans le commerce et de protéger le système contre ces défaillances;

   d) la stratégie de protection doit faire l'objet de tests de validation;

   e) des journaux des erreurs doivent exister et doivent être évalués;

   f) dans la mesure du possible, seules les fonctions les plus simples des logiciels standards disponibles dans le commerce doivent être utilisées.

9.4.6 S'il est prévu d'utiliser un logiciel développé précédemment comme partie de la conception, celui-ci doit être clairement identifié et documenté. La Spécification de l'Architecture du Logiciel doit justifier de l'adéquation du logiciel à satisfaire à la Spécification des Exigences du Logiciel et au niveau d'intégrité de la sécurité logicielle. Il faut que les répercussions éventuelles sur le reste du système de tout changement apporté au logiciel soient soigneusement prises en considération de manière à décider si cette situation nécessite un nouveau contrôle et une nouvelle évaluation. On doit apporter la preuve que les spécifications d'interface avec les autres modules qui ne sont pas à nouveau vérifiés, validés et évalués sont respectées.

9.4.7 Chaque fois que possible les modules logiciels vérifiés existants, développés conformément à la présente norme doivent être utilisés dans la conception.

9.4.8 L'architecture du logiciel doit minimiser la partie de sécurité de l'application.

9.4.9 Lorsque le logiciel est constitué de composants de niveaux d'intégrité de la sécurité logicielle différents, tous les composants du logiciel doivent alors être traités comme s'ils appartenaient au niveau le plus élevé d'intégrité de la sécurité logicielle, à moins qu'il n'existe une preuve d'indépendance entre les composants de plus haut niveau d'intégrité de la sécurité logicielle et les composants de moindre niveau d'intégrité de la sécurité logicielle. Cette preuve doit être enregistrée dans la Spécification de l'Architecture du Logiciel.

9.4.10 La Spécification de l'Architecture du Logiciel doit identifier la stratégie de développement du logiciel dans la limite requise par le niveau d'intégrité de la sécurité logicielle. La Spécification de l'Architecture du Logiciel doit être exprimée et structurée de manière à être

i) complète, claire, précise, sans équivoque, susceptible d'être vérifiée, testée, maintenue et réalisée,

ii) traçable depuis la Spécification des Exigences du Logiciel.

9.4.5 The use of COTS software shall be subject to the following restrictions:

i) for software safety integrity level 0, the use of the COTS software shall be accepted with no further precautions;

ii) if COTS software is to be used at software safety integrity levels 1 or 2, it shall be included in the software validation process;

iii) if COTS software is to be used at software safety integrity levels 3 or 4, the following precautions shall also be taken:

   a) the COTS software shall be included in the validation testing;

   b) an analysis of possible failures of the COTS software shall be carried out;

   c) a strategy shall be defined to detect failures of the COTS software and to protect the system from these failures;

   d) the protection strategy shall be the subject of validation testing;

   e) error logs shall exist and shall be evaluated;

   f) as far as practicable, only the simplest functions of the COTS software shall be used.

9.4.6 If previously developed software is to be used as part of the design then it shall be clearly identified and documented. The Software Architecture Specification shall justify the software's suitability in satisfying the Software Requirements Specification and the software safety integrity level. The effects of any changes to the software on the rest of the system must be carefully considered in order to decide whether they require a re-inspection and re-assessment. There shall be evidence that interface specifications to other modules which are not being re-verified, re-validated and re-assessed are being followed.

9.4.7 Whenever possible existing verified software modules developed according to this standard shall be used in the design.

9.4.8 The Software Architecture shall minimise the safety part of the application.

9.4.9 Where the software consists of components of different software safety integrity levels then all of the software components shall be treated as belonging to the highest software safety integrity level unless there is evidence of independence between the higher software safety integrity level components and the lower software safety integrity level components. This evidence shall be recorded in the Software Architecture Specification.

9.4.10 The Software Architecture Specification shall identify the strategy for the software development to the extent required by the software safety integrity level. The Software Architecture Specification shall be expressed and structured in such a way that it is

i) complete, clear, precise, unequivocal, verifiable, testable, maintainable and feasible,

ii) traceable back to the Software Requirements Specification.

9.4.11 La Spécification de l'Architecture du Logiciel doit justifier l'équilibre existant entre les stratégies d'évitement de «fautes» et de tolérance aux «fautes».

9.4.12 La Spécification de l'Architecture du Logiciel doit démontrer que les techniques et les mesures choisies forment un ensemble qui satisfait à la Spécification des Exigences du Logiciel au niveau requis d'intégrité de la sécurité logicielle.

10 Conception et développement du logiciel

10.1 Objets

10.1.1 Concevoir et mettre en oeuvre un logiciel à un niveau défini d'intégrité de la sécurité logicielle à partir de la Spécification des Exigences du Logiciel et de la Spécification de l'Architecture du Logiciel.

10.1.2 Réaliser un logiciel qui soit analysable, testable, vérifiable et maintenable. Les tests des modules font aussi partie de cette phase. Puisque la vérification et les tests seront un élément crucial de la validation, un intérêt tout particulier doit être porté aux besoins en matière de vérification et de test tout au long de la conception et du développement, de manière à s'assurer que le système résultant et son logiciel seront facilement testables depuis le commencement.

10.1.3 Sélectionner un jeu d'outils adaptés, y compris des langages et des compilateurs, pour le niveau requis d'intégrité de la sécurité logicielle, pour l'ensemble du cycle de vie du logiciel, qui permettent d'aider à la vérification, la validation, l'évaluation et la maintenance.

10.1.4 Effectuer l'intégration du logiciel.

10.2 Documents en entrée

1) Spécification des Exigences du Logiciel
2) Spécification de l'Architecture du Logiciel
3) Plan d'Assurance Qualité du Logiciel

10.3 Documents en sortie

1) Spécification de la Conception du Logiciel
2) Spécifications de la Conception des Modules Logiciel
3) Spécifications des Tests des Modules Logiciel
4) Code Source du Logiciel et Documentation de Support
5) Rapport des Tests des Modules Logiciel

10.4 Exigences

10.4.1 La Spécification des Exigences du Logiciel et la Spécification de l'Architecture du Logiciel doivent être disponibles, mais pas nécessairement finalisées avant le démarrage du processus de conception.

10.4.2 La taille et la complexité du logiciel développé doivent être limitées au minimum.

10.4.3 La Spécification de la Conception du Logiciel doit décrire la conception du logiciel basée sur une décomposition en modules, chaque module comportant une Spécification de la Conception des Modules Logiciel et une Spécification des Tests des Modules Logiciel.

9.4.11 The Software Architecture Specification shall justify the balance taken between the strategies of avoiding faults and handling faults.

9.4.12 The Software Architecture Specification shall justify that the techniques and measures chosen form a set which satisfies the Software Requirements Specification at the required software safety integrity level.

10 Software design and implementation

10.1 Objectives

10.1.1 To design and implement software of a defined software safety integrity level from the Software Requirements Specification and the Software Architecture Specification.

10.1.2 To achieve software which is analysable, testable, verifiable and maintainable. Module testing is also included in this phase. As verification and test will be a critical element in the validation, particular consideration shall be given to verification and test needs throughout the design and development, in order to ensure the resultant system and its software will be readily testable from the outset.

10.1.3 To select a suitable set of tools, including languages and compilers, for the required software safety integrity level, over the whole life cycle of the software which assists verification, validation, assessment and maintenance.

10.1.4 To carry out software integration.

10.2 Input documents

1) Software Requirements Specification
2) Software Architecture Specification
3) Software Quality Assurance Plan

10.3 Output documents

1) Software Design Specification
2) Software Module Design Specification
3) Software Module Test Specification
4) Software Source Code and Supporting Documentation
5) Software Module Test Report

10.4 Requirements

10.4.1 The Software Requirements Specification and the Software Architecture Specification shall be available, although not necessarily finalised, prior to the start of the design process.

10.4.2 The size and complexity of the software developed shall be kept to a minimum.

10.4.3 The Software Design Specification shall describe the software design based on a decomposition into modules with each module having a Software Module Design Specification and a Software Module Test Specification.

10.4.4 La Spécification de la Conception du Logiciel doit décrire

i) les composants du logiciel tracés par rapport à l'architecture du logiciel et leur niveau d'intégrité de la sécurité,

ii) les interfaces des composants du logiciel avec l'environnement,

iii) les interfaces entre les composants du logiciel,

iv) les structures de données,

v) la répartition des exigences sur les composants,

vi) les algorithmes principaux et le séquencement,

vii) les diagrammes.

10.4.5 Les Spécifications de la Conception des Modules du Logiciel doivent décrire (une Spécification de la Conception des Modules du Logiciel par module)

i) l'identification de tous les composants de plus bas niveau du logiciel (appelés modules dans la présente norme) tracés par rapport au niveau supérieur,

ii) leurs interfaces détaillées avec l'environnement et les autres modules avec le détail de leurs entrées/sorties,

iii) leur niveau d'intégrité de la sécurité,

iv) les algorithmes et les structures de données détaillés.

Il est recommandé que chaque Spécification de la Conception des Modules du Logiciel soit autosuffisante et permette le codage du module correspondant.

10.4.6 Chaque module logiciel doit être lisible, compréhensible et testable.

10.4.7 Un jeu d'outils adaptés, y compris des méthodes de conception, des langages et des compilateurs, doit être sélectionné pour le niveau requis d'intégrité de la sécurité logicielle, pour l'ensemble du cycle de vie du logiciel.

10.4.8 Chaque fois que possible, des outils de tests automatiques et des outils de développement intégrés doivent être utilisés. Ils doivent prendre en comptes les besoins des chargés de vérification et chargés de validation.

10.4.9 Dans la limite requise par le niveau d'intégrité de la sécurité logicielle, le langage de programmation sélectionné doit disposer d'un traducteur/compilateur comportant l'un des éléments suivants:

i) un «Certificat de Validation» par rapport à une norme nationale/internationale reconnue;

ii) un rapport d'évaluation qui détaille le bien-fondé de son utilisation dans ce contexte;

iii) un processus redondant basé sur un contrôle de signature qui permet de détecter les erreurs de traduction.

10.4.10 Le langage choisi doit respecter les exigences suivantes:

i) le langage choisi doit offrir des caractéristiques qui facilitent l'identification des erreurs de programmation;

ii) le langage choisi doit offrir des caractéristiques compatibles avec la méthode de conception.

10.4.4 The Software Design Specification shall address

i) software components traced back to software architecture and their safety integrity level,

ii) interfaces of software components with the environment,

iii) interfaces between the software components,

iv) data structure,

v) partitioning of requirements on components,

vi) main algorithms and sequencing,

vii) diagrams.

10.4.5 The Software Module Design Specifications shall address (one Software module Design Specification per module)

i) identification of all lowest software components (called modules in the present standard) traced back to the upper level;

ii) their detailed interfaces with environment and other modules with detailed inputs and outputs;

iii) their safety integrity level;

iv) detailed algorithms and data structures.

Each Software Module design Specification should be self-sufficient and allow coding of the corresponding module.

10.4.6 Each software module shall be readable, understandable and testable.

10.4.7 A suitable set of tools, including design methods, languages and compilers shall be selected for the required software safety integrity level over the whole life cycle of the software.

10.4.8 When applicable, automatic testing tools and integrated development tools shall be used. This shall take account of the needs of the Verifier and Validator.

10.4.9 To the extent required by the software safety integrity level, the programming language selected shall have a translator/compiler which has one of the following:

i) a "Certificate of Validation" to a recognised National/International Standard;

ii) an assessment report which details its fitness for purpose;

iii) a redundant signature control based process that provides detection of the translation errors.

10.4.10 The language chosen shall meet the following requirements:

i) the language chosen shall contain features that facilitate the identification of programming errors;

ii) the language chosen shall support features that match the design method.

10.4.11 Quand 10.4.10 ne peut être respecté, il est nécessaire qu'une justification pour tout autre langage détaillant le bien-fondé de son utilisation dans ce contexte soit enregistrée dans la Spécification de l'Architecture du Logiciel ou dans le Plan d'Assurance Qualité du Logiciel.

10.4.12 Des normes de codage doivent être développées et utilisées pour le développement de tous les logiciels. Ces normes doivent être référencées dans le Plan d'Assurance Qualité du Logiciel (voir 15.4.5).

10.4.13 Les normes de codage doivent spécifier le bon usage de la programmation, proscrire les caractéristiques peu sûres du langage et décrire des procédures pour la documentation du code source. Pour le moins, chaque module logiciel doit contenir dans le code source, les informations définies dans la liste (non exhaustive) suivante:

– auteur;

– historique de la configuration;

– courte description.

L'utilisation d'un format standard de présentation est recommandée. Il convient que ce format soit le même pour tous les modules.

10.4.14 Tests des Modules Logiciel: Chaque module doit avoir une Spécification des Tests des Modules Logiciel par rapport à laquelle il doit être testé. Ces tests doivent montrer que chaque module exécute la fonction prévue. La Spécification des Tests des Modules du Logiciel doit définir le taux requis de la couverture des tests.

Un Rapport des Tests des Modules du Logiciel doit être rédigé et les points suivants doivent être respectés:

i) on doit y trouver un compte rendu des résultats de tests et la mention du respect ou non par chaque module des exigences de sa Spécification de la Conception des Modules du Logiciel;

ii) on doit fournir pour chaque module, un compte rendu de la couverture des tests montrant que toutes les instructions du code source du module ont été exécutées au mois une fois;

iii) sa forme doit permettre l'audit;

iv) les scénarios de tests et leurs résultats doivent être enregistrés dans un format lisible par une machine pour une analyse ultérieure; si cela est faisable, il convient que les tests soient répétables et qu'ils soient réalisés automatiquement.

L'opération qui consiste à contrôler que le module a satisfait à sa spécification des tests est une activité de vérification, voir l'article 11.

10.4.15 Conformément au niveau d'intégrité de la sécurité logicielle, la méthode de conception choisie doit offrir des caractéristiques qui facilitent

i) l'abstraction, la modularité et d'autres caractéristiques pour maîtriser la complexité;

ii) l'expression claire et précise de

  – la fonctionnalité,

  – le flux d'informations entre les composants,

  – les séquencements et les informations temporelles,

  – le parallélisme,

  – la structure et les propriétés des données;

iii) la compréhension humaine;

iv) la vérification et la validation.

10.4.11 When 10.4.10 cannot be satisfied then a justification for any alternative language detailing its fitness for purpose shall be recorded in the Software Architecture Specification or Software Quality Assurance Plan.

10.4.12 Coding standards shall be developed and used for the development of all software. These shall be referenced in the Software Quality Assurance Plan (see 15.4.5).

10.4.13 The coding standards shall specify good programming practice, proscribe unsafe language features and describe procedures for source code documentation. As a minimum, each software module shall contain in the source code the information defined in the following (non-exhaustive) list:

– author;
– configuration history;
– short description.

The use of a standard form for this information is recommended. It should be the same for all the modules.

10.4.14 Software Module Testing: Each module shall have a Software Module Test Specification which the module shall be tested against. These tests shall show that each module performs its intended function. The Software Module Test Specification shall define the required degree of test coverage.

A Software Module Test Report shall be produced and shall include the following features:

i) a statement of the test results and whether each module has met the requirements of its Software Module Design Specification;
ii) a statement of test coverage shall be provided for each module, showing that all source code instructions have been executed at least once;
iii) it shall be in a form that is auditable;
iv) test cases and their results shall be recorded in a machine readable form for subsequent analysis; tests should be repeatable and be performed by automatic means, if practicable.

Checking that the module has correctly satisfied its test specification is a verification activity, see clause 11.

10.4.15 In accordance with the required software safety integrity level the design method chosen shall possess features that facilitate

i) abstraction, modularity and other features which control complexity;
ii) the clear and precise expression of
    – functionality,
    – information flow between components,
    – sequencing and time-related information,
    – concurrency,
    – data structure and properties;
iii) human comprehension;
iv) verification and validation.

10.4.16 La méthode de conception choisie doit offrir des caractéristiques qui facilitent la maintenance du logiciel. Ces caractéristiques doivent inclure la modularité, le masquage des informations et l'encapsulation.

10.4.17 L'intégration des modules logiciels doit être le processus de regroupement progressif de chacun des modules logiciel testés au préalable, au sein d'une entité composite (ou au sein d'un certain nombre de sous-systèmes composites) de manière à ce que les interfaces des modules et le logiciel assemblé puissent être convenablement testés avant l'intégration et les tests du système.

10.4.18 Dans le contexte de la présente norme et dans une limite appropriée au niveau d'intégrité de la sécurité logicielle spécifié, la traçabilité doit concerner particulièrement

i) la traçabilité des exigences vers la conception ou les autres objets qui répondent à ces exigences;

ii) la traçabilité des objets de la conception vers les objets développés qui les instancient.

11 Vérification et tests du logiciel

11.1 Objets

Dans la limite requise par le niveau d'intégrité de la sécurité logicielle, tester et évaluer les produits d'une phase donnée pour assurer l'exactitude et la cohérence vis-à-vis des produits et des normes stipulés en entrée pour cette phase.

11.2 Documents en entrée

1) Spécification des Exigences du Système
2) Spécification des Exigences de Sécurité du Système
3) Spécification des Exigences du Logiciel
4) Spécification des Tests des Exigences du Logiciel
5) Spécification de l'Architecture du Logiciel
6) Spécification de la Conception du Logiciel
7) Spécifications de la Conception des Modules Logiciel
8) Spécifications des Tests des Modules Logiciel
9) Code Source du Logiciel et Documentation de Support
10) Plan d'Assurance Qualité du Logiciel
11) Rapport des Tests des Modules Logiciel

11.3 Documents en sortie

1) Plan de Vérification du Logiciel
2) Rapport de Vérification des Exigences du Logiciel
3) Rapport de Vérification de l'Architecture et de la Conception du Logiciel
4) Rapport de Vérification des Modules Logiciel
5) Rapport de Vérification du Code Source du Logiciel
6) Plan des Tests d'Intégration du Logiciel
7) Rapport des Tests d'Intégration du Logiciel

10.4.16 The design method chosen shall possess features that facilitate software maintenance. Such features shall include modularity, information hiding and encapsulation.

10.4.17 The integration of software modules shall be the process of progressively combining individual and previously tested modules of software into a composite whole (or into a number of composite subsystems) in order that the module interfaces and the assembled software may be adequately proven prior to system integration and test.

10.4.18 Within the context of this standard, and to a degree appropriate to the specified software safety integrity level, traceability shall particularly address

i) traceability of requirements to the design or other objects which fulfil them;

ii) traceability of design objects to the implementation objects which instantiate them.

11 Software verification and testing

11.1 Objective

To the extent required by the software safety integrity level, to test and evaluate the products of a given phase to ensure correctness and consistency with respect to the products and standards provided as input to that phase.

11.2 Input documents

1) System Requirements Specification
2) System Safety Requirements Specification
3) Software Requirements Specification
4) Software Requirements Test Specification
5) Software Architecture Specification
6) Software Design Specification
7) Software Module Design Specification
8) Software Module Test Specification
9) Software Source code and supporting documentation
10) Software Quality Assurance Plan
11) Software Module Test Report

11.3 Output documents

1) Software Verification Plan
2) Software Requirements Verification Report
3) Software Architecture and Design Verification Report
4) Software Module Verification Report
5) Software Source Code Verification Report
6) Software Integration Test Plan
7) Software Integration Test Report

11.4 Exigences

11.4.1 Un Plan de Vérification du Logiciel doit être créé pour que les activités de vérification puissent être convenablement menées et que les besoins particuliers en matière de conception ou autre vérification puissent être correctement assurés. Pendant le développement (et selon la taille du système), il est admis que le plan soit divisé en un certain nombre de sous-documents qui seront naturellement complétés à mesure que les besoins détaillés de vérification deviennent plus clairs.

11.4.2 Le Plan de Vérification du Logiciel doit documenter tous les critères, techniques et outils à utiliser dans le processus de vérification de cette phase.

11.4.3 Le Plan de Vérification du Logiciel doit décrire les activités à exécuter pour assurer l'exactitude et la cohérence vis-à-vis des produits et des normes stipulés en entrée pour cette phase.

11.4.4 Le Plan de Vérification du Logiciel doit inclure les éléments suivants:

i) la sélection des stratégies et des techniques de vérification. Pour éviter de compliquer inutilement l'évaluation de l'activité de vérification et de tests, il convient de préférer la sélection de modèles de tests, de méthodes etc., qui sont eux-mêmes facilement analysables;

ii) la sélection et l'utilisation du matériel de tests du logiciel;

iii) la sélection et la documentation des activités de vérification;

iv) l'évaluation des résultats de vérification obtenus;

v) l'évaluation des exigences en matière de fiabilité;

vi) les rôles et responsabilités des personnes impliquées dans le processus de test;

vii) le degré requis de couverture des tests à réaliser.

11.4.5 Le Plan de Tests d'Intégration du Logiciel doit documenter les éléments suivants:

i) les scénarios de test et les données liées au test;

ii) les types de tests à réaliser;

iii) l'environnement de tests, les outils, la configuration et les programmes;

iv) les critères de tests sur lesquels l'arrêt des tests sera jugé.

11.4.6 Dans chaque phase de développement, il doit être montré que les exigences en matière de fonctionnement, de fiabilité, de performance et de sécurité sont respectées.

11.4.7 La vérification doit être effectuée par un parti indépendant dans la limite requise par le niveau d'intégrité de la sécurité logicielle tel que défini dans la Figure 5.

11.4.8 Tout test qui n'est pas entièrement documenté et qui est exécuté par le concepteur avant la vérification ne doit pas être considéré comme faisant partie de celle-ci.

11.4.9 Les résultats de chaque vérification doivent être conservés dans un format pouvant être audité et défini ou référencé dans le Plan de Vérification du Logiciel.

11.4 Requirements

11.4.1 A Software Verification Plan shall be created in order that verification activities may be properly directed and that particular design or other verification needs may be suitably provided for. During development (and depending upon the size of the system) the plan may be sub-divided into a number of child documents and be naturally added to, as the detailed needs of verification become clearer.

11.4.2 The Software Verification Plan shall document all the criteria, techniques and tools to be utilised in the verification process for that phase.

11.4.3 The Software Verification Plan shall describe the activities to be performed to ensure correctness and consistency with respect to the products and standards provided as input to that phase.

11.4.4 The Software Verification Plan shall address the following:

i) the selection of verification strategies and techniques. To avoid undue complexity in the assessment of the verification and test activity, preference should be given to the selection of test cases and methods etc., which are in themselves readily analysable;

ii) the selection and utilisation of the software test equipment;

iii) the selection and documentation of verification activities;

iv) the evaluation of verification results gained;

v) the evaluation of the reliability requirements;

vi) the roles and responsibilities of those involved in the test process;

vii) the degree of test coverage required to be achieved.

11.4.5 The Software Integration Test Plan shall document the following:

i) test cases and test data;

ii) types of tests to be performed;

iii) test environment, tools, configuration and programs;

iv) test criteria on which the completion of the test will be judged.

11.4.6 In each development phase it shall be shown that the functional, reliability, performance and safety requirements are met.

11.4.7 Verification shall be carried out by an independent party to the extent required by the software safety integrity level as defined in Figure 5.

11.4.8 Testing which is not fully documented and is performed by the designer prior to verification shall not be regarded as part of the verification.

11.4.9 The results of each verification shall be retained in a form defined or referenced in the Software Verification Plan such that it is auditable.

11.4.10 Après chaque activité de vérification, un rapport de vérification précisant que le logiciel a satisfait à la vérification ou précisant les raisons de son échec doit être produit. Les rapports de vérification doivent aborder les points suivants:

i) les éléments qui ne sont pas conformes à la Spécification des Exigences du Logiciel, à la Spécification de la Conception du Logiciel ou aux Spécifications de la Conception des Modules Logiciel;

ii) les éléments qui ne sont pas conformes au Plan d'Assurance Qualité du Logiciel;

iii) les modules, les données, les structures et les algorithmes mal adaptés au problème;

iv) les erreurs ou déficiences détectées;

v) l'identité et la configuration des éléments vérifiés.

11.4.11 Vérification des Exigences du Logiciel: une fois que la Spécification des Exigences du Logiciel a été définie, la vérification doit porter sur

i) l'adéquation de la Spécification des Exigences du Logiciel pour satisfaire aux exigences exposées dans la Spécification des Exigences du Système, la Spécification des Exigences de Sécurité du Système et le Plan d'Assurance Qualité du Logiciel;

ii) l'adéquation de la Spécification des Tests des Exigences du Logiciel comme tests de la Spécification des Exigences du Logiciel;

iii) la cohérence interne de la Spécification des Exigences du Logiciel.

Les résultats doivent être enregistrés dans un Rapport de Vérification des Exigences du Logiciel.

11.4.12 Vérification de l'architecture et de la conception du logiciel: une fois que la Spécification de l'Architecture du Logiciel et la Spécification de la Conception du Logiciel ont été définies, la vérification doit porter sur

i) l'adéquation de la Spécification de l'Architecture du Logiciel et de la Spécification de la Conception du Logiciel pour satisfaire à la Spécification des Exigences du Logiciel;

ii) l'adéquation de la Spécification de la Conception du Logiciel vis-à-vis de la Spécification des Exigences du Logiciel en ce qui concerne la cohérence et la complétude;

iii) l'adéquation du Plan de Tests d'Intégration du Logiciel comme ensemble de scénarios de tests pour la Spécification de l'Architecture du Logiciel et la Spécification de la Conception du Logiciel;

iv) la cohérence interne des Spécifications de l'Architecture et de la Conception du Logiciel.

Les résultats doivent être enregistrés dans un Rapport de Vérification de l'Architecture et de la Conception du Logiciel.

11.4.13 Vérification des modules logiciel: une fois que chaque Spécification de la Conception d'un Module Logiciel a été définie, la vérification doit porter sur

i) l'adéquation de la Spécification de la Conception d'un Module Logiciel pour satisfaire à la Spécification de la Conception du Logiciel;

ii) l'adéquation de la Spécification des Tests d'un Module Logiciel comme ensemble de scénarios de tests pour la Spécification de la Conception du Module Logiciel;

iii) la décomposition de la Spécification de la Conception du Logiciel en modules logiciels et les Spécifications de Conception des Modules Logiciel en faisant référence à

- la faisabilité de la performance requise,
- la testabilité pour une vérification ultérieure,
- la lisibilité par l'équipe de développement et de vérification, et
- la maintenabilité pour permettre une évolution ultérieure;

iv) l'adéquation du Rapport des Tests des Modules Logiciel comme enregistrement des tests effectués conformément aux Spécifications des Tests des Modules Logiciel.

11.4.10 After each verification activity a verification report shall be produced stating either that the software has passed the verification or the reasons for the failures. The verification reports shall address the following:

i) items which do not conform to the Software Requirements Specification, Software Design Specification or Software Module Design Specifications;

ii) items which do not conform to the Software Quality Assurance Plan;

iii) modules, data, structures and algorithms poorly adapted to the problem;

iv) detected errors or deficiencies;

v) the identity and configuration of the items verified.

11.4.11 Software Requirements Verification: once the Software Requirements Specification has been established, verification shall address

i) the adequacy of the Software Requirements Specification in fulfilling the requirements set out in the System Requirements Specification, the System Safety Requirements Specification and the Software Quality Assurance Plan;

ii) the adequacy of the Software Requirements Test Specification as a test of the Software Requirements Specification;

iii) the internal consistency of the Software Requirements Specification.

The results shall be recorded in a Software Requirements Verification Report.

11.4.12 Software Architecture and Design Verification: after the Software Architecture Specification and the Software Design Specification have been established, verification shall address

i) the adequacy of the Software Architecture Specification and the Software Design Specification in fulfilling the Software Requirements Specification;

ii) the adequacy of the Software Design Specification for the Software Requirements Specification with respect to consistency and completeness;

iii) the adequacy of the Software Integration Test Plan as a set of test cases for the Software Architecture Specification and the Software Design Specification;

iv) the internal consistency of the Software Architecture and Design Specifications.

The results shall be recorded in a Software Architecture and Design Verification Report.

11.4.13 Software Module Verification: after each Software Module Design Specification has been established, verification shall address

i) the adequacy of the Software Module Design Specification in fulfilling the Software Design Specification;

ii) the adequacy of the Software Module Test Specification as a set of test cases for the Software Module Design Specification;

iii) the decomposition of the Software Design Specification into software modules and the Software Module Design Specifications with reference to

   – feasibility of the performance required,

   – testability for further verification,

   – readability by the development and verification team, and

   – maintainability to permit further evolution;

iv) the adequacy of the Software Module Test Reports as a record of the tests carried out in accordance with the Software Module Test Specification.

Les résultats doivent être enregistrés dans un Rapport de Vérification des Modules Logiciel.

11.4.14 Vérification du code source du logiciel: dans la limite exigée par le niveau d'intégrité de la sécurité logicielle, le Code Source du Logiciel doit être vérifié pour assurer sa conformité aux Spécifications de la Conception des Modules Logiciel et au Plan d'Assurance Qualité du Logiciel. La vérification doit contenir un contrôle qui détermine si les normes de codage ont été correctement appliquées.

Les résultats doivent être enregistrés dans un Rapport de Vérification du Code Source du Logiciel.

11.4.15 Un Rapport des Tests d'Intégration du Logiciel doit être présenté comme suit:

i) un Rapport des Tests d'Intégration du Logiciel doit être présenté; il doit contenir les résultats des tests et préciser si les objectifs et les critères du Plan des Tests d'Intégration du Logiciel ont été respectés. En cas d'échec les raisons doivent être consignées dans le rapport;

ii) le Rapport des Tests d'Intégration du Logiciel doit être présenté dans un format permettant l'audit;

iii) les scénarios de tests et leurs résultats doivent être consignés, de préférence dans un format lisible par la machine en vue d'une analyse ultérieure;

iv) il convient que les tests soient reproductibles et exécutés par des moyens automatiques, dans la mesure du possible;

v) l'identité et la configuration des éléments doivent être vérifiées.

11.4.16 Pour l'intégration logiciel/matériel, voir 12.4.8.

12 Intégration logiciel/matériel

12.1 Objets

12.1.1 Démontrer que le logiciel et le matériel interagissent correctement pour exécuter les fonctions attendues.

12.1.2 Combiner le logiciel et le matériel, en vérifiant leur compatibilité, pour satisfaire à la Spécification des Exigences de Sécurité du Système et aux exigences relatives au niveau prévu d'intégrité de la sécurité logicielle.

12.2 Documents en entrée

1) Spécification des Exigences du Système
2) Spécification des Exigences de Sécurité du Système
3) Description de l'Architecture du Système
4) Spécification des Exigences du Logiciel
5) Spécification des Tests des Exigences du Logiciel
6) Spécification de l'Architecture du Logiciel
7) Spécification de la Conception du Logiciel
8) Spécifications de la Conception des Modules du Logiciel
9) Spécifications des Tests des Modules du Logiciel
10) Code Source du Logiciel et Documentation de Support
11) Documentation du Matériel

The results shall be recorded in a Software Module Verification Report.

11.4.14 Software Source Code Verification: to the extent demanded by the software safety integrity level, the Software Source Code shall be verified to ensure conformance to the Software Module Design Specification and the Software Quality Assurance Plan. This shall include a check to determine whether the coding standards have been applied correctly.

The results shall be recorded in a Software Source Code Verification Report.

11.4.15 A Software Integration Test Report shall be produced as follows:

i) a Software Integration Test Report shall be produced stating the test results and whether the objectives and criteria of the Software Integration Test Plan have been met. If there is a failure, the reasons for the failure shall be recorded;

ii) the Software Integration Test Report shall be in a form that is auditable;

iii) test cases and their results shall be recorded, preferably in machine readable form for subsequent analysis;

iv) tests should be repeatable and be performed by automatic means, if practicable;

v) the identity and configuration of the items verified.

11.4.16 For software/hardware integration, see 12.4.8.

12 Software/hardware integration

12.1 Objectives

12.1.1 To demonstrate that the software and the hardware interact correctly to perform their intended functions.

12.1.2 To combine the software and hardware, ensuring their compatibility, to meet the System Safety Requirements Specification and the requirements of the intended software safety integrity level.

12.2 Input documents

1) System Requirements Specification
2) System Safety Requirements Specification
3) System Architecture Description
4) Software Requirements Specification
5) Software Requirements Test Specification
6) Software Architecture Specification
7) Software Design Specification
8) Software Module Design Specification
9) Software Module Test Specification
10) Software Source code and supporting documentation
11) Hardware documentation

12.3 Documents en sortie

1) Plan de Tests d'Intégration Logiciel/Matériel
2) Rapport de Tests d'Intégration Logiciel/Matériel

12.4 Exigences

12.4.1 Pour les niveaux d'intégrité de la sécurité logicielle supérieurs à 0, un Plan de Tests d'Intégration Logiciel/Matériel sera créé tôt dans le cycle de vie du développement de manière à ce que les activités d'intégration puissent être convenablement menées et que les besoins particuliers en matière de conception ou autre intégration puissent être correctement assurés. Pendant le développement (et selon la taille du système), il est permis de diviser le plan en un certain nombre de sous-documents qui seront naturellement complétés à mesure que les conceptions de type matériel et logiciel évoluent et que les besoins détaillés d'intégration deviennent plus clairs.

12.4.2 Le Plan de Tests d'Intégration Logiciel/Matériel doit documenter les éléments suivants:

i) les scénarios de tests et les données de test;
ii) les types de tests à effectuer;
iii) l'environnement de tests y compris les outils, le logiciel de support et la description de la configuration;
iv) les critères de tests sur lesquels l'achèvement du test sera jugé.

12.4.3 Le Plan de Tests d'Intégration Logiciel/Matériel doit faire une distinction entre les activités qui peuvent être réalisées par le développeur dans ses locaux et celles exigeant l'accès au site de l'utilisateur.

12.4.4 Le Plan de Tests d'Intégration Logiciel/Matériel doit faire une distinction entre les activités suivantes:

i) l'implantation du logiciel sur le matériel cible;
ii) l'intégration du système.

12.4.5 Il convient que les outils et ressources identifiés dans le Plan de Tests d'Intégration Logiciel/Matériel soient disponibles dès que possible.

12.4.6 Au cours de l'intégration logiciel/matériel, toute modification ou changement apporté au système intégré doit faire l'objet d'une étude d'impact qui doit identifier tous les modules affectés et les activités indispensables à une nouvelle vérification.

12.4.7 Les scénarios de tests et leurs résultats doivent être consignés, de préférence dans un format lisible par la machine, en vue d'une analyse ultérieure.

12.4.8 Un Rapport de Tests d'Intégration Logiciel/Matériel doit être rédigé comme suit:

i) le Rapport de Tests d'Intégration Logiciel/Matériel doit contenir les résultats des tests et préciser si les objectifs et les critères du Plan de Tests d'Intégration Logiciel/Matériel ont été respectés. En cas d'échec les raisons doivent être consignées;
ii) le Rapport de Tests d'Intégration Logiciel/Matériel doit être présenté sous une forme permettant l'audit;
iii) les scénarios de tests et leurs résultats doivent être consignés, de préférence dans un format lisible par la machine, en vue d'une analyse ultérieure;

12.3 Output documents

1) Software/Hardware Integration Test Plan
2) Software/Hardware Integration Test Report

12.4 Requirements

12.4.1 For software safety integrity levels greater than zero, a Software/Hardware Integration Test Plan will be created early in the development life cycle, in order that integration activities may be properly directed and that particular design or other integration needs may be suitably provided for. During development (and depending upon the size of the system) the plan may be subdivided into a number of child documents and be naturally added to, as the hardware and software designs evolve and the detailed needs of integration become clearer.

12.4.2 The Software/Hardware Integration Test Plan shall document the following:

i) test cases and test data;
ii) types of tests to be performed;
iii) test environment including tools, support software and configuration description; and
iv) test criteria on which the completion of the test will be judged.

12.4.3 The Software/Hardware Integration Test Plan shall distinguish between those activities which can be carried out by the developer on his premises and those that require access to the user's site.

12.4.4 The Software/Hardware Integration Test Plan shall distinguish between the following activities:

i) merging of software onto the target hardware;
ii) system integration;

12.4.5 Tools and facilities identified in the Software/Hardware Integration Test Plan should be available at the earliest practicable time.

12.4.6 During Software/Hardware Integration any modification or change to the integrated system shall be subject to an impact study which shall identify all modules impacted and the necessary reverification activities.

12.4.7 Test cases and their results shall be recorded, preferably in machine readable form for subsequent analysis.

12.4.8 A Software/Hardware Integration Test Report shall be produced as follows:

i) the Software/Hardware Integration Test Report shall state the test results and whether the objectives and criteria of the Software/Hardware Integration Test Plan have been met. If there is a failure, it shall be recorded;
ii) the Software/Hardware Integration Test Report shall be in a form that is auditable;
iii) test cases and their results shall be recorded, preferably in a machine-readable form for subsequent analysis;

iv) le Rapport de Tests d'Intégration Logiciel/Matériel doit aussi contenir la preuve que le Chargé de Vérification est satisfait de l'adéquation du Rapport de Tests d'Intégration Logiciel/Matériel comme enregistrement prouvant que les tests ont été réalisés conformément au Plan de Tests d'Intégration Logiciel/Matériel;

v) le Rapport de Tests d'Intégration Logiciel/Matériel doit détailler l'identité et la configuration de tous les éléments concernés.

13 Validation du logiciel

13.1 Objets

Analyser et tester le système intégré pour assurer la conformité avec la Spécification des Exigences du Logiciel, en mettant tout particulièrement l'accent sur les aspects fonctionnels et ceux de sécurité selon le niveau d'intégrité de la sécurité logicielle.

13.2 Documents en entrée

1) Spécification des Exigences du Logiciel
2) Toute la documentation du matériel et du logiciel
3) Spécification des Exigences de Sécurité du Système

13.3 Documents en sortie

1) Plan de Validation du Logiciel
2) Rapport de Validation du Logiciel

13.4 Exigences

13.4.1 L'analyse et les tests doivent constituer la principale activité de validation.

13.4.2 Il est permis d'utiliser la simulation et la modélisation pour compléter le processus de validation.

13.4.3 Un Plan de Validation du Logiciel doit être établi et détaillé dans la documentation appropriée.

13.4.4 Le Plan de Validation du Logiciel doit être développé, exécuté et les résultats évalués par un parti indépendant dans la limite requise par le niveau d'intégrité de la sécurité logicielle.

13.4.5 Si cela est requis par le niveau d'intégrité de la sécurité logicielle, le domaine d'application et le contenu du Plan de Validation du Logiciel doivent être approuvés par le chargé d'évaluation. Cet accord doit également mentionner la présence du chargé d'évaluation pendant les tests.

13.4.6 Le Plan de Validation du Logiciel doit inclure un résumé justifiant le choix de la stratégie de validation. La justification doit inclure, selon le niveau requis d'intégrité de la sécurité logicielle, l'étude de

i) techniques manuelles et/ou automatisées;

ii) techniques statiques et/ou dynamiques;

iii) techniques analytiques et/ou statistiques.

iv) the Software/Hardware Integration Test Report shall also contain evidence that the Verifier is satisfied with the adequacy of the Software/Hardware Integration Test Report as a record of the tests carried out in accordance with the Software/Hardware Integration Test Plan;

v) the Software/Hardware Integration Test Report shall document the identity and configuration of all items involved.

13 Software validation

13.1 Objective

To analyse and test the integrated system to ensure compliance with the Software Requirements Specification with particular emphasis on the functional and safety aspects according to the Software Safety integrity level.

13.2 Input documents

1) Software Requirements Specification
2) All Hardware and Software Documentation
3) System Safety Requirements Specification

13.3 Output documents

1) Software Validation Plan
2) Software Validation Report

13.4 Requirements

13.4.1 Analysing and testing shall be the main validation activity.

13.4.2 Simulation and modelling may be used to supplement the validation process.

13.4.3 A Software Validation Plan shall be established and detailed in suitable documentation.

13.4.4 The Software Validation Plan shall be developed, performed and results evaluated by an independent party to the extent required by the software safety integrity level.

13.4.5 If required by the software safety integrity level, the scope and contents of the Software Validation Plan shall be agreed with the Assessor. This agreement shall also make a statement concerning the presence of the Assessor during testing.

13.4.6 The Software Validation Plan shall include a summary justifying the validation strategy chosen. The justification shall include consideration, according to the required software safety integrity level, of

i) manual or automated techniques or both;

ii) static or dynamic techniques or both;

iii) analytical or statistical techniques or both.

13.4.7 Le Plan de Validation du Logiciel doit identifier les étapes nécessaires pour démontrer l'adéquation

– de la Spécification des Exigences du Logiciel,

– de la Spécification de l'Architecture du Logiciel,

– de la Spécification de la Conception du Logiciel,

– des Spécifications de la Conception des Modules du Logiciel,

à satisfaire aux exigences de sécurité exposées dans la Spécification des Exigences de Sécurité du Système. Le chargé de validation doit contrôler que le processus de vérification est complet.

13.4.8 Le matériel de mesure utilisé pour la validation doit être étalonné de manière appropriée. Il faut démontrer que tous les outils, matériels ou logiciels utilisés pour la validation, sont adaptés aux objectifs.

13.4.9 Le logiciel doit être testé par la simulation des signaux d'entrée présents en fonctionnement normal, ou présents lors de situations prévues et de conditions non voulues exigeant une action du système.

13.4.10 Les résultats de la validation doivent être documentés dans le Rapport de Validation du Logiciel sous une forme permettant l'audit.

13.4.11 Une fois l'intégration matériel/logiciel terminée, un Rapport de Validation du Logiciel doit être rédigé comme suit:

i) il doit préciser si les objectifs et les critères du Plan de Validation du Logiciel ont été respectés. En cas d'échec, les raisons doivent être consignées;

ii) il doit contenir les résultats des tests et préciser si le logiciel complet sur sa machine cible remplit les exigences de la Spécification des Exigences du Logiciel;

iii) il doit contenir une évaluation de la couverture des exigences de la Spécification des Exigences du Logiciel, par les tests;

iv) le Rapport de Validation du Logiciel doit être présenté sous une forme permettant l'audit;

v) les scénarios de tests et leurs résultats doivent être consignés dans un format lisible par la machine, en vue d'une analyse ultérieure;

vi) il convient que les tests soient reproductibles et exécutés par des moyens automatiques, si cela est réalisable.

13.4.12 Le Rapport de Validation du Logiciel doit documenter l'identité et la configuration de tous les points ci-après:

i) le matériel et le logiciel utilisés;

ii) l'équipement utilisé;

iii) l'étalonnage de l'équipement;

iv) les modèles de simulation utilisés;

v) les divergences trouvées;

vi) les actions correctives réalisées.

13.4.13 Toute divergence découverte, y compris les erreurs détectées, doit être clairement identifiée dans une section distincte du Rapport de Validation du Logiciel et incluse dans tout document qui accompagne le logiciel délivré.

13.4.14 Le logiciel doit subir des tests par rapport à la Spécification des Tests des Exigences du Logiciel. Ces tests doivent montrer que toutes les exigences de la Spécification des Exigences du Logiciel sont correctement respectées.

13.4.7 The Software Validation Plan shall identify the steps necessary to demonstrate the adequacy of the

– Software Requirements Specification,
– Software Architecture Specification,
– Software Design Specification,
– Software Module Design Specification,

in fulfilling the safety requirements set out in the System Safety Requirements Specification. The Validator shall check that the verification process is complete.

13.4.8 Measurement equipment used for validation shall be calibrated appropriately. Any tools, hardware or software, used for validation shall be shown to be suitable for the purpose.

13.4.9 The software shall be exercised by simulation of input signals present during normal operation, anticipated occurrences and undesired conditions requiring system action.

13.4.10 The results of the validation shall be documented in the Software Validation Report in an auditable form.

13.4.11 Once hardware/software integration is finished, a Software Validation Report shall be produced as follows:

i) it shall state whether the objectives and criteria of the Software Validation Plan have been met. If there is a failure, it shall be recorded;
ii) it shall state the tests results and whether the whole software on its target machine fulfils the requirements set out in the Software Requirements Specification;
iii) an evaluation of the test coverage on the requirements of the Software Requirements Specification shall be provided;
iv) the Software Validation Report shall be in a form that is auditable;
v) test cases and their results shall be recorded in a machine readable form for subsequent analysis;
vi) tests should be repeatable and be performed by automatic means, if practicable.

13.4.12 The Software Validation Report shall document the identity and configuration of all of the following items:

i) the hardware and software used;
ii) the equipment used;
iii) the equipment's calibration;
iv) the simulation models used;
v) the discrepancies found;
vi) the corrective actions performed.

13.4.13 Any discrepancies found, including detected errors, shall be clearly identified in a separate section of the Software Validation Report and included in any release note which accompanies the delivered software.

13.4.14 The software shall be tested against the Software Requirements Test Specification. These tests shall show that all of the requirements in the Software Requirements Specification are correctly performed.

Les résultats doivent être enregistrés dans un Rapport de Validation du Logiciel.

14 Evaluation du logiciel

14.1 Objets

Evaluer que les processus du cycle de vie et les produits résultants sont tels que le logiciel correspond au niveau défini d'intégrité de la sécurité logicielle et qu'il est adapté à l'application prévue.

14.2 Documents en entrée

1) Spécification des Exigences de Sécurité du Système
2) Toute la documentation du matériel et du logiciel

14.3 Documents en sortie

Rapport d'Evaluation du Logiciel

14.4 Exigences

14.4.1 Pour les logiciels de niveau 0 d'intégrité de la sécurité:

i) il n'est exigé du chargé d'évaluation que de confirmer qu'il s'agit du niveau d'intégrité de la sécurité logicielle approprié;
ii) il est également possible que cette confirmation du niveau soit convenue entre le fournisseur et l'utilisateur au moment de la soumission.

14.4.2 Un logiciel accompagné d'un Rapport d'Evaluation du Logiciel, émanant d'un autre chargé d'évaluation, n'a pas à faire l'objet d'une évaluation entièrement nouvelle. Le deuxième chargé d'évaluation doit vérifier que le logiciel a le niveau requis d'intégrité de la sécurité logicielle et qu'il est adapté à l'application prévue sur l'ordinateur cible qui lui est destiné.

14.4.3 Le chargé d'évaluation doit avoir accès au processus de conception et de développement et à l'ensemble de la documentation associée au projet.

14.4.4 L'évaluation du logiciel doit être menée à bien par un chargé d'évaluation qui est indépendant de l'équipe de conception.

14.4.5 Le chargé d'évaluation doit évaluer si le logiciel du système est adapté à l'objectif prévu et s'il répond correctement aux problèmes de sécurité dérivés de la Spécification des Exigences de Sécurité du Système.

14.4.6 Dans la limite requise par le niveau d'intégrité de la sécurité logicielle, le chargé d'évaluation doit décider si les méthodes appropriées ont été sélectionnées et appliquées à chaque phase du cycle de vie du logiciel.

14.4.7 Si cela est requis par le niveau d'intégrité de la sécurité logicielle, le chargé d'évaluation doit être d'accord avec le domaine d'application et avec le contenu du Plan de Validation du Logiciel. Cet accord doit également mentionner la présence du chargé d'évaluation pendant les tests.

14.4.8 Le chargé d'évaluation peut demander un travail de vérification et de validation supplémentaire, s'il le désire.

The results shall be recorded in a Software Validation Report.

14 Software assessment

14.1 Objective

To evaluate that the life cycle processes and products resulting are such that the software is of the defined software safety integrity level and is fit for its intended application.

14.2 Input documents

1) System Safety Requirements Specification
2) All Hardware and Software Documentation

14.3 Output documents

Software Assessment Report

14.4 Requirements

14.4.1 For software of safety integrity level zero:

i) the Assessor shall only be required to confirm that this is the appropriate software safety integrity level;
ii) it is also possible to agree this level between the supplier and the user at the time of tendering.

14.4.2 Software with a Software Assessment Report from another Assessor does not have to be an object for an entirely new assessment. The second Assessor shall check that the software is of the required software safety integrity level and that it is fit for its intended application on the intended target computer.

14.4.3 The Assessor shall have access to the design and development process and all project-related documentation.

14.4.4 The assessment of the software shall be carried out by an Assessor who is independent from the design team.

14.4.5 The Assessor shall assess that the software of the system is fit for its intended purpose and responds correctly to safety issues derived from the System Safety Requirements Specification.

14.4.6 To the extent required by the software safety integrity level, the Assessor shall decide if appropriate methods have been selected and applied at each phase of the software life cycle.

14.4.7 If required by the software safety integrity level, the Assessor shall agree the scope and contents of the Software Validation Plan. This agreement shall also make a statement concerning the presence of the Assessor during testing.

14.4.8 The Assessor may ask for additional verification and validation work if he so chooses.

14.4.9 Pour chaque examen, le chargé d'évaluation doit produire un rapport qui doit détailler ses résultats d'évaluation.

14.4.10 Si, de l'avis du chargé d'évaluation, le logiciel est adapté à l'application prévue, le Rapport définitif d'Evaluation du Logiciel doit inclure un compte rendu précisant le niveau d'intégrité de la sécurité logicielle atteint par le logiciel.

14.4.11 Si le logiciel n'est pas adapté à l'application prévue ou s'il n'atteint pas le niveau requis d'intégration de la sécurité logicielle, le chargé d'évaluation doit alors faire exclusivement état des éléments de non-conformité dans le Rapport d'Evaluation du Logiciel et il ne doit donner aucune solution technique.

15 Assurance qualité du logiciel

15.1 Objets

15.1.1 Identifier, superviser et contrôler toutes les activités, à la fois techniques et de gestion, qui sont nécessaires pour assurer que le logiciel possède la qualité requise. Il est nécessaire de fournir la démarche qualitative requise contre les défauts systématiques et de s'assurer qu'une trace puisse être laissée afin de permettre un déroulement efficace des activités de vérification et de validation.

15.1.2 Apporter la preuve que toutes les activités ci-dessus ont été menées à bien.

15.2 Documents en entrée

Tous les documents disponibles à chaque étape du cycle de vie

15.3 Documents en sortie

1) Plan d'Assurance Qualité du Logiciel
2) Plan de Gestion de la Configuration du Logiciel

Tous les plans ci-dessus doivent être rédigés au commencement du projet et mis à jour pendant le cycle de vie.

15.4 Exigences

15.4.1 Le fournisseur et/ou le développeur doivent posséder et utiliser, pour le moins, un Système d'assurance qualité conforme à l'ISO 9000:2000 afin de prendre en charge les exigences de la présente norme. L'habilitation ISO 9001 est hautement recommandée.

15.4.2 Pour le moins, le fournisseur et/ou le développeur et le client doivent mettre en oeuvre les parties appropriées de l'ISO 9001 dans le cadre du développement du logiciel conformément aux directives contenues dans l'ISO 9000-3.

15.4.3 Le fournisseur et/ou le développeur doivent préparer et documenter, projet par projet, un Plan d'Assurance Qualité du Logiciel afin de mettre en application les exigences de 15.4.1 et 15.4.2 de la présente norme, qui doivent, chaque fois que possible, être exprimées en termes mesurables.

15.4.4 Le Plan d'Assurance Qualité du Logiciel doit comporter un paragraphe spécifiant les détails relatifs à sa propre mise à jour tout au long du projet: fréquence, responsabilité, méthode.

14.4.9 The Assessor shall produce a report for each review which shall detail his assessment results.

14.4.10 If, in the opinion of the Assessor, the software is fit for its intended application, the final Software Assessment Report shall include a statement as to the software safety integrity level achieved by the software.

14.4.11 When the software is not fit for its purpose or has not achieved the required software safety integrity level then the Assessor shall only report the non conformities in the Software Assessment Report and shall not give any technical solution.

15 Software quality assurance

15.1 Objectives

15.1.1 To identify, monitor and control all those activities, both technical and managerial, which are necessary to ensure that the software achieves the quality required. This is necessary to provide the required qualitative defence against systematic faults and to ensure that an audit trail can be established to allow verification and validation activities to be undertaken effectively.

15.1.2 To provide evidence that the above activities have been carried out.

15.2 Input documents

All the documents available at each stage of the life cycle

15.3 Output documents

1) Software Quality Assurance Plan
2) Software Configuration Management Plan

All the above plans shall be issued at the beginning of the project and updated during the life cycle.

15.4 Requirements

15.4.1 The supplier and/or developer shall have and use as a minimum a Quality Assurance System compliant with the ISO 9000:2001, to support the requirements of this standard. ISO 9001 accreditation is highly recommended.

15.4.2 As a minimum, the supplier and/or developer and the customer shall implement for the software development the relevant parts of ISO 9001, in accordance with the guidelines contained in ISO 9000-3.

15.4.3 The supplier and/or developer shall prepare and document, on a project-by-project basis, a Software Quality Assurance Plan to implement the requirements of 15.4.1 and 15.4.2 of this standard, which shall be expressed in measurable terms wherever possible.

15.4.4 The Software Quality Assurance Plan shall have a paragraph specifying details about its own updating throughout the project: frequency, responsibility, method.

15.4.5 Toutes les activités, actions, documents, etc. requis par toutes les sections de l'ISO 9000-3 et de la présente norme (annexe A incluse) doivent être spécifiés ou référencés dans le Plan d'Assurance Qualité du Logiciel et adaptés en fonction du développement spécifique. Aucune des listes contenues dans l'ISO 9000-3 ne doit être présumée exhaustive.

Pour le moins, sauf pour une intégrité de la sécurité logicielle de niveau 0, les éléments suivants doivent également être spécifiés ou référencés dans le Plan d'Assurance Qualité du Logiciel.

Ceci permet d'assurer la couverture intégrale de tous les aspects de sécurité du logiciel par rapport au niveau requis d'intégrité de la sécurité logicielle.

La présente liste n'est pas exhaustive:

i) définition du modèle du cycle de vie

   définition de chaque phase y compris:

   – activités et tâches élémentaires;
   – critères d'entrée et de sortie;
   – entrées et sorties de chaque phase;
   – principales activités qualité dans chaque phase;
   – unité organisationnelle responsable de chaque activité et tâche élémentaire;

ii) traçabilité des exigences;

iii) traçabilité de la structure de la documentation;

iv) documentation associée au développement, à la vérification et à la validation, à l'utilisation et à la maintenance du logiciel;

v) procédures d'intégration du système;

vi) normes de codage à utiliser;

vii) évaluation des tests de validation précédents;

viii) la définition de métriques (mesures quantitatives) à mesurer à la fois sur le produit et sur le processus. Pour les métriques du produit logiciel, il doit être fait référence aux caractéristiques de la qualité et aux directives d'évaluation définies par la série de normes ISO/CEI 9126.

15.4.6 Pour le moins, la gestion de la configuration doit être réalisée conformément aux directives contenues dans l'ISO 9000-3.

Chaque document du logiciel doit être soumis au contrôle de configuration avant la diffusion de sa première version approuvée. Le code source du logiciel doit être soumis au contrôle de configuration avant le commencement des tests documentés des modules.

Il ne doit pas être possible d'effectuer des changements non autorisés sur tout élément sous contrôle de gestion de configuration. Des précautions doivent être prises pour empêcher ou détecter des erreurs survenant dans un code lisible par une machine pendant le stockage, le transfert, la transmission ou la duplication.

La gestion de la configuration ne doit pas se limiter strictement au développement et à la maintenance du produit mais elle doit aussi englober l'environnement utilisé pendant tout le cycle de vie. Cette extension, nécessaire pour la reproductibilité du développement et pour les activités de maintenance, doit concerner les fichiers de configuration informatique, les assembleurs, les compilateurs, les programmes de mise au point et tous les autres outils utilisés.

15.4.7 L'adéquation et les résultats des Plans de Vérification du Logiciel doivent être examinés.

15.4.5 All activities, actions, documents, etc. required by all the sections of ISO 9000-3 and of this standard (annex A included) shall be specified or referenced in the Software Quality Assurance Plan and tailored to the specific development. None of the lists in ISO 9000-3 shall be presumed to be exhaustive.

As a minimum, except at software safety integrity level zero, the following items shall also be specified or referenced in the Software Quality Assurance Plan.

This is to ensure that all the safety aspects in the software with respect to the required Software Safety integrity level will be covered.

The present list is not exhaustive:

i) definition of the life cycle model

   definition of each phase including:

   – activities and elementary tasks;
   – entry and exit criteria;
   – inputs and outputs of each phase;
   – major quality activities in each phase;
   – organisational unit responsible for each activity and elementary task;

ii) requirements traceability;

iii) documentation structure traceability;

iv) documentation associated with the development, verification and validation, operation and maintenance of software;

v) system integration procedures;

vi) coding standards to be used;

vii) assessment of previous validation tests;

viii) the definition of the metrics (quantitative measures) to be carried out on both the product and the process. For the software product metrics carried out, reference shall be made to the quality characteristics and evaluation guidelines defined by the ISO/IEC 9126 series.

15.4.6 As a minimum, configuration management shall be carried out in accordance with the guidelines contained in ISO 9000-3.

Each software document shall be placed under configuration control before the release of its first approved version. The software source code shall be placed under configuration control before the commencement of documented module testing.

It shall not be possible to make any unauthorised changes to any item under Configuration Management Control. Precautions shall be taken to prevent or detect errors occurring in machine-readable code during storage, transfer, transmission or duplication.

Configuration Management shall not be limited to the strict product development and maintenance, but it shall also cover the environment used during the full life cycle. This extension, necessary for the reproducibility of the development and for the maintenance activities, shall include computer configuration files, assemblers, compilers, debuggers and all the other tools used.

15.4.7 The adequacy and results of Software Verification Plans shall be examined.

15.4.8 Le fournisseur et/ou le développeur doit établir, documenter et maintenir des procédures pour le contrôle des fournisseurs externes, notamment:

– des méthodes visant à assurer que les logiciels fournis par des fournisseurs externes respectent les exigences définies. Les logiciels développés préalablement doivent être garantis conformes au niveau requis d'intégrité de la sécurité logicielle et de sûreté de fonctionnement. Les nouveaux logiciels doivent être développés et maintenus en conformité avec le Plan d'Assurance Qualité du Logiciel du fournisseur ou avec un Plan d'Assurance Qualité du Logiciel spécifique, préparé par le fournisseur externe en accord avec le Plan d'Assurance Qualité Logiciel du fournisseur;

– des méthodes visant à assurer que les exigences transmises au fournisseur de logiciel externe sont pertinentes et complètes.

15.4.9 Le fournisseur et/ou le développeur doit établir, documenter et mettre à jour des procédures pour le compte rendu des problèmes et les actions correctives. Ces procédures, comme partie intégrante du Système d'assurance qualité, doivent mettre en oeuvre les parties pertinentes de l'ISO 9001, notamment pour couvrir au minimum les aspects suivants:

– définir la documentation nécessaire pour le compte rendu des problèmes et/ou les actions correctives, dans le but de permettre le retour d'information vers la direction responsable;

– définir l'analyse des informations recueillies dans les comptes rendus de problèmes pour en identifier les causes;

– définir les pratiques à suivre pour le compte rendu, le suivi et la résolution des problèmes identifiés à la fois au cours de la phase de développement et au cours de la phase de maintenance du logiciel;

– définir des actions préventives pour traiter les problèmes à un niveau correspondant au niveau requis d'intégrité de la sécurité logicielle;

– définir les responsabilités organisationnelles spécifiques par rapport au développement et à la maintenance du logiciel;

– définir la manière d'appliquer des contrôles pour s'assurer que des actions correctives sont prises et qu'elles sont efficaces;

– définir les formats à utiliser;

– définir les exigences pour répéter les tests, la vérification, la validation et l'évaluation.

Pour le moins, le compte rendu des problèmes et la gestion des actions correctives doivent être mis en place dans le cycle de vie du logiciel, immédiatement après l'intégration du logiciel et avant le début de la validation formelle du logiciel, en couvrant aussi toute la phase de maintenance du logiciel.

16 Maintenance du logiciel

16.1 Objets

Assurer que le logiciel fonctionne comme requis, en préservant le niveau d'intégrité de la sécurité logicielle et la sûreté de fonctionnement lorsqu'on procède à des corrections, des améliorations ou des adaptations sur le logiciel lui-même.

16.2 Documents en entrée

Tous documents

16.3 Documents en sortie

1) Plan de Maintenance du Logiciel
2) Enregistrements des Evolutions du Logiciel
3) Enregistrement de la Maintenance du Logiciel

15.4.8 The supplier and/or developer shall establish, document and maintain procedures for External Supplier Control, including:

– methods to ensure that software provided by external suppliers adheres to established requirements. Previously developed software shall be assured to be compliant with the required software safety integrity level and dependability. New software shall be developed and maintained in conformity with the Software Quality Assurance Plan of the Supplier or with a specific Software Quality Assurance Plan prepared by the external supplier in accordance with the Software Quality Assurance Plan of the Supplier;

– methods to ensure that the requirements provided to the External Software Supplier are adequate and complete.

15.4.9 The supplier and/or developer shall establish, document and maintain procedures for Problem Reporting and Corrective Actions. These procedures, as part of the Quality Assurance System, shall implement the relevant parts of ISO 9001, especially covering the following aspects at least:

– define the documentation needed for problem reporting and/or corrective actions, with the aim of giving feedback to the responsible management;

– define analysis of the information collected in the problem reports to identify its causes;

– define the practices to be followed for reporting, tracking and resolving problems identified both during the development phase and during software maintenance;

– define preventive actions to deal with problems to a level corresponding to the required software safety integrity level;

– define the specific organisational responsibilities with regard to development and software maintenance;

– define how to apply controls to ensure that corrective actions are taken and that they are effective;

– define the forms to be used;

– define the requirements for re-test, re-verification, re-validation and re-assessment.

As a minimum, problem reporting and corrective action management shall be applied in the software life cycle starting immediately after Software Integration and before the starting of formal Software Validation, also covering the whole phase of Software Maintenance.

16 Software maintenance

16.1 Objective

To ensure that the software performs as required, preserving the required software safety integrity level and dependability when making corrections, enhancements or adaptations to the software itself.

16.2 Input documents

All documents

16.3 Output documents

1) Software Maintenance Plan
2) Software Change Records
3) Software Maintenance Record

16.4 Exigences

16.4.1 Pour le moins, la maintenance doit être réalisée conformément aux directives exposées dans l'ISO 9000-3.

En outre, les exigences suivantes, applicables à la maintenance du logiciel, doivent également être respectées.

16.4.2 La maintenabilité doit être intégrée dans le système logiciel, en particulier, en respectant les exigences de l'article 10 de la présente norme. Il convient également d'employer la série de normes ISO/CEI 9126 pour exiger et vérifier le niveau minimum de maintenabilité.

16.4.3 Les procédures de maintenance du logiciel doivent être définies et enregistrées dans le Plan de maintenance du logiciel. Ces procédures doivent également inclure

i) le contrôle du compte rendu des erreurs, des journaux des erreurs, des enregistrements de maintenance, de l'autorisation de modification et de la configuration logiciel/système;

ii) la vérification, la validation et l'évaluation; et

iii) la définition de l'autorité qui approuve le logiciel modifié.

16.4.4 Les activités de maintenance doivent être auditées par rapport au Plan de Maintenance du Logiciel, à des intervalles définis dans le Plan d'Assurance Qualité du Logiciel.

16.4.5 La maintenance doit être réalisée avec le même niveau de compétence, d'outils, de documentation, de planification et de gestion, que le développement initial du système. Ceci doit s'appliquer également à la gestion de la configuration, au contrôle des évolutions, au contrôle des documents et à l'indépendance des parties impliquées.

16.4.6 La présente norme n'est pas destinée à être rétroactive. Elle s'applique donc principalement aux nouveaux développements et n'est applicable dans son intégralité aux systèmes existants que s'ils font l'objet de modifications importantes. Pour ce qui concerne les niveaux 3 ou 4 de l'intégrité de la sécurité logicielle, les entités contractantes doivent, avant de commencer tout travail de modification, décider si les actions de maintenance doivent être considérées comme majeures ou mineures ou si les méthodes existantes de maintenance du système sont appropriées. Pour ce qui concerne les niveaux 0, 1 ou 2 de l'intégrité de la sécurité logicielle, cette même décision doit être prise par le fournisseur.

16.4.7 Le contrôle des fournisseurs externes, le compte rendu des problèmes et les actions correctives doivent être gérés en fonction des mêmes critères que ceux spécifiés dans les paragraphes applicables de l'article consacré à l'assurance qualité du logiciel.

16.4.8 Un Enregistrement de la Maintenance du Logiciel doit être établi pour chaque logiciel avant sa première diffusion et doit être mis à jour. Outre les exigences de l'ISO 9000-3 concernant «Les enregistrements et les rapports de maintenance», cet Enregistrement doit également inclure

i) des références à tous les Enregistrements des Evolutions du Logiciel applicables à ce logiciel;

ii) des informations sur les conséquences de ces modifications;

iii) des scénarios de tests pour les composants, incluant les données de tests pour la revalidation et la régression; et

iv) l'historique de la configuration logicielle.

16.4 Requirements

16.4.1 As a minimum, maintenance shall be carried out in accordance with the guidelines contained in ISO 9000-3.

In addition, the following requirements concerning software maintenance shall also be met.

16.4.2 Maintainability shall be designed into the software system, in particular, by following the requirements of clause 10 of this standard. The ISO/IEC 9126 series should also be employed in order to require and verify a minimum level of maintainability.

16.4.3 Procedures for the maintenance of software shall be established and recorded in the Software Maintenance Plan. These procedures shall also include

i) control of error reporting, error logs, maintenance records, change authorisation and software/system configuration;

ii) verification, validation and assessment; and

iii) definition of the Authority which approves the changed software.

16.4.4 The maintenance activities shall be audited against the Software Maintenance Plan, at intervals defined in the Software Quality Assurance Plan.

16.4.5 Maintenance shall be performed with the same level of expertise, tools, documentation, planning and management as the initial development of the system. This shall apply also to configuration management, change control, document control, and independence of involved parties.

16.4.6 This standard is not intended to be retrospective. It therefore applies primarily to new developments and only applies in its entirety to existing systems if these are subjected to major modifications. For software safety integrity level 3 or 4, the contracting entities shall, before starting work on any change, decide whether the maintenance actions are to be considered as major or minor or whether the existing maintenance methods for the system are adequate. For software safety integrity levels 0, 1 or 2, the same decision shall be taken by the supplier.

16.4.7 External supplier control, problem reporting and corrective actions shall be managed with the same criteria specified in the relevant paragraphs of the Software Quality Assurance clause.

16.4.8 A Software Maintenance Record shall be established for each Software Item before its first release, and it shall be maintained. In addition to the requirements of ISO 9000-3 for "Maintenance Records and Reports", this Record shall also include

i) references to all the Software Change Records for that Software Item;

ii) change consequence information;

iii) test cases for components, including revalidation and regression testing data; and

iv) software configuration history.

16.4.9 Un Enregistrement des Evolutions du Logiciel doit être établi pour chaque activité de maintenance. Cet enregistrement doit inclure

i) la demande de modification ou de changement;

ii) une analyse de l'impact de l'activité de maintenance sur le système global, y compris l'interaction matérielle, logicielle et humaine ainsi que des interactions éventuelles avec l'environnement;

iii) la spécification détaillée de la modification ou du changement; et

iv) la revalidation, les tests de non régression et la réévaluation de la modification ou du changement dans la limite requise par le niveau d'intégrité de la sécurité logicielle. La responsabilité en matière de revalidation peut varier d'un projet à l'autre, selon le niveau d'intégrité de la sécurité logicielle. L'impact de la modification ou du changement sur le processus de revalidation peut aussi se limiter à différents niveaux du système (seulement les modules modifiés, tous les modules affectés identifiés, le système complet). Le Plan de Validation du Logiciel doit donc aborder les deux problèmes, selon le niveau d'intégrité de la sécurité logicielle. Le degré d'indépendance de la revalidation doit être le même que celui de la validation.

17 Systèmes configurés par des données d'application

17.1 Objets

17.1.1 Un trait caractéristique des systèmes de commande et de protection ferroviaire est la nécessité de concevoir chaque installation en vue de répondre aux exigences individuelles d'une application spécifique. Un système configuré par des données d'application, permet l'utilisation d'un logiciel générique avec homologation de types, complété des exigences individuelles pour chaque installation, fournies sous la forme de données (données spécifiques d'application). Ces données se présentent, en principe, sous forme d'une information tabulaire ou d'un langage spécifique d'application qui est interprété par le logiciel générique.

17.1.2 Pour un système critique de sécurité, le haut niveau de ressources requis pour développer un logiciel atteignant le niveau exigé d'intégrité de la sécurité du système, rend très attractive l'adoption d'un système configuré par des données d'application, car elle permet la réutilisation du logiciel générique. Toutefois, comme il est probable que la sécurité du fonctionnement du système soit tributaire de l'exactitude des données, il est impératif que les procédures utilisées pour le développement des données possèdent également le niveau approprié d'intégrité de la sécurité du système.

17.1.3 Les paragraphes ci-dessous décrivent les exigences de la présente norme applicables au développement initial du logiciel générique pour un système configuré par des données d'application, et au développement ultérieur de chaque ensemble de données propres à l'installation.

17.2 Documents en entrée

1) Spécification des Exigences du Logiciel
2) Spécification de l'Architecture du Logiciel

17.3 Documents en sortie

1) Définitions des Exigences de l'Installation
2) Plan de Préparation des Données
3) Plan de Tests des Données
4) Rapport de Tests des Données

16.4.9 A Software Change Record shall be established for each maintenance activity. This record shall include

i) the modification or change request;

ii) an analysis of the impact of the maintenance activity on the overall system, including hardware, software, human interaction and the environment and possible interactions;

iii) the detailed specification of the modification or change; and

iv) revalidation, regression testing and re-assessment of the modification or change to the extent required by the software safety integrity level. The responsibility for revalidation can vary from project to project, according to the software safety integrity level. Also the impact of the modification or change on the process of revalidation can be confined to different system levels (only changed modules, all identified affected modules, the complete system). Therefore the Software Validation Plan shall address both problems, according to the software safety integrity level. The degree of independence of revalidation shall be the same as that for validation.

17 Systems configured by application data

17.1 Objectives

17.1.1 A characteristic feature of railway control and protection systems is the need to design each installation to meet the individual requirements for a specific application. A system configured by application data allows type-approved generic software to be used, with the individual requirements for each installation defined as data (application specific data). This data is normally in the form of tabular information or an application specific language which is interpreted by the generic software.

17.1.2 For a safety critical system, the high level of resources required to develop software to achieve the required system safety integrity level for the system makes the adoption of a system configured by application data very attractive because it allows the re-use of generic software. However, as the safe operation of the system is likely to depend on the correctness of the data, the procedures used for development of the data must also be to an appropriate system safety integrity level.

17.1.3 The subclauses below describe the requirements of this standard for the initial development of the generic software for a system configured by application data, and for the subsequent development of each set of installation-specific data.

17.2 Input documents

1) Software Requirements Specification
2) Software Architecture Specification

17.3 Output documents

1) Application Requirements Specification
2) Data Preparation Plan
3) Data Test Plan
4) Data Test Report

17.4 Exigences

17.4.1 Cycle de vie de la préparation des données

La Figure 6 illustre le cycle de vie d'une application utilisant un matériel et un logiciel générique, ainsi que des données propres à l'application. Les phases du cycle de vie sont les suivantes.

17.4.1.1 Spécification des exigences de l'application

Les exigences de l'application doivent être définies. Ceci doit inclure les exigences qui sont propres à l'installation individuelle (par exemple le tracé de la voie, la localisation des signaux, les limites de vitesse) et les normes auxquelles l'application doit se conformer (par exemple, principes de signalisation, niveaux d'intégrité de la sécurité du système).

17.4.1.2 Conception globale de l'installation

L'architecture du système doit être définie, et la quantité et le type des composants génériques à utiliser doivent être spécifiés. Les composants qui contiennent des éléments logiciels doivent avoir été développés conformément à la présente norme. Les fonctions spécifiées dans les exigences doivent être affectées aux composants et l'implantation physique de chaque composant doit être définie.

17.4.1.3 Préparation de données

Le processus de préparation des données doit inclure la production d'informations spécifiques (par exemple, des tableaux de contrôle), la production du code source des données et sa compilation, le contrôle et autres activités de vérification et les tests des données d'application.

17.4.1.4 Intégration et acceptation

Dans certains systèmes, les données d'application seront intégrées dans le matériel et le logiciel génériques pour un essai en usine avant l'installation sur site. Dans les cas où le niveau de confiance peut être obtenu par d'autres moyens, il n'est pas exclu de supprimer cette démarche. L'équipement doit alors être installé sur site et les tests d'intégration doivent être effectués sur le nouvel équipement. Enfin, le système doit être mis en service comme système pleinement opérationnel et un processus d'acceptation finale doit être réalisé sur l'installation complète.

17.4.1.5 Validation et évaluation

Les activités de validation et d'évaluation doivent contrôler la performance de chaque étape du cycle de vie.

17.4.2 Procédures et outils de préparation des données

Pour chaque nouveau type de système configuré par des données d'application, des procédures et outils spécifiques de préparation des données doivent être développés pour permettre d'appliquer le cycle de vie de la préparation des données, spécifié en 17.4.1, aux installations du nouveau système. Le développement de ces procédures et outils doit être réalisé selon la présente norme en parallèle avec le matériel et le logiciel génériques du système. Les activités de vérification, de validation et d'évaluation doivent assurer que les outils de préparation des données et le logiciel générique sont compatibles.

17.4 Requirements

17.4.1 Data Preparation Life Cycle

Figure 6 illustrates a life cycle for an application making use of hardware and generic software, together with application-specific data. The phases of the life cycle are as follows.

17.4.1.1 Application Requirements Specification

The requirements for the application shall be defined. This shall include requirements which are specific to the individual installation (e.g. track layout, signal locations, speed limits), and standards with which the application must comply (e.g. signalling principles, system safety integrity levels).

17.4.1.2 Overall Installation Design

The system architecture shall be defined, and the quantity and type of the generic components to be used shall be specified. Those components which contain software shall have been developed in conformance with this standard. The functions specified in the requirements shall be allocated to the components, and the physical location of each component shall be defined.

17.4.1.3 Data Preparation

The data preparation process shall include the production of specific information (e.g. control tables), production of the data source code and its compilation, checking and other verification activities, and testing of the application data.

17.4.1.4 Integration and Acceptance

For some systems the application data will be integrated with the generic hardware and software for a factory test before installation on site. This may not be necessary where a sufficient degree of confidence can be obtained by other means. The equipment shall then be installed on site, and integration tests carried out on the new equipment. Finally the system shall be commissioned as a fully operational system, and a final acceptance process shall be carried out on the complete installation.

17.4.1.5 Validation and Assessment

Validation and assessment activities shall audit the performance of each stage of the life cycle.

17.4.2 Data Preparation Procedures and Tools

For each new type of system configured by application data, specific data preparation procedures and tools shall be developed to allow the data preparation life cycle specified in 17.4.1 to be applied to installations of the new system. Development of these procedures and tools shall be carried out in accordance with this standard in parallel with the generic software and hardware for the system. The verification, validation and assessment activities shall ensure that the data preparation tools and the generic software are compatible.

17.4.2.1 Lors de la phase de conception logicielle du système configuré par des données d'application, un Plan de Préparation des Données doit être présenté en vue de définir une structure de documentation applicable au processus de préparation des données. Ces documents doivent être associés au modèle du cycle de vie de la préparation des données décrit en 17.4.1. Le plan doit spécifier les procédures et outils de préparation des données à utiliser pour satisfaire aux niveaux requis d'intégrité de la sécurité du système. Le plan doit spécifier les exigences d'indépendance entre les équipes réalisant les tâches de vérification, de validation et de conception.

17.4.2.2 Le Plan de Préparation des Données doit affecter un niveau d'intégrité de la sécurité à tout outil matériel ou logiciel utilisé dans le cycle de vie de la préparation des données. Ce niveau d'intégrité de la sécurité doit être dérivé du niveau requis d'intégrité de la sécurité du système et du degré de contrôle de la sortie de chaque outil, effectué au moyen d'autres procédures manuelles ou automatisées.

17.4.2.3 Chaque fois que possible, le Plan de Préparation des Données doit utiliser, pour spécifier les exigences et la conception, des notations qui sont familières aux ingénieurs d'applications, par exemple des plans de signalisation standards et des tableaux de contrôle. Lorsque de nouvelles notations sont introduites, il est indispensable de fournir la documentation utilisateur nécessaire et, le cas échéant, la formation doit être assurée.

17.4.2.4 Les rapports de vérification, de tests, de validation et d'évaluation indispensables pour démontrer que la préparation des données a été exécutée conformément au plan, peuvent être standardisés sous forme de listes de contrôle, afin de minimiser la charge de travail impliquée dans la production de la documentation pour chaque installation. Cette information doit être contenue dans le Plan de Tests des Données et les résultats enregistrés dans le Rapport de Tests des Données.

17.4.2.5 Toutes les données et leur documentation associée doivent être conformes aux exigences de gestion de la configuration définies à l'article 15 de la présente norme. Les enregistrements de la gestion de la configuration doivent mentionner la version du logiciel générique avec lequel les données ont été conçues pour fonctionner et les versions des outils utilisés dans le processus de préparation des données.

17.4.3 Développement du logiciel

Le développement du logiciel générique à utiliser dans un système configuré par les données d'application, doit être conforme aux exigences définies aux articles 1 à 16 de la présente norme. Les exigences supplémentaires suivantes doivent également être respectées:

17.4.3.1 Au cours de la phase de Spécification des Exigences du Logiciel, il est indispensable d'identifier les fonctions utilisant les données d'application dans chaque système et sous-système. Le niveau d'intégrité de la sécurité du système affecté à chaque sous-système déterminera les normes à appliquer au développement ultérieur des données pour toutes les installations du système.

17.4.3.2 Au cours de la phase de conception du logiciel, les interfaces détaillées entre le logiciel générique et les données d'application doivent être spécifiées, à moins qu'elles n'aient déjà été spécifiées lors d'une phase précédente du cycle de vie, par exemple à cause d'une exigence d'utilisation d'un langage d'application spécifique existant.

17.4.3.3 Au cours de la phase de conception des modules logiciel, il doit être mis en pratique une séparation rigide entre le code du programme et les données, autrement dit il doit être possible de recompiler et de mettre à jour, soit le logiciel générique, soit les données, sans avoir à mettre l'autre à jour, à moins qu'une modification n'ait été apportée dans l'interface définie entre le logiciel et les données. De la même façon, il convient de séparer les données propres à l'application des autres données.

17.4.2.1 At the Software Design phase for the system configured by application data, a Data Preparation Plan shall be produced to define a documentation structure for the data preparation process. These documents shall be related to the data preparation life cycle model described in 17.4.1. The plan shall specify the data preparation procedures and tools to be used to meet the required system safety integrity levels. The plan shall specify the requirements for the independence between staff carrying out verification, validation and design tasks.

17.4.2.2 The Data Preparation Plan shall allocate a safety integrity level to any hardware or software tools used in the data preparation life cycle. This safety integrity level shall be derived from the required system safety integrity level and the degree to which the output of each tool is checked by other manual or automated procedures.

17.4.2.3 Where possible the Data Preparation Plan shall call for notations for specifying requirements and design which are familiar to applications engineers, for example, standard signalling plans and control tables. Where new notations are introduced, the necessary user documentation must be provided and training shall also be provided where appropriate.

17.4.2.4 The verification, test, validation and assessment reports required to demonstrate that the data preparation has been carried out in accordance with the plan can be standardised in the form of checklists to minimise the workload in producing documentation for each installation. This information shall be contained in the Data Test Plan and the results shall be recorded in the Data Test Report.

17.4.2.5 All data and associated documentation shall be subject to the configuration management requirements of clause 15 of this standard. The configuration management records shall list the version of generic software with which the data has been designed to operate, and the versions of the tools used in the data preparation process.

17.4.3 Software Development

Development of the generic software to be used in a system configured by application data shall comply with the requirements in clauses 1 to 16 of this standard. The following additional requirements shall also be observed.

17.4.3.1 During the Software Requirements Specification phase, those functions which make use of application data in each system and subsystem shall be identified. The system safety integrity level allocated to each subsystem will determine the standards to be applied to the subsequent development of the data for all installations of the system.

17.4.3.2 During the Software Design phase the detailed interfaces between the generic software and application data shall be specified, unless this has already been specified at an earlier phase of the life cycle, for example, as a result of a requirement to use an existing application-specific language.

17.4.3.3 During the Software Module Design phase a rigid separation between program code and data shall be enforced, i.e. it shall be possible to recompile and update either the generic software or the data without needing to update the other, unless there has been a change to the defined interface between the software and data. Likewise, application specific data should be separated from other data.

17.4.3.4 Au cours de la phase de maintenance du logiciel, il faut que les procédures de contrôle des modifications assurent que toute modification apportée au logiciel générique puisse être installée uniquement quand il a été établi que le logiciel révisé est compatible avec les données originelles, ou que les données ont été révisées de façon adéquate.

17.4.3.5 Il faut accorder une attention particulière au processus de vérification du logiciel et à la phase de validation du logiciel pour assurer que toutes les combinaisons pertinentes de données sont examinées.

17.4.3.6 Lorsque cela est réalisable, le logiciel générique doit être conçu pour détecter les données de configuration altérées.

17.4.3.4 During the Software Maintenance phase, the change control procedures must ensure that any amendment to the generic software may only be installed after it has been established that either the revised software is compatible with the original data, or the data has been revised as necessary.

17.4.3.5 Care must be taken in the Software Verification process and Software Validation phase in order to assure that all relevant combinations of data are considered.

17.4.3.6 The generic software shall be designed to detect corrupted configuration data where this is feasible.

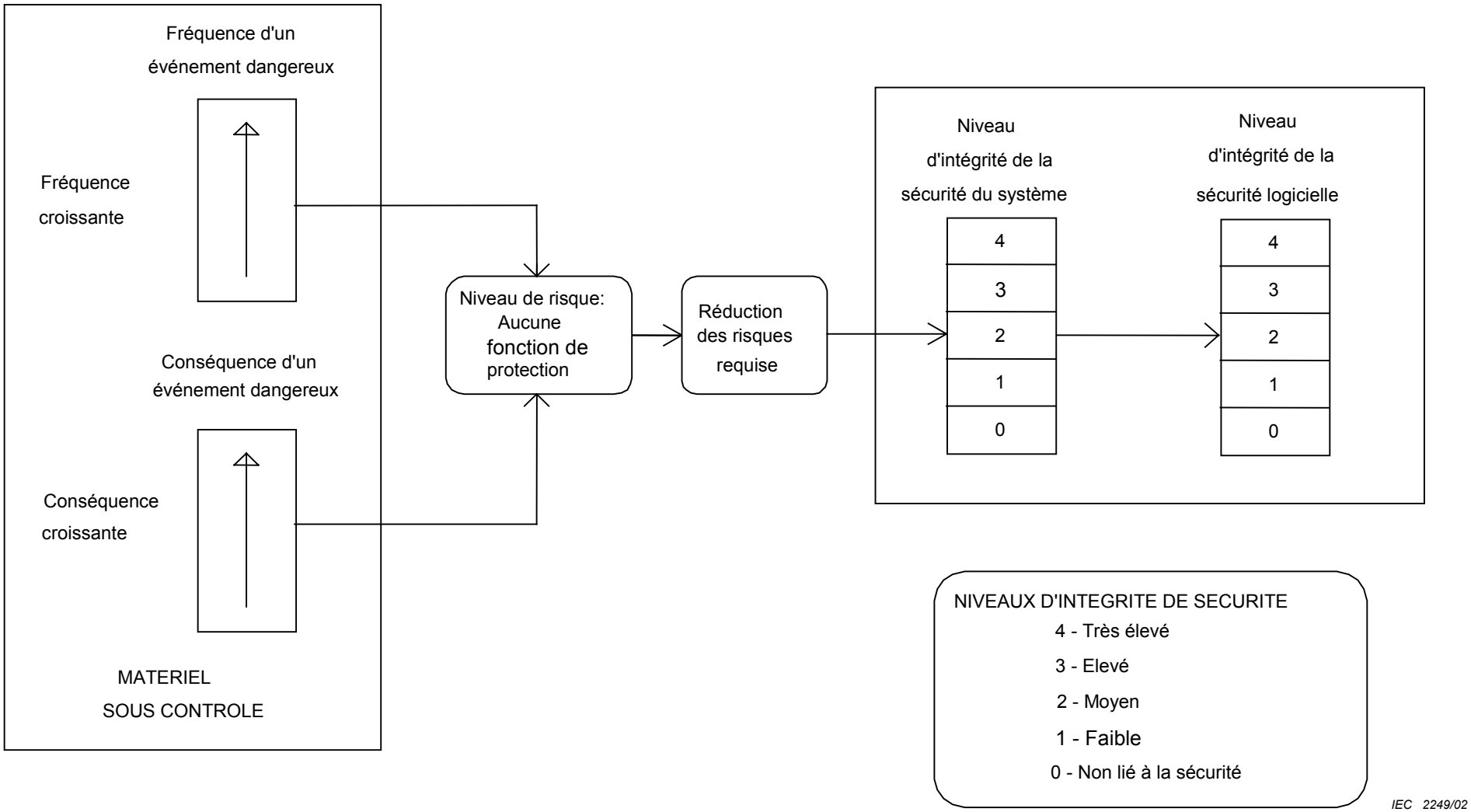


Figure 1 – Niveaux d'intégrité de la sécurité pour les systèmes liés à la sécurité

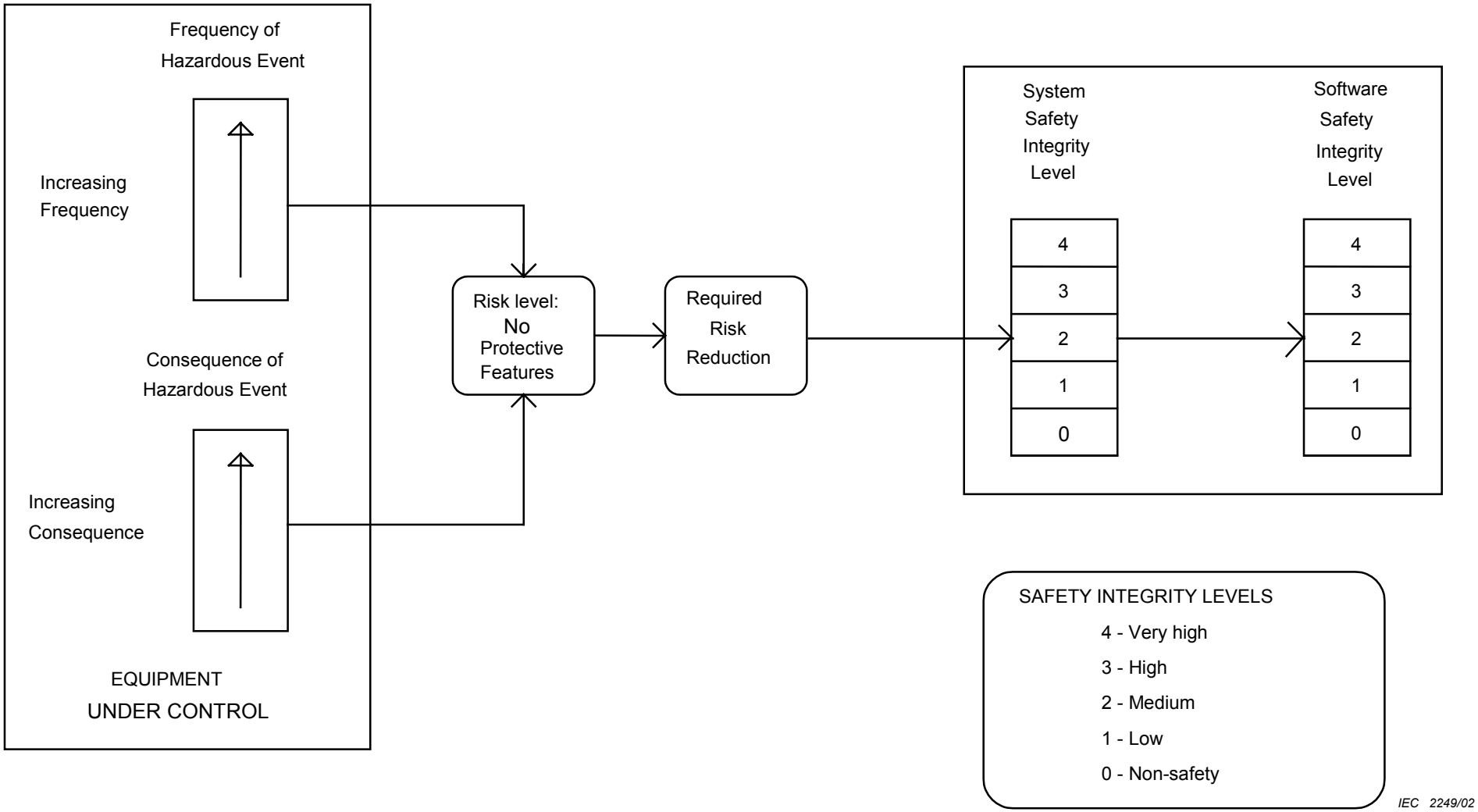


Figure 1 – Integrity Levels for Safety-Related Systems

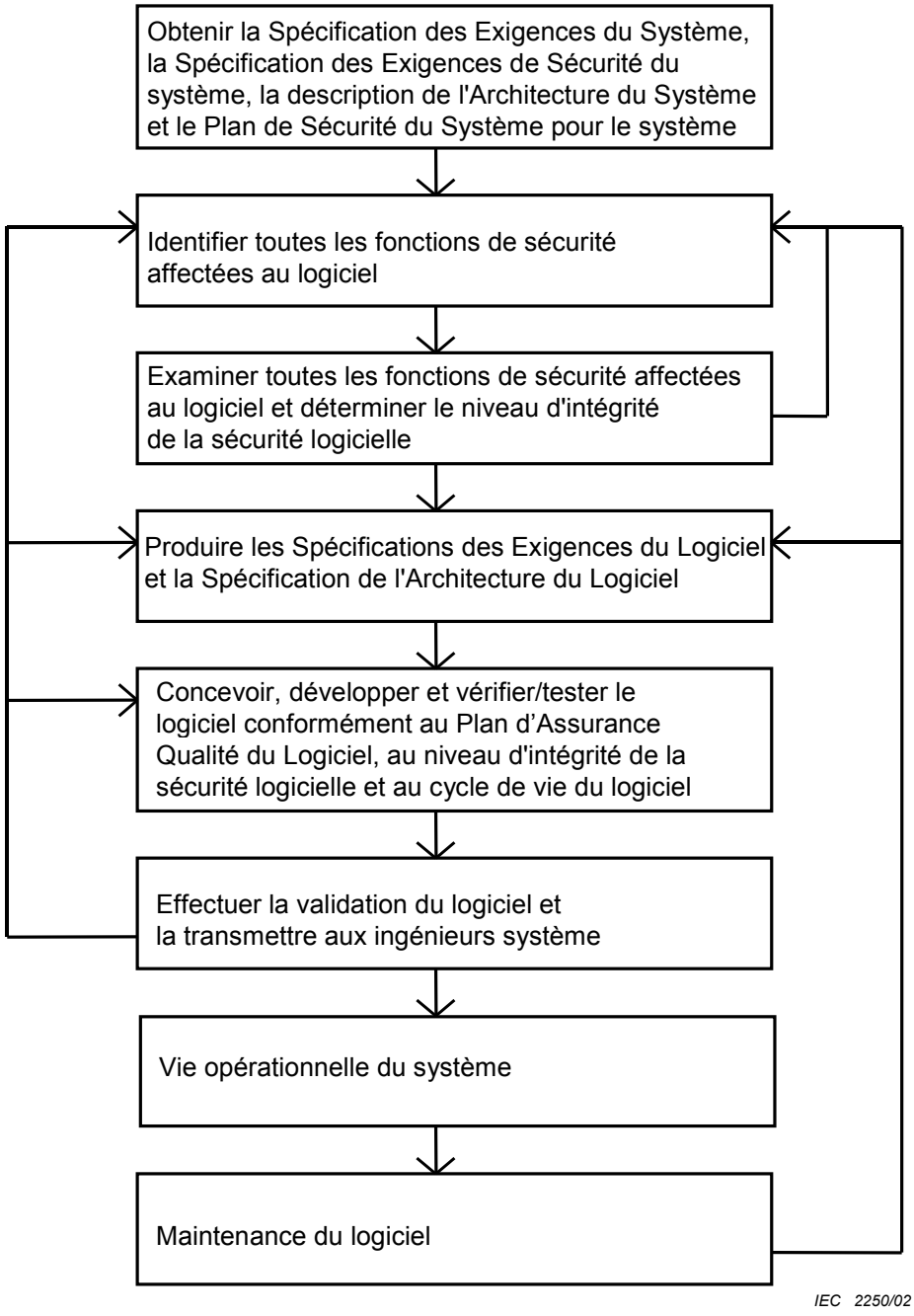


Figure 2 – Démarche de la sécurité du logiciel

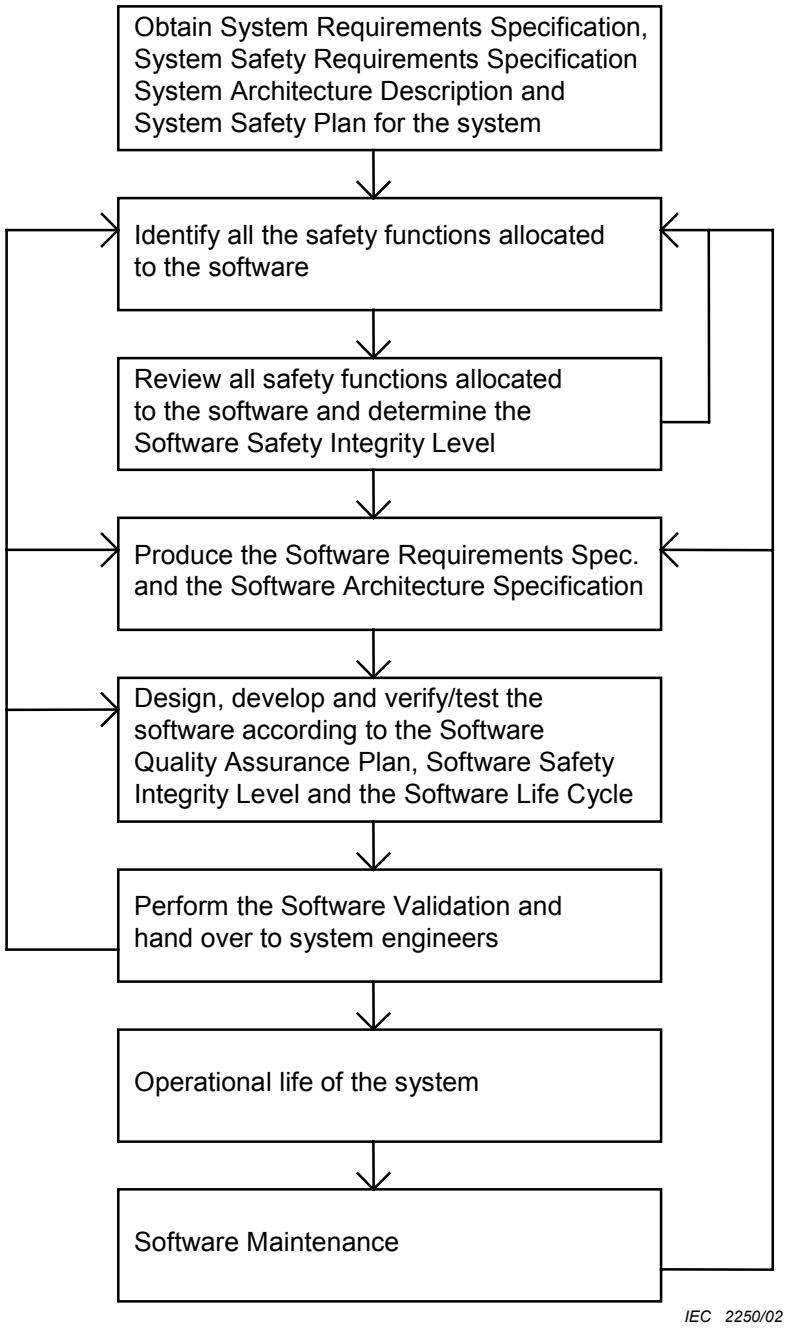


Figure 2 – Software Safety Route Map

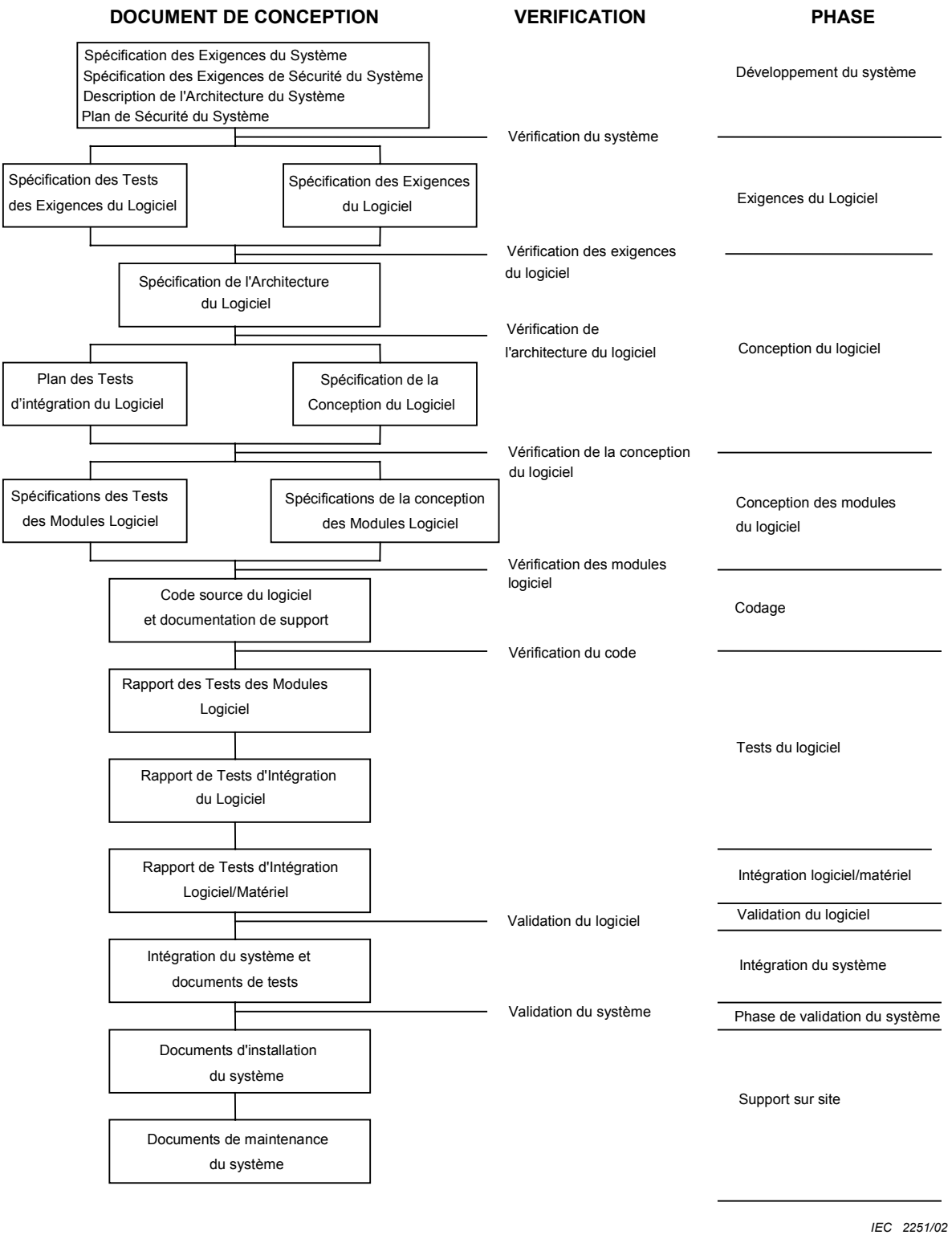


Figure 3 – Cycle de vie 1 du développement

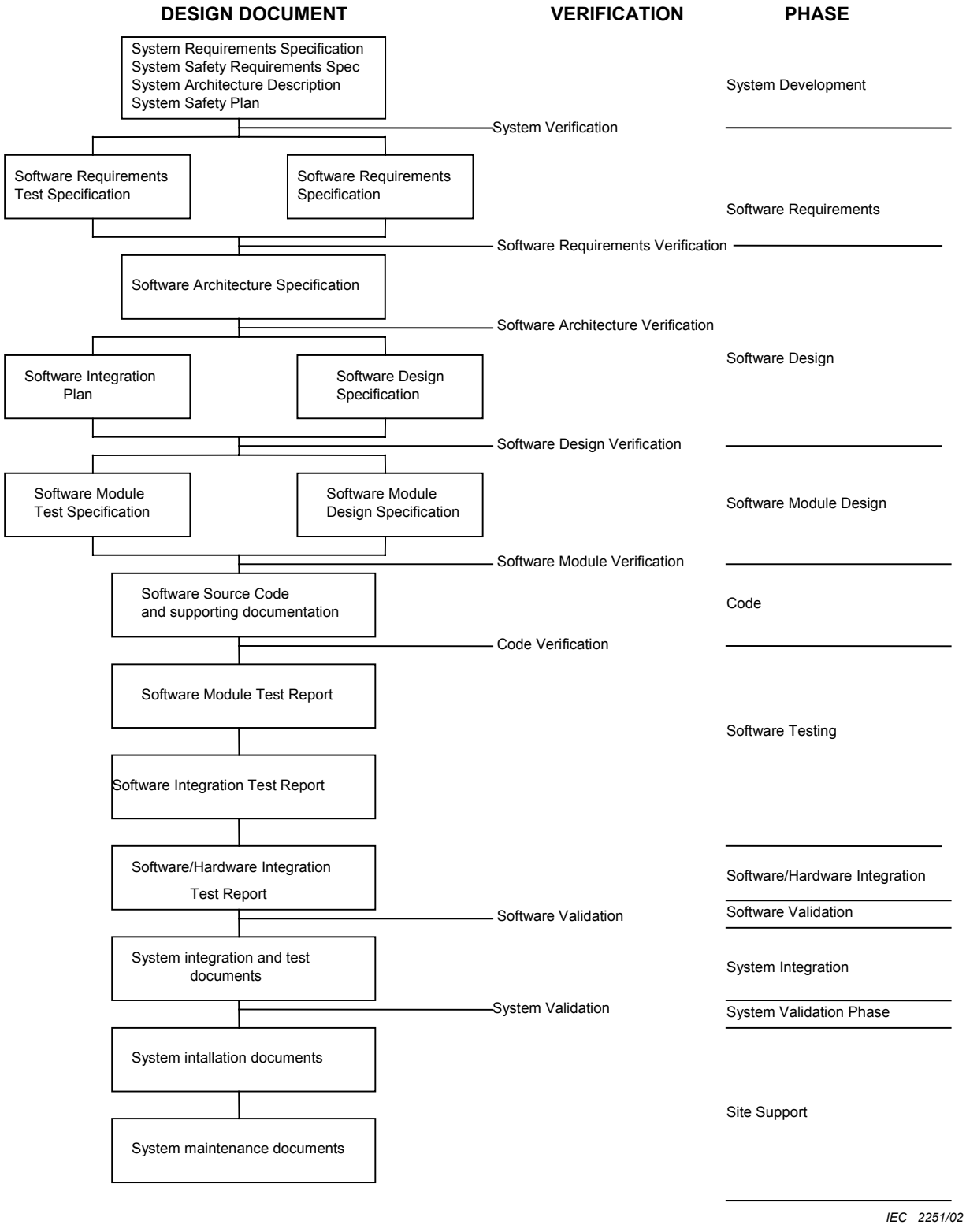


Figure 3 – Development Life Cycle 1

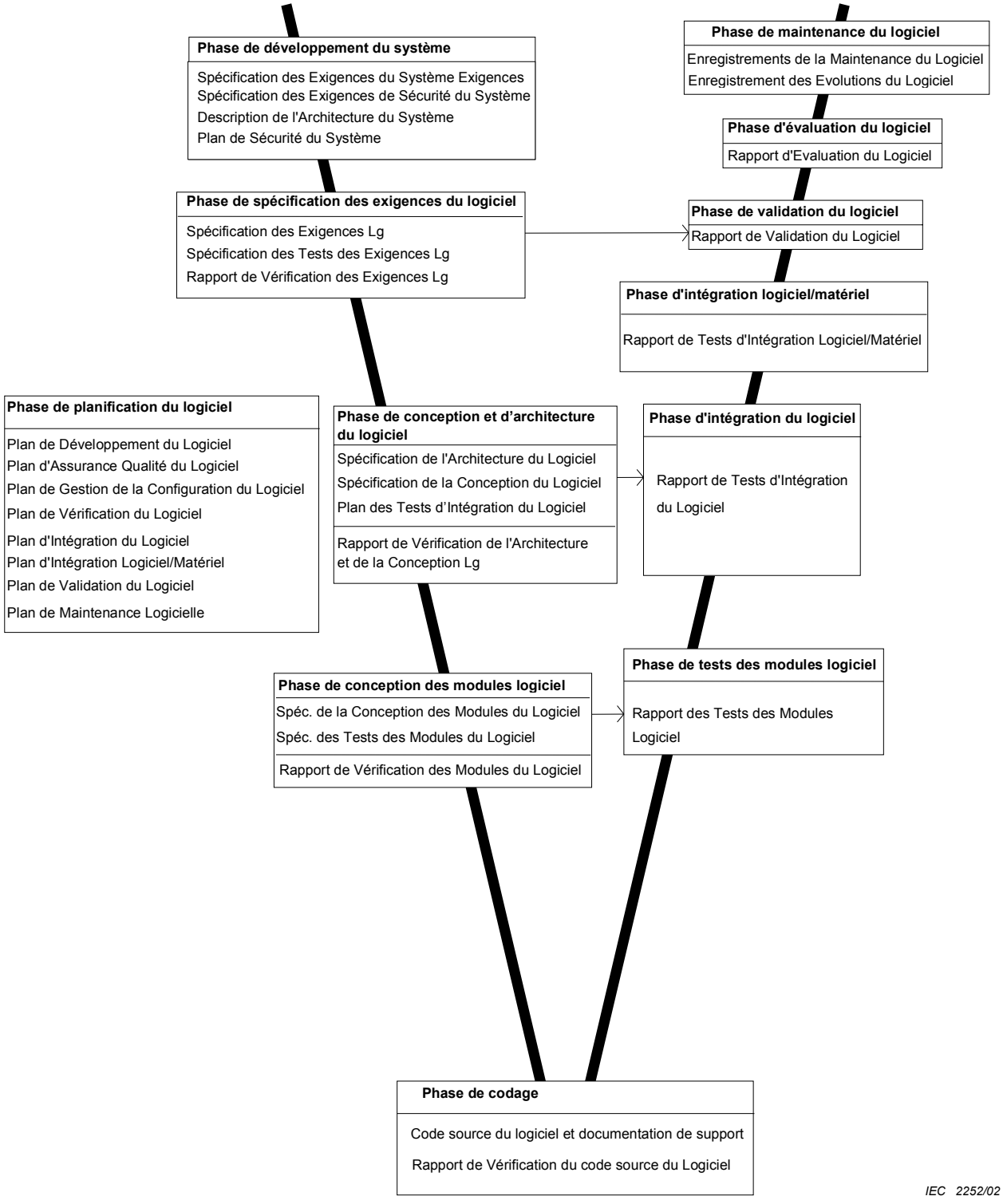


Figure 4 – Cycle de vie 2 du développement

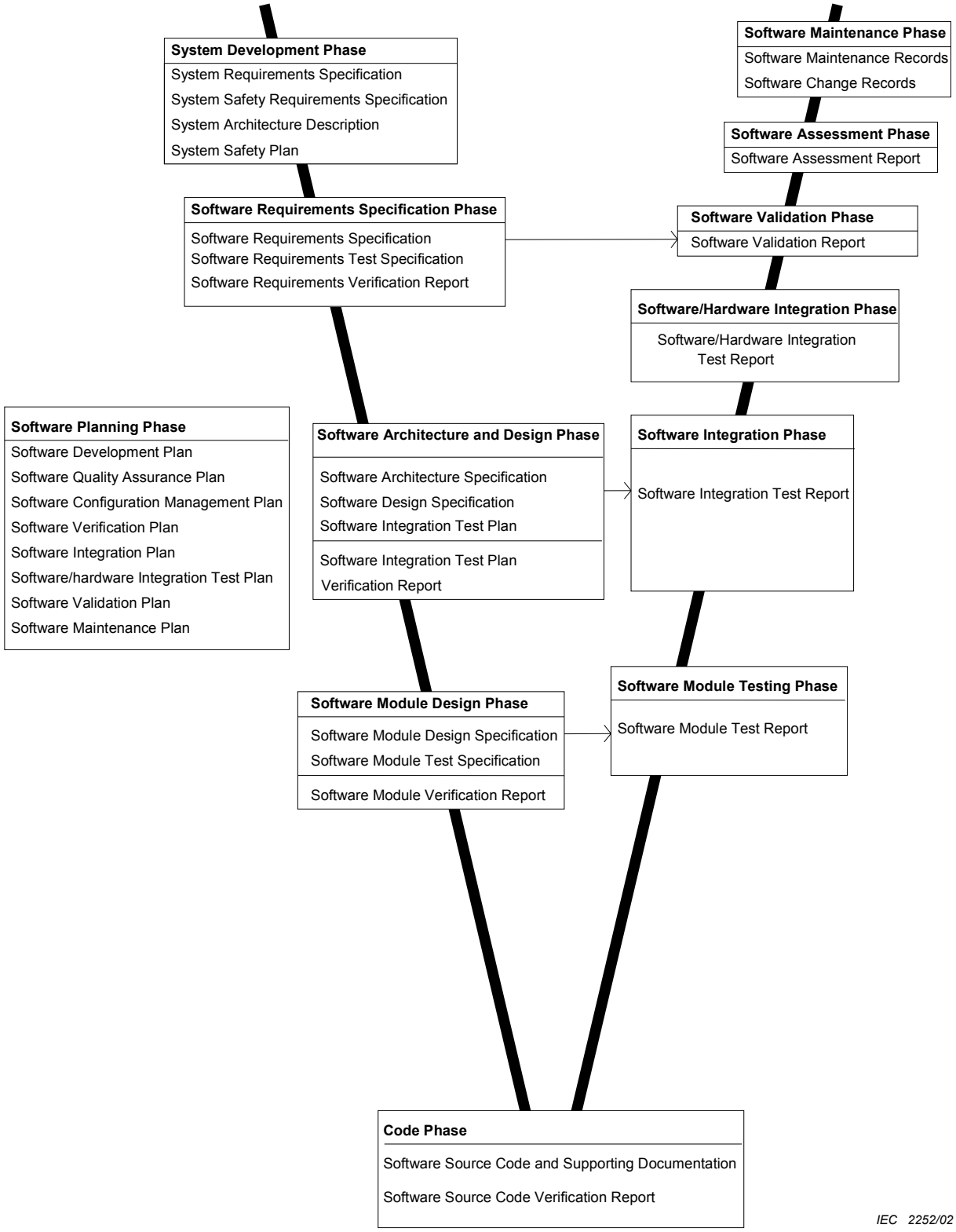
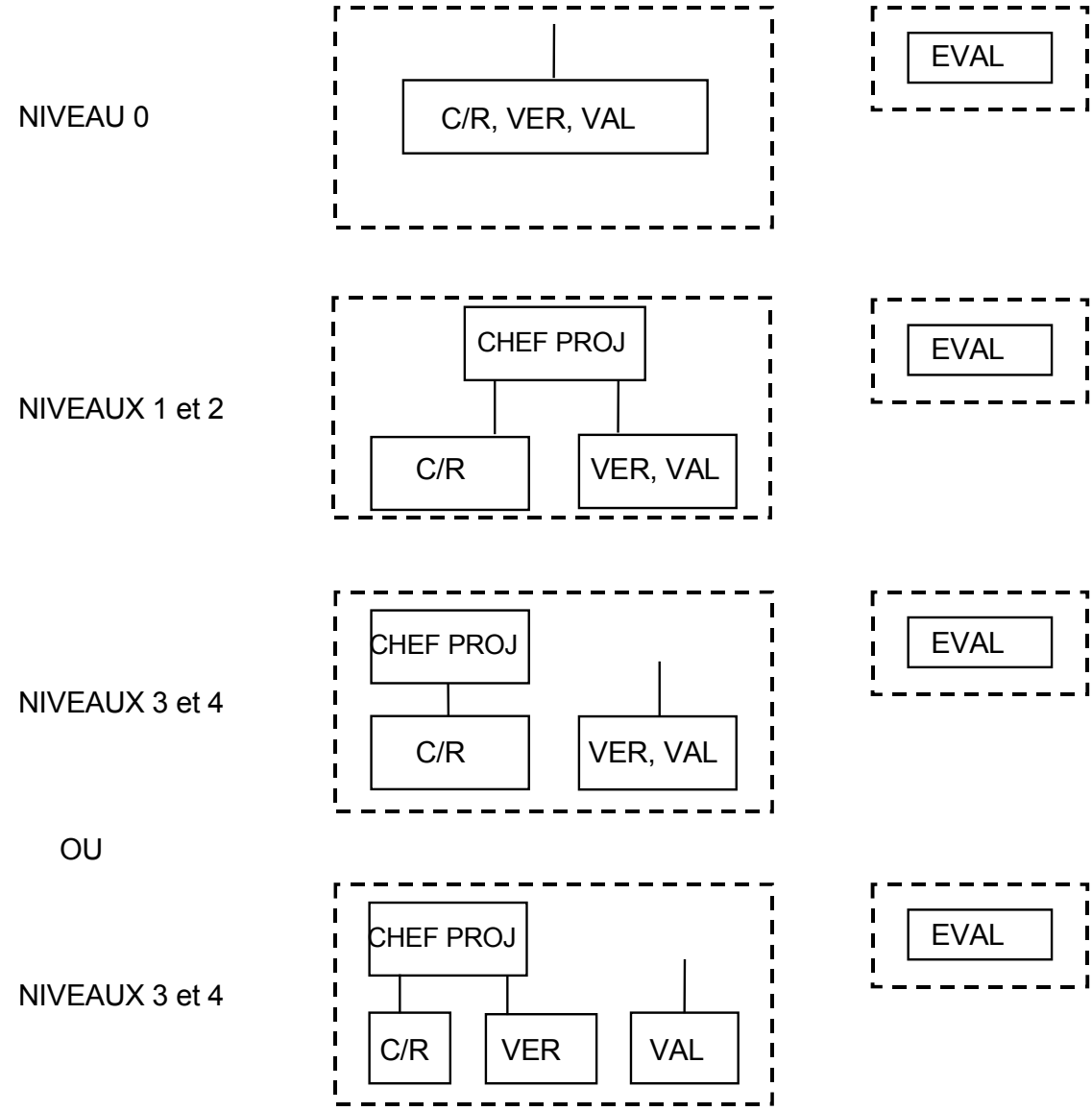


Figure 4 – Development Life Cycle 2



IEC 2253/02

Figure 5 – Indépendance en fonction du niveau d'intégrité de la sécurité du logiciel

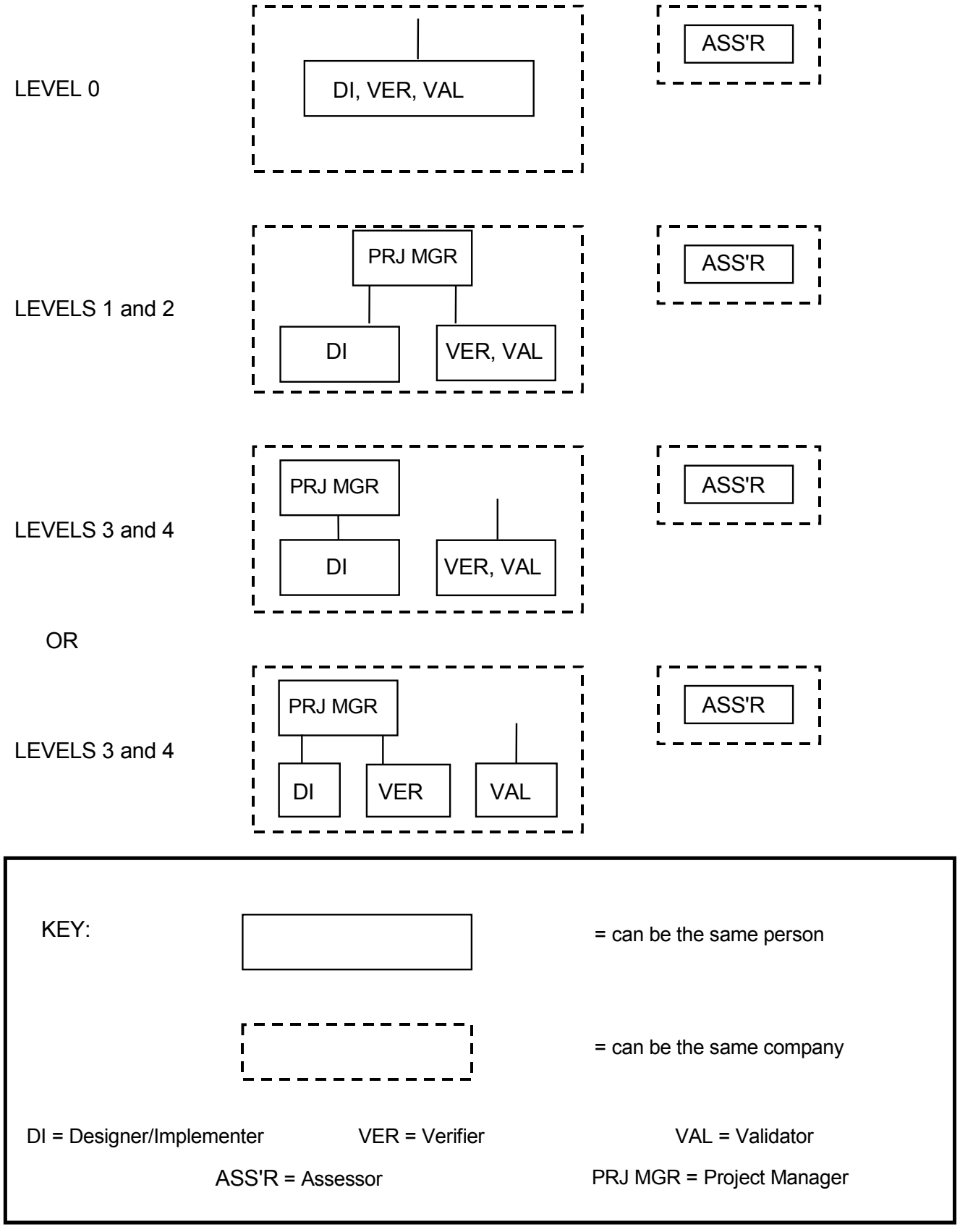


Figure 5 – Independence Versus Software Integrity Level

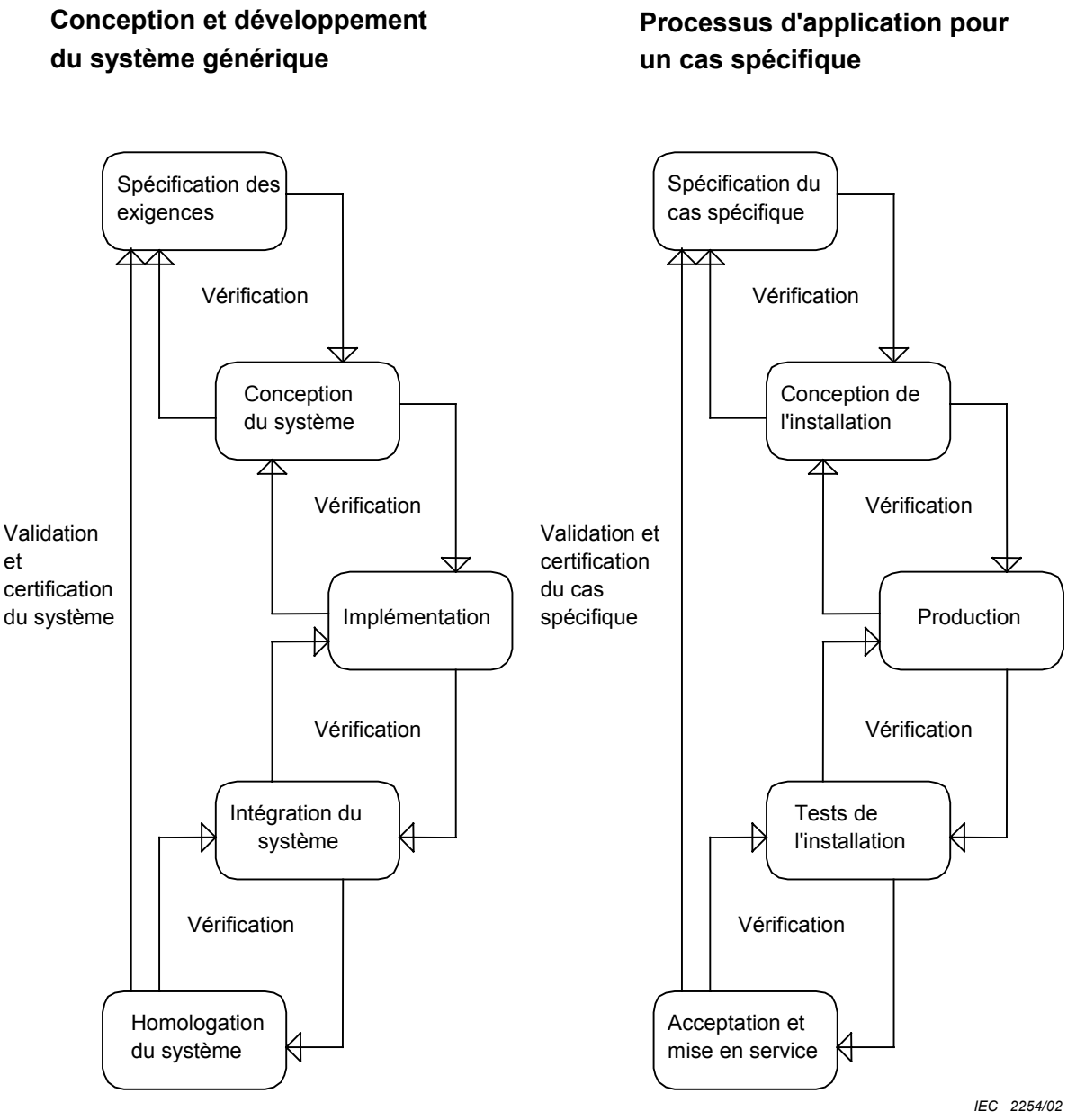


Figure 6 – Relation entre le développement du système générique et le développement du cas spécifique

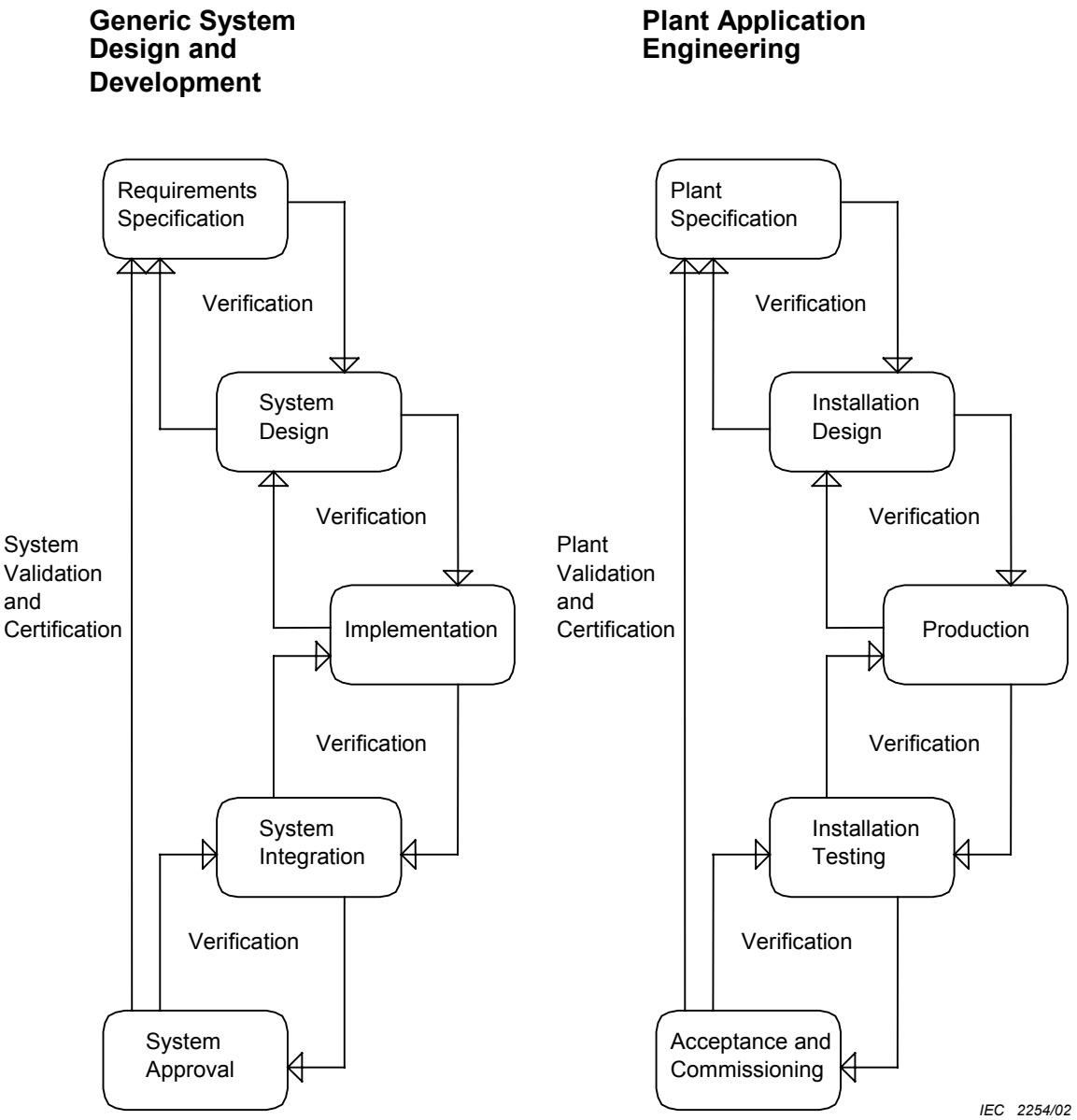


Figure 6 – Relationship between Generic System Development and Application Development

Annexe A
(normative)

Critères de sélection des techniques et mesures

A chacun des articles 7 à 16 de la présente norme est associé un tableau de l'article qui illustre les moyens permettant d'atteindre la conformité. Il existe des tableaux de niveau inférieur, les tableaux détaillés (td)[2], qui développent certaines entrées des tableaux des articles. Par exemple, les méthodes semi-formelles sont développées dans le tableau détaillé td7 (tableau A.18). Il existe aussi une annexe informative B référencée à partir des tableaux des articles.

Avec chaque technique ou mesure figurant dans les tableaux, il existe une exigence pour chacun des niveaux 1 à 4 d'intégrité de la sécurité logicielle (NISL), ainsi que pour le niveau 0 non lié à la sécurité. Dans cette version du document, les exigences pour les niveaux 1 et 2 d'intégrité de la sécurité logicielle sont les mêmes pour chaque technique. De la même façon, chaque technique comporte les mêmes exigences au niveaux 3 et 4 d'intégrité de la sécurité logicielle. Ces exigences peuvent être les suivantes:

'M' ce symbole signifie que l'utilisation d'une technique est obligatoire (Mandatory);

'HR' ce symbole signifie que cette technique ou mesure est Hautement Recommandée pour ce niveau d'intégrité de la sécurité. Si cette technique ou mesure n'est pas utilisée, il convient de détailler le raisonnement sous-tendant cette décision dans le Plan d'Assurance Qualité du Logiciel ou dans un autre document référencé par le Plan d'Assurance Qualité du Logiciel;

'R' ce symbole signifie que la technique ou mesure est Recommandée pour ce niveau d'intégrité de la sécurité. Il s'agit d'un niveau de recommandation inférieur à un symbole 'HR' et il est possible de combiner ces techniques pour former une partie d'un ensemble;

'-' ce symbole signifie que la technique ou mesure ne comporte aucune recommandation en faveur ou contre son utilisation;

'NR' ce symbole signifie que la technique ou mesure est catégoriquement Non Recommandée pour ce niveau d'intégrité de la sécurité. Si cette technique ou mesure est utilisée, il convient de détailler le raisonnement sous-tendant cette décision dans le Plan d'Assurance Qualité du Logiciel ou dans un autre document référencé par le Plan d'Assurance Qualité du Logiciel.

La combinaison de techniques ou de mesures est à préciser dans le Plan d'Assurance Qualité du Logiciel ou dans un autre document référencé par le Plan d'Assurance Qualité du Logiciel en sélectionnant une ou plusieurs techniques ou mesures, à moins que les notes jointes au tableau n'imposent d'autres exigences. Ces notes peuvent inclure la référence à des techniques approuvées ou à des combinaisons de techniques approuvées. Si de telles techniques ou combinaisons sont utilisées, le chargé d'évaluation doit accepter leur validité et ne doit se préoccuper que de leur application correcte. Si un ensemble différent de techniques est utilisé et peut être justifié, il est admis que le chargé d'évaluation les trouve acceptables.

---

2 td = tableau détaillé.
Le premier des tableaux détaillés est le tableau A.12, c'est la raison pour laquelle il porte la référence «td1».

Annex A
(normative)

Criteria for the Selection of Techniques and Measures

Each of clauses 7 to 16 of this standard has an associated clause table to illustrate the means of achieving conformance. There exist lower level tables, the detailed tables (dt)[2], which expand upon certain entries in the clause tables. For example, Semi-Formal Methods are expanded upon in the detailed table dt7 (table A.18). There also exists an informative annex B which is referred to from the clause tables.

With each technique or measure in the tables there is a requirement for each software safety integrity level (SWSIL) 1 to 4 and also for the non-safety-related level 0. In this version of the document, the requirements for software safety integrity levels 1 and 2 are the same for each technique. Similarly, each technique has the same requirements at software safety integrity levels 3 and 4. These requirements can be:

'M' this symbol means that the use of a technique is mandatory;

'HR' this symbol means that the technique or measure is Highly Recommended for this safety integrity level. If this technique or measure is not used then the rationale behind not using it should be detailed in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan;

'R' this symbol means that the technique or measure is Recommended for this safety integrity level. This is a lower level of recommendation than an 'HR' and such techniques can be combined to form part of a package;

'-' this symbol means that the technique or measure has no recommendation for or against being used;

'NR' this symbol means that the technique or measure is positively Not Recommended for this safety integrity level. If this technique or measure is used then the rationale behind using it should be detailed in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan.

The combination of techniques or measures are to be stated in the Software Quality Assurance Plan or in another document referenced by the Software Quality Assurance Plan with one or more techniques or measures being selected unless the notes attached to the table makes other requirements. These notes can include reference to approved techniques or approved combinations of techniques. If such techniques or combinations of techniques are used, then the Assessor shall accept them as valid and shall only be concerned that they have been correctly applied. If a different set of techniques is used and can be justified, then the Assessor may find this acceptable.

---

2 dt = detailed table.
The first detailed table is table A.12; it is for this reason that it bears the reference "dt1".

Tableaux des articles

Tableau A.1 – Problèmes liés au cycle de vie et documentation (article 7)

DOCUMENTATION	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Documents de planification du logiciel	R	HR	HR	HR	HR
2. Documents des exigences du logiciel	R	HR	HR	HR	HR
3. Documents de conception du logiciel	-	HR	HR	HR	HR
4. Documents des modules du logiciel	-	HR	HR	HR	HR
5. Code source et documentation	R	HR	HR	HR	HR
6. Rapport des Tests du Logiciel	-	HR	HR	HR	HR
7. Rapport d'Intégration Logiciel/Matériel	-	HR	HR	HR	HR
8. Rapport de Validation du Logiciel	R	HR	HR	HR	HR
9. Rapport d'Evaluation du Logiciel	-	HR	HR	HR	HR
10. Enregistrements de la Maintenance du Logiciel	R	HR	HR	HR	HR

Exigence

1. La conformité à l'ISO 9000-3 implique la production d'une documentation adéquate pour tous les niveaux d'intégrité de la sécurité logicielle. Pour le niveau 0 d'intégrité de la sécurité logicielle, le concepteur doit choisir les types de document appropriés.

Tableau A.2 – Spécification des Exigences du Logiciel (article 8)

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Méthodes formelles comprenant par exemple CCS, CSP, HOL, LOTOS, OBJ, logique temporelle, VDM, Z et B	B.30	-	R	R	HR	HR
2. Méthodes semi-formelles	D.7	R	R	R	HR	HR
3. Méthode structurée comprenant par exemple JSD, MASCOT, SADT, SDL, SSADM et Yourdon	B.60	R	HR	HR	HR	HR

Exigences

1. La Spécification des Exigences du Logiciel exige toujours une description du problème en langage naturel et toutes les notations mathématiques nécessaires pour refléter l'application.

2. Le tableau reproduit des exigences supplémentaires permettant d'obtenir une spécification claire et précise. Une ou plusieurs de ces techniques doivent être choisies en vue de satisfaire au niveau d'intégrité de la sécurité logicielle utilisé.

Clause tables

Table A.1 – Life Cycle Issues and Documentation (clause 7)

DOCUMENTATION	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Software Planning Documents	R	HR	HR	HR	HR
2. S/W Requirements Documents	R	HR	HR	HR	HR
3. S/W Design Documents	-	HR	HR	HR	HR
4. S/W Module Documents	-	HR	HR	HR	HR
5. Source Code and Documentation	R	HR	HR	HR	HR
6. S/W Test Reports	-	HR	HR	HR	HR
7. S/W and H/W Integration Test Report	-	HR	HR	HR	HR
8. S/W Validation Report	R	HR	HR	HR	HR
9. S/W Assessment Report	-	HR	HR	HR	HR
10. S/W Maintenance Records	R	HR	HR	HR	HR
Requirement Compliance with ISO 9000-3 implies the production of adequate documentation for all Software Safety Integrity Levels. For Software Safety Integrity Level 0, the designer shall choose suitable types of document.					

Table A.2 – Software Requirements Specification (clause 8)

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Formal Methods including for example CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM, Z and B	B.30	-	R	R	HR	HR
2. Semi-Formal Methods	dt7	R	R	R	HR	HR
3. Structured Methodology including, for example, JSD, MASCOT, SADT, SDL, SSADM, and Yourdon	B.60	R	HR	HR	HR	HR
Requirements 1. The Software Requirements Specification will always require a description of the problem in natural language and any necessary mathematical notation that reflects the application. 2. The table reflects additional requirements for defining the specification clearly and precisely. One or more of these techniques shall be selected to satisfy the Software Safety Integrity Level being used.						

Tableau A.3 – Architecture du Logiciel (article 9)

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Programmation défensive	B.15	-	R	R	HR	HR
2. Détection des défauts et diagnostic	B.27	-	R	R	HR	HR
3. Codes correcteurs d'erreurs	B.20	-	-	-	-	-
4. Codes détecteurs d'erreurs	B.20	-	R	R	HR	HR
5. Programmation par assertion	B.25	-	R	R	HR	HR
6. Techniques de sécurité contrôlée (safety bag)	B.54	-	R	R	R	R
7. Programmation diversifiée	B.17	-	R	R	HR	HR
8. Bloc de rattrapage	B.50	-	R	R	R	R
9. Rattrapage par régression	B.5	-	NR	NR	NR	NR
10. Rattrapage par progression	B.32	-	NR	NR	NR	NR
11. Rattrapage par réexécution	B.53	-	R	R	R	R
12. Mémorisation des cas exécutés	B.39	-	R	R	HR	HR
13. Intelligence artificielle – Correction des défauts	B.1	-	NR	NR	NR	NR
14. Reconfiguration dynamique du logiciel	B.18	-	NR	NR	NR	NR
15. Analyse des effets des erreurs du logiciel	B.26	-	R	R	HR	HR
16. Analyse par arbre des causes	B.28	R	R	R	HR	HR

Exigences

1) Les combinaisons de techniques approuvées applicables aux niveaux 3 et 4 d'intégrité de la sécurité logicielle doivent être les suivantes:

   a) 1, 7 et une choisie entre 4, 5 ou 12

   b) 1, 4 et 5

   c) 1, 4 et 12

   d) 1, 2 et 4

   e) 1 et 4, et une choisie entre 15 et 16

2. A l'exception des entrées 3, 9, 10, 13 et 14, une ou plusieurs de ces techniques doivent être sélectionnées pour satisfaire aux exigences des niveaux 1 et 2 d'intégrité de la sécurité logicielle.

3. Il est permis de définir certains de ces problèmes au niveau du système.

4. Des codes correcteurs d'erreurs peuvent être utilisés selon les exigences de la CEI 62280-1 et de la CEI 62280-2.

Table A.3 – Software Architecture (clause 9)

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Defensive Programming	B.15	-	R	R	HR	HR
2. Fault Detection and Diagnosis	B.27	-	R	R	HR	HR
3. Error Correcting Codes	B.20	-	-	-	-	-
4. Error Detecting Codes	B.20	-	R	R	HR	HR
5. Failure Assertion Programming	B.25	-	R	R	HR	HR
6. Safety Bag Techniques	B.54	-	R	R	R	R
7. Diverse Programming	B.17	-	R	R	HR	HR
8. Recovery Block	B.50	-	R	R	R	R
9. Backward Recovery	B.5	-	NR	NR	NR	NR
10. Forward Recovery	B.32	-	NR	NR	NR	NR
11. Re-try Fault Recovery Mechanisms	B.53	-	R	R	R	R
12. Memorising Executed Cases	B.39	-	R	R	HR	HR
13. Artificial Intelligence – Fault Correction	B.1	-	NR	NR	NR	NR
14. Dynamic Reconfiguration of software	B.18	-	NR	NR	NR	NR
15. Software Error Effect Analysis	B.26	-	R	R	HR	HR
16. Fault Tree Analysis	B.28	R	R	R	HR	HR

Requirements

1. Approved combinations of techniques for Software Safety Integrity Levels 3 and 4 shall be as follows:
   a) 1, 7 and one from 4, 5 or 12
   b) 1, 4 and 5
   c) 1, 4 and 12
   d) 1, 2 and 4
   e) 1 and 4, and one of 15 and 16
2. With the exception of entries 3, 9, 10, 13 and 14, one or more of these techniques shall be selected to satisfy the requirements for Software Safety Integrity Levels 1 and 2.
3. Some of these issues may be defined at the system level.
4. Error correcting codes may be used in accordance with the requirements of IEC 62280-1 and IEC 62280-2.

Tableau A.4 – Conception et mise en œuvre du Logiciel (article 10)

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Méthodes formelles comprenant par exemple CCS, CSP, HOL, LOTOS, OBJ, logique temporelle, VDM, Z et B	B.30	-	R	R	HR	HR
2. Méthodes semi-formelles	td7	R	HR	HR	HR	HR
3. Méthode structurée comprenant par exemple JSD, MASCOT, SADT, SDL, SSADM et Yourdon	B.60	R	HR	HR	HR	HR
4. Approche modulaire	td9	HR	M	M	M	M
5. Normes de conception et de codage	td1	HR	HR	HR	M	M
6. Programmes analysables	B.2	HR	HR	HR	HR	HR
7. Langage de programmation à fort typage	B.57	R	HR	HR	HR	HR
8. Programmation structurée	B.61	R	HR	HR	HR	HR
9. Langage de programmation	td4	R	HR	HR	HR	HR
10. Sous-ensemble de langage	B.38	-	-	-	HR	HR
11. Traducteur validé	B.7	R	HR	HR	HR	HR
12. Traducteur éprouvé à l'utilisation	B.65	HR	HR	HR	HR	HR
13. Bibliothèque de modules et de composants sécurisés/vérifiés	B.40	R	R	R	R	R
14. Tests fonctionnels et boîte noire	td3	HR	HR	HR	M	M
15. Tests de performance	td6	-	HR	HR	HR	HR
16. Tests d'interface	B.37	HR	HR	HR	HR	HR
17. Enregistrement et analyse des données	B.13	HR	HR	HR	M	M
18. Logique floue	B.67	-	-	-	-	-
19. Programmation orientée objet	B.68	-	R	R	R	R

Exigences

1. Un ensemble de techniques adapté doit être choisi en fonction du niveau d'intégrité de la sécurité logicielle.
2. A une intégrité de la sécurité logicielle de niveau 3 ou 4, l'ensemble de techniques approuvées doit inclure l'une des techniques 1, 2 ou 3, ainsi que l'une des techniques 11 ou 12. Les autres techniques doivent être traitées conformément à leurs recommandations.

Table A.4 – Software Design and Implementation (clause 10)

TECHNIQUE/MEASURE	Ref.	SWSIL O	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Formal Methods including for example CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM, Z and B	B.30	-	R	R	HR	HR
2. Semi-Formal Methods	dt7	R	HR	HR	HR	HR
3. Structured Methodology including for example JSD, MASCOT, SADT, SDL, SSADM and Yourdon	B.60	R	HR	HR	HR	HR
4. Modular Approach	dt9	HR	M	M	M	M
5. Design and Coding Standards	dt1	HR	HR	HR	M	M
6. Analysable Programs	B.2	HR	HR	HR	HR	HR
7. Strongly Typed Programming Language	B.57	R	HR	HR	HR	HR
8. Structured Programming	B.61	R	HR	HR	HR	HR
9. Programming Language	dt4	R	HR	HR	HR	HR
10. Language Subset	B.38	-	-	-	HR	HR
11. Validated Translator	B.7	R	HR	HR	HR	HR
12. Translator Proven in Use	B.65	HR	HR	HR	HR	HR
13. Library of Trusted/Verified Modules and Components	B.40	R	R	R	R	R
14. Functional/ Black-box Testing	dt3	HR	HR	HR	M	M
15. Performance Testing	dt6	-	HR	HR	HR	HR
16. Interface Testing	B.37	HR	HR	HR	HR	HR
17. Data Recording and Analysis	B.13	HR	HR	HR	M	M
18. Fuzzy Logic	B.67	-	-	-	-	-
19. Object Oriented Programming	B.68	-	R	R	R	R

Requirements

1. A suitable set of techniques shall be chosen according to the software safety integrity level.
2. At software safety integrity level 3 or 4, the approved set of techniques shall include one of techniques 1, 2 or 3, together with one of techniques 11 or 12. The remaining techniques shall still be treated according to their recommendations.

Tableau A.5 – Vérification et Tests (article 11)

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Preuve formelle	B.31	-	R	R	HR	HR
2. Tests probabilistes	B.47	-	R	R	HR	HR
3. Analyse statique	td8	-	HR	HR	HR	HR
4. Analyse et tests dynamiques	td2	-	HR	HR	HR	HR
5. Métriques	B.42	-	R	R	R	R
6. Matrice de traçabilité	B.69	-	R	R	HR	HR
7. Analyse des effets des erreurs du logiciel	B.26	-	R	R	HR	HR

Exigences

1. Pour un niveau 3 ou 4 d'intégrité de la sécurité logicielle, les combinaisons approuvées de techniques doivent être:
   a) 1 et 4 ou
   b) 3 et 4 ou
   c) 4, 6 et 7
2. Pour un niveau 1 ou 2 d'intégrité de la sécurité logicielle, les combinaisons approuvées de techniques doivent être:
   a) 1 ou
   b) 3 et 4 ou
   c) 4

Tableau A.6 – Intégration Logiciel/Matériel (article 12)

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Tests fonctionnels et boîte noire	td3	HR	HR	HR	HR	HR
2. Tests de performance	td6	-	R	R	HR	HR

Exigences

1. Pour un niveau 0 d'intégrité de la sécurité logicielle, la technique 1 doit être la technique approuvée.
2. Pour un niveau 1, 2, 3 ou 4 d'intégrité de la sécurité logicielle, la combinaison approuvée de techniques doit être 1 et 2.

Tableau A.7 – Validation du Logiciel (article 13)

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Tests probabilistes	B.47	-	R	R	HR	HR
2. Tests de performance	td6	-	HR	HR	M	M
3. Tests fonctionnels et boîte noire	td3	HR	HR	HR	M	M
4. Modélisation	td5	-	R	R	R	R

Exigence

Pour un niveau 1, 2, 3 ou 4 d'intégrité de la sécurité logicielle, une combinaison approuvée de techniques doit être 2 et 3.

Table A.5 – Verification and Testing (clause 11)

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Formal Proof	B.31	-	R	R	HR	HR
2. Probabilistic Testing	B.47	-	R	R	HR	HR
3. Static Analysis	dt8	-	HR	HR	HR	HR
4. Dynamic Analysis and Testing	dt2	-	HR	HR	HR	HR
5. Metrics	B.42	-	R	R	R	R
6. Traceability Matrix	B.69	-	R	R	HR	HR
7. Software Error Effect Analysis	B26	-	R	R	HR	HR
Requirements 1. For Software Safety Integrity Level 3 or 4, the approved combinations of techniques shall be: a) 1 and 4 or b) 3 and 4 or c) 4, 6 and 7 2. For Software Safety Integrity Level 1 or 2, the approved combinations of techniques shall be: a) 1 or b) 3 and 4 or c) 4						

Table A.6 – Software/Hardware Integration (clause 12)

TECHNIQUE/MEASURE	Ref.	SWSIL O	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Functional and Black-box Testing	dt3	HR	HR	HR	HR	HR
2. Performance Testing	dt6	-	R	R	HR	HR
Requirements 1. For software safety integrity level 0, technique 1 shall be the approved technique. 2. For software safety integrity level 1, 2, 3 or 4, the approved combination of techniques shall be 1 and 2.						

Table A.7 – Software Validation (clause 13)

TECHNIQUE/MEASURE	Ref.	SWSIL O	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Probabilistic Testing	B.47	-	R	R	HR	HR
2. Performance Testing	dt6	-	HR	HR	M	M
3. Functional and Black-box Testing	dt3	HR	HR	HR	M	M
4. Modelling	dt5	-	R	R	R	R
Requirement For Software Safety Integrity Level 1, 2, 3 or 4, an approved combination of techniques shall be 2 and 3.						

Tableau A.8 – Articles à évaluer

ARTICLE A EVALUER	Article	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Niveaux d'intégrité de la sécurité du logiciel	5	HR	HR	HR	HR	HR
2. Personnel et responsabilité	6	-	R	R	HR	HR
3. Cycle de vie et documentation	7	-	HR	HR	HR	HR
4. Spécification des Exigences du Logiciel	8	R	HR	HR	HR	HR
5. Architecture du logiciel	9	-	R	R	HR	HR
6. Conception et développement	10	-	R	R	HR	HR
7. Vérification	11	-	HR	HR	HR	HR
8. Intégration logiciel/matériel	12	-	R	R	HR	HR
9. Validation du logiciel	13	-	HR	HR	HR	HR
10. Assurance qualité	15	-	HR	HR	HR	HR
11. Maintenance	16	-	HR	HR	HR	HR

Tableau A.9 – Evaluation du logiciel (article 14)
Techniques d'évaluation

TECHNIQUE D'EVALUATION	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Listes de contrôle	B.8	HR	HR	HR	HR	HR
2. Analyse logicielle statique	B.14 B.42 td8	R	HR	HR	HR	HR
3. Analyse logicielle dynamique	td2 td3	-	R	R	HR	HR
4. Schémas de cause et de conséquence	B.6	R	R	R	R	R
5. Analyse par arbre des événements	B.23	-	R	R	R	R
6. Analyse par arbre de causes	B.28	R	R	R	HR	HR
7. Analyse des effets des erreurs du logiciel	B.26	-	R	R	HR	HR
8. Analyse des défaillances de mode commun	B.10	-	R	R	HR	HR
9. Modèles de Markov	B.41	-	R	R	R	R
10. Schéma bloc de la fiabilité	B.51	-	R	R	R	R
11. Essai sur le terrain avant la mise en service	-	R	HR	HR	HR	HR
Exigence Une ou plusieurs de ces techniques doivent être sélectionnées pour satisfaire au niveau d'intégrité de la sécurité logicielle utilisé.						

Tableau A.10 – Assurance qualité du logiciel (article 15)

TECHNIQUE/MESURE	Article ou réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Accréditée par l'ISO 9001	15	R	HR	HR	HR	HR
2. Conforme à l'ISO 9000-3	15	M	M	M	M	M
3. Système qualité de l'entreprise	15	M	M	M	M	M
4. Gestion de la configuration du logiciel	B.56	M	M	M	M	M

Table A.8 – Clauses to be assessed

CLAUSE TO BE ASSESSED	Clause	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. S/W Safety Integrity Levels	5	HR	HR	HR	HR	HR
2. Personnel and Responsibility	6	-	R	R	HR	HR
3. Life Cycle and Documentation	7	-	HR	HR	HR	HR
4. S/W Requirements Specification	8	R	HR	HR	HR	HR
5. S/W Architecture	9	-	R	R	HR	HR
6. Design and Development	10	-	R	R	HR	HR
7. Verification	11	-	HR	HR	HR	HR
8. S/W/H/W Integration	12	-	R	R	HR	HR
9. S/W Validation	13	-	HR	HR	HR	HR
10. Quality Assurance	15	-	HR	HR	HR	HR
11. Maintenance	16	-	HR	HR	HR	HR

Table A.9 – Software Assessment (clause 14)
Assessment Techniques

ASSESSMENT TECHNIQUE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Checklists	B.8	HR	HR	HR	HR	HR
2. Static Software Analysis	B.14 B.42 dt8	R	HR	HR	HR	HR
3. Dynamic Software Analysis	dt2 dt3	-	R	R	HR	HR
4. Cause Consequence Diagrams	B.6	R	R	R	R	R
5. Event Tree Analysis	B.23	-	R	R	R	R
6. Fault Tree Analysis	B.28	R	R	R	HR	HR
7. Software Error Effect Analysis	B.26	-	R	R	HR	HR
8. Common Cause Failure Analysis	B.10	-	R	R	HR	HR
9. Markov Models	B.41	-	R	R	R	R
10. Reliability Block Diagram	B.51	-	R	R	R	R
11. Field Trial Before Commissioning	-	R	HR	HR	HR	HR
Requirement One or more of these techniques shall be selected to satisfy the Software Safety Integrity Level being used.						

Table A.10 – Software Quality Assurance (clause 15)

TECHNIQUE/MEASURE	Clause or ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Accredited to ISO 9001	15	R	HR	HR	HR	HR
2. Compliant with ISO 9000-3	15	M	M	M	M	M
3. Company Quality System	15	M	M	M	M	M
4. Software Configuration Management	B.56	M	M	M	M	M

Tableau A.11 – Maintenance du logiciel (article 16)

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Analyse d'impact	B.35	R	HR	HR	M	M
2. Enregistrement et analyse des données	B.13	HR	HR	HR	M	M

Tableaux détaillés (td)

Tableau A.12 – Normes de conception et de codage (td1)
Référencé par l'article 10

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. La norme de codage existe	B.16	HR	HR	HR	HR	HR
2. Guide du style de codage	B.16	HR	HR	HR	HR	HR
3. Pas d'objets dynamiques	B.16	-	R	R	HR	HR
4. Pas de variables dynamiques	B.16	-	R	R	HR	HR
5. Usage limité des pointeurs	B.16	-	R	R	R	R
6. Usage limité de la récursivité	B.16	-	R	R	HR	HR
7. Pas de branchements inconditionnels	B.16	-	HR	HR	HR	HR
Exigences 1. Il est accepté que les techniques 3, 4 et 5 soient présentes comme partie intégrante d'un compilateur ou d'un traducteur validé. 2. Un ensemble approprié de techniques doit être choisi en fonction du niveau d'intégrité de la sécurité.						

Tableau A.13 – Analyse et tests dynamiques (td2)
Référencé par les articles 11 et 14

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Exécution d'un dossier de tests à partir d'une analyse des valeurs aux limites	B.4	-	HR	HR	HR	HR
2. Exécution d'un dossier de tests à partir de la supposition d'erreurs	B.21	R	R	R	R	R
3. Exécution d'un modèle de tests à partir de l'insertion d'erreurs	B.22	-	R	R	R	R
4. Modélisation des performances	B.45	-	R	R	HR	HR
5. Tests de classes d'équivalence et de partition d'entrée	B.19	-	R	R	HR	HR
6. Tests structurels	B.58	-	R	R	HR	HR
Exigences 1. Pour les dossiers de test, l'analyse se fait au niveau du sous-système et repose sur la spécification et/ou sur la spécification et le code. 2. Un ensemble de techniques adapté doit être choisi en fonction du niveau d'intégrité de la sécurité logicielle.						

Table A.11 – Software Maintenance (clause 16)

TECHNIQUE/MEASURE	Ref.	SWSIL O	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Impact Analysis	B.35	R	HR	HR	M	M
2. Data Recording and Analysis	B.13	HR	HR	HR	M	M

Detailed tables (dt)

Table A.12 – Design and Coding Standards (dt1)
Referenced by clause 10

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Coding Standard Exists	B.16	HR	HR	HR	HR	HR
2. Coding Style Guide	B.16	HR	HR	HR	HR	HR
3. No Dynamic Objects	B.16	-	R	R	HR	HR
4. No Dynamic Variables	B.16	-	R	R	HR	HR
5. Limited Use of Pointers	B.16	-	R	R	R	R
6. Limited Use of Recursion	B.16	-	R	R	HR	HR
7. No Unconditional Jumps	B.16	-	HR	HR	HR	HR
Requirements 1. It is accepted that techniques 3, 4 and 5 may be present as part of a validated compiler or translator. 2. A suitable set of techniques shall be chosen according to the software safety integrity level.						

Table A.13 – Dynamic Analysis and Testing (dt2)
Referenced by clauses 11 and 14

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Test Case Execution from Boundary Value Analysis	B.4	-	HR	HR	HR	HR
2. Test Case Execution from Error Guessing	B.21	R	R	R	R	R
3. Test Case Execution from Error Seeding	B.22	-	R	R	R	R
4. Performance Modelling	B.45	-	R	R	HR	HR
5. Equivalence Classes and Input Partition Testing	B.19	-	R	R	HR	HR
6. Structure-Based Testing	B.58	-	R	R	HR	HR
Requirements 1. The analysis for the test cases is at the subsystem level and is based on the specification and/or the specification and the code. 2. A suitable set of techniques shall be chosen according to the software safety integrity level.						

Tableau A.14 – Test fonctionnel/boîte noire (td3)
Référencé par les articles 10, 12, 13 et 14

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Exécution d'un dossier de test à partir des schémas de cause et de conséquence	B.6	-	-	-	R	R
2. Prototypage/Animation	B.49	-	-	-	R	R
3. Analyse des valeurs aux limites	B.4	R	HR	HR	HR	HR
4. Tests de classes d'équivalence et de partition d'entrée	B.19	R	HR	HR	HR	HR
5. Simulation du processus	B.48	R	R	R	R	R

Exigences

1. L'exhaustivité de la simulation dépendra de la limite du niveau d'intégrité de la sécurité logicielle, de la complexité et de l'application.
2. Un ensemble de techniques adapté doit être choisi en fonction du niveau d'intégrité de la sécurité logicielle.

Tableau A.15 – Langages de programmation (td4)
Référencé par l'article 10

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. ADA	B.62	R	HR	HR	R	R
2. MODULA-2	B.62	R	HR	HR	R	R
3. PASCAL	B.62	R	HR	HR	R	R
4. Fortran 77	B.62	R	R	R	R	R
5. C ou C++ (sans restriction)	B.62	R	-	-	NR	NR
6. Sous-ensemble du C ou du C++ avec normes de codage	B.62 B.38	R	R	R	R	R
7. PL/M	B.62	R	R	R	NR	NR
8. BASIC	B.62	R	NR	NR	NR	NR
9. Assembleur	B.62	R	R	R	-	-
10. Schémas à contacts	B.62	R	R	R	R	R
11. Blocs fonctionnels	B.62	R	R	R	R	R
12. Liste d'instructions	B.62	R	R	R	R	R

Exigences

1. Aux niveaux 3 et 4 d'intégrité de la sécurité logicielle, quand un sous-ensemble des langages 1, 2, 3 et 4 est utilisé, la recommandation devient HR.
2. Pour certaines applications, il n'est pas exclu que les langages 7 et 9 soient les seuls disponibles. Aux niveaux 3 et 4 d'intégrité de la sécurité logicielle, quand l'option Hautement Recommandée n'est pas disponible, il est vivement conseillé, pour augmenter la recommandation à «R», d'utiliser un sous-ensemble de ces langages ainsi qu'un ensemble précis de normes de codage.
3. Pour toute information sur l'évaluation de l'adéquation d'un langage de programmation, consulter la bibliographie pour «Un langage de programmation adapté», B.62.
4. Si un langage spécifique ne figure pas dans le tableau, il n'est pas exclu automatiquement. Il convient, toutefois, qu'il soit en conformité avec B.62.

Table A.14 – Functional/Black-box Test (dt3)
Referenced by clauses 10,12, 13 and 14

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Test Case Execution from Cause Consequence Diagrams	B.6	-	-	-	R	R
2. Prototyping/Animation	B.49	-	-	-	R	R
3. Boundary Value Analysis	B.4	R	HR	HR	HR	HR
4. Equivalence Classes and Input Partition Testing	B.19	R	HR	HR	HR	HR
5. Process Simulation	B.48	R	R	R	R	R

Requirements

1. The completeness of the simulation will depend upon the extent of the software safety integrity level, complexity and application.
2. A suitable set of techniques shall be chosen according to the software safety integrity level.

Table A.15 – Programming Languages (dt4)
Referenced by clause 10

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. ADA	B.62	R	HR	HR	R	R
2. MODULA-2	B.62	R	HR	HR	R	R
3. PASCAL	B.62	R	HR	HR	R	R
4. Fortran 77	B.62	R	R	R	R	R
5. C or C++ (unrestricted)	B.62	R	-	-	NR	NR
6. Subset of C or C++ with coding standards	B.62 B.38	R	R	R	R	R
7. PL/M	B.62	R	R	R	NR	NR
8. BASIC	B.62	R	NR	NR	NR	NR
9. Assembler	B.62	R	R	R	-	-
10. Ladder Diagrams	B.62	R	R	R	R	R
11. Functional Blocks	B.62	R	R	R	R	R
12. Statement List	B.62	R	R	R	R	R

Requirements

1. At Software Safety Integrity Level 3 and 4, when a subset of languages 1, 2, 3 and 4 are used, the recommendation changes to HR.
2. For certain applications languages 7 and 9 may be the only ones available. At Software Safety Integrity Levels 3 and 4 where a highly recommended option is not available, it is strongly recommended that to raise the recommendation to 'R' there should be a subset of these languages and that there should be a precise set of coding standards.
3. For information on assessing the suitability of a programming language see entry in the bibliography for 'Suitable Programming Language', B.62.
4. If a specific language is not in the table, it is not automatically excluded. It should, however, conform to B.62.

Tableau A.16 – Modélisation (td5)
Référencé par l'article 13

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Diagramme des flux de données	B.12	-	R	R	R	R
2. Automates à états finis	B.29	-	HR	HR	HR	HR
3. Méthodes formelles	B.30	-	R	R	HR	HR
4. Modélisation des performances	B.45	-	R	R	HR	HR
5. Réseaux de Pétri temporels	B.64	-	HR	HR	HR	HR
6. Prototypage/Animation	B.49	-	R	R	R	R
7. Schémas de structure	B.59	-	R	R	HR	HR
Exigence Un ensemble de techniques adapté doit être choisi en fonction du niveau d'intégrité de la sécurité logicielle.						

Tableau A.17 – Tests de performance (td6)
Référencé par les articles 10, 12 et 13

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Tests en avalanche/en surcharge	B.3	-	R	R	HR	HR
2. Temps de réponse et contraintes de place mémoire	B.52	-	HR	HR	HR	HR
3. Exigences en matière de performance	B.46	-	HR	HR	HR	HR
Exigence Un ensemble de techniques adapté doit être choisi en fonction du niveau d'intégrité de la sécurité logicielle.						

Tableau A.18 – Méthodes semi-formelles (td7)
Référencé par les articles 8 et 10

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Bloc diagramme logique/fonction	-	R	R	R	HR	HR
2. Diagrammes de séquence	-	R	R	R	HR	HR
3. Diagrammes des flux de données	B.12	R	R	R	R	R
4. Automates à état finis/Schémas de transition d'état	B.29	-	R	R	HR	HR
5. Réseaux de Pétri temporels	B.64	-	R	R	HR	HR
6. Tables de décision/de vérité	B.14	R	R	R	HR	HR
Exigence Un ensemble de techniques adapté doit être choisi en fonction du niveau d'intégrité de la sécurité logicielle.						

Table A.16 – Modelling (dt5)
Referenced by clause 13

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Data Flow Diagrams	B.12	-	R	R	R	R
2. Finite State Machines	B.29	-	HR	HR	HR	HR
3. Formal Methods	B.30	-	R	R	HR	HR
4. Performance Modelling	B.45	-	R	R	HR	HR
5. Time Petri Nets	B.64	-	HR	HR	HR	HR
6. Prototyping/Animation	B.49	-	R	R	R	R
7. Structure Diagrams	B.59	-	R	R	HR	HR
Requirement A suitable set of techniques shall be chosen according to the software safety integrity level.						

Table A.17 –Performance Testing (dt6)
Referenced by clauses 10, 12 and 13

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Avalanche/Stress Testing	B.3	-	R	R	HR	HR
2. Response Timing and Memory Constraints	B.52	-	HR	HR	HR	HR
3. Performance Requirements	B.46	-	HR	HR	HR	HR
Requirement A suitable set of techniques shall be chosen according to the software safety integrity level.						

Table A.18 – Semi-Formal Methods (dt7)
Referenced by clauses 8 and 10

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Logic/Function Block Diagrams	-	R	R	R	HR	HR
2. Sequence Diagrams	-	R	R	R	HR	HR
3. Data flow Diagrams	B.12	R	R	R	R	R
4. Finite State Machines/State Transition Diagrams	B.29	-	R	R	HR	HR
5. Time Petri Nets	B.64	-	R	R	HR	HR
6. Decision/Truth Tables	B.14	R	R	R	HR	HR
Requirement A suitable set of techniques shall be chosen according to the software safety integrity level.						

Tableau A.19 – Analyse statique (td8)
Référencé par les articles 11 et 14

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Analyse des valeurs aux limites	B.4	-	R	R	HR	HR
2. Listes de contrôle	B.8	-	R	R	R	R
3. Analyse de flux de contrôle	B.9	-	HR	HR	HR	HR
4. Analyse du flux de données	B.11	-	HR	HR	HR	HR
5. Supposition d'erreurs	B.21	-	R	R	R	R
6. Inspection de Fagan	B.24	-	R	R	HR	HR
7. Analyse des chemins insidieux	B.55	-	-	-	R	R
8. Exécution symbolique	B.63	-	R	R	HR	HR
9. Revues/Examens de la conception	B.66	HR	HR	HR	HR	HR
Exigence Un ensemble de techniques adapté doit être choisi en fonction du niveau d'intégrité de la sécurité logicielle.						

Tableau A.20 – Approche modulaire (td9)
Référencé par l'article 10

TECHNIQUE/MESURE	Réf.	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Taille du module limitée	B.43	HR	HR	HR	HR	HR
2. Masquage d'informations/Encapsulage	B.36	R	HR	HR	HR	HR
3. Limite du nombre de paramètres	B.43	R	R	R	R	R
4. Un point d'entrée/un point de sortie dans les sous-programmes et les fonctions	B.43	R	HR	HR	HR	HR
5. Interface entièrement définie	B.43	HR	HR	HR	M	M
Exigence Un ensemble de techniques adapté doit être choisi en fonction du niveau d'intégrité de la sécurité logicielle.						

Table A.19 – Static Analysis (dt8)
Referenced by clauses 11 and 14

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Boundary Value Analysis	B.4	-	R	R	HR	HR
2. Checklists	B.8	-	R	R	R	R
3. Control Flow Analysis	B.9	-	HR	HR	HR	HR
4. Data Flow Analysis	B.11	-	HR	HR	HR	HR
5. Error Guessing	B.21	-	R	R	R	R
6. Fagan Inspections	B.24	-	R	R	HR	HR
7. Sneak Circuit Analysis	B.55	-	-	-	R	R
8. Symbolic Execution	B.63	-	R	R	HR	HR
9. Walkthroughs/Design Reviews	B.66	HR	HR	HR	HR	HR
Requirement A suitable set of techniques shall be chosen according to the software safety integrity level.						

Table A.20 – Modular Approach (dt9)
Referenced by clause 10

TECHNIQUE/MEASURE	Ref.	SWSIL 0	SWSIL 1	SWSIL 2	SWSIL 3	SWSIL 4
1. Module Size Limited	B.43	HR	HR	HR	HR	HR
2. Information Hiding/Encapsulation	B.36	R	HR	HR	HR	HR
3. Parameter Number Limit	B.43	R	R	R	R	R
4. One Entry/One Exit Point in Subroutines and Functions	B.43	R	HR	HR	HR	HR
5. Fully Defined Interface	B.43	HR	HR	HR	M	M
Requirement A suitable set of techniques shall be chosen according to the software safety integrity level.						

Annexe B
(informative)

Bibliographie des techniques

B.1 Intelligence artificielle – Correction des défauts (référencé par l'article 9)

Objectif

Etre capable de réagir aux dangers possibles d'une manière très souple en introduisant un mélange (une combinaison) de méthodes et de modèles de traitement, et un certain type d'analyse en ligne de la sécurité et de la fiabilité.

Description

En particulier, il est admis que la prévision des défauts (calcul des tendances), la correction des défauts et les actions de maintenance et de surveillance soient supportées de manière très efficace par des systèmes basés IA dans divers canaux d'un système, puisque les règles seraient éventuellement dérivées des spécifications et vérifiées par rapport à celles-ci. Il n'est pas exclu d'éviter efficacement, grâce à cette approche, certains défauts communs qui sont introduits dans les spécifications du fait d'avoir déjà implicitement présentes à l'esprit certaines règles de conception et d'implémentation, particulièrement lorsqu'on applique une combinaison de modèles et de méthodes d'une manière fonctionnelle ou descriptive.

Les méthodes sont sélectionnées de manière à permettre que les défauts soient corrigés et les effets des défaillances minimisés en vue de satisfaire à la sécurité et à la fiabilité recherchées.

B.2 Programmes analysables (référencé par l'article 10)

Objectif

Concevoir un programme de sorte que l'analyse du programme soit facilement réalisable. Le comportement du programme doit être complètement testable sur la base de l'analyse.

Description

L'objectif consiste à produire des programmes faciles à analyser à l'aide des méthodes d'analyse statique. Pour atteindre ce but, il convient de respecter les règles de la programmation structurée, par exemple:

– il convient que le flux de contrôle du module soit composé d'éléments structurés, autrement dit de séquences, d'itérations et de sélections;
– il convient que les modules soient petits;
– le nombre de chemins possibles via un module est faible;
– il faut que les parties du programme soient conçues de manière à être découplées autant que possible;
– il convient que la relation entre les paramètres d'entrée et de sortie soit aussi simple que possible;
– il convient de ne pas utiliser des calculs complexes comme base des décisions de branchements et de boucles;
– il convient que les décisions de branchements et de boucles soient liées aux paramètres d'entrée du module de façon simple;
– les limites entre les différents types d'organisation doivent être simples.

Annex B
(informative)

Bibliography of techniques

B.1 Artificial Intelligence – Fault Correction (referenced by clause 9)

Aim

To be able to react to possible hazards in a very flexible way by introducing a mix (combination) of methods and process models and some kind of on-line safety and reliability analysis.

Description

In particular fault forecasting (calculating trends), fault correction, maintenance and supervisory actions may be supported by AI-based systems in a very efficient way in diverse channels of a system, since the rules might be derived directly from the specifications and checked against these. Certain common faults which are introduced into specifications by implicitly already having some design and implementation rules in mind may be avoided effectively by this approach, especially when applying a combination of models and methods in a functional or descriptive manner.

The methods are selected such that faults may be corrected and the effects of failures be minimised, in order to meet the desired safety and reliability.

B.2 Analysable Programs (referenced by clause 10)

Aim

To design a program in a way that program analysis is easily feasible. The program behaviour must be testable completely on the basis of the analysis.

Description

The intention is to produce programs which are easy to analysis using static analysis methods. In order to achieve this, the rules of structured programming should be followed, for instance:

– the module control flow should be composed of structured constructs, that is sequences, iterations and selection;
– the modules should be small;
– the number of possible paths through a module is small;
– the individual program parts have to be designed so that they are decoupled as far as possible;
– the relation between the input and output parameters should be as simple as possible;
– complex calculations should not be used as the basis of branching and loop decisions;
– branch and loop decisions should be simply related to the module input parameters;
– boundaries between different types of mappings shall be simple.

Références:

Vérification – The Practical Problems. B.J.T. Webb et D.J. Mannering, SARSS 87, Nov. 1987, Altrincham, Angleterre, Elsevier Applied Science, ISBN 1-85166-167-0, 1987.

An Experience in Design and Validation of Software for a Reactor Protection System. S. Bologna, E. de Agostino *et al.*, IFAC Workshop, SAFECOMP 1979, Stuttgart, 16-18 mai 1979, Pergamon Press 1979.

B.3 Tests de surcharge (référencé par td6)

Objectif

Charger l'objet testé avec une charge de travail exceptionnellement élevée de manière à montrer qu'il résisterait facilement à des charges de travail normales.

Description

Il existe une variété de conditions de tests qui peuvent être appliquées aux tests de surcharge. Certaines de ces conditions de test sont énumérées ci-dessous:

– si vous travaillez en mode interrogation, l'objet testé reçoit beaucoup plus de changements d'entrée par unité de temps que dans des conditions normales;
– si vous travaillez à la demande, le nombre de demandes par unité de temps adressées à l'objet testé est augmenté au-delà des conditions normales;
– si la taille de la base de données joue un rôle important, elle est augmentée au-delà des conditions normales;
– les périphériques influents sont réglés sur leur vitesse maximum ou minimum respectivement;
– dans les cas extrêmes, tous les facteurs influents sont, dans la mesure du possible, réglés sur leurs conditions limites en même temps.

Dans ces conditions de test, le comportement temporel de l'objet testé peut être évalué. L'influence des variations de travail peut être observée. Le dimensionnement correct des tampons internes ou des variables dynamiques, des piles, etc., peut être vérifié.

B.4 Analyse des valeurs aux limites (référencé par td2, td3 et td8)

Objectif

Supprimer les erreurs logicielles se produisant aux limites ou frontières des paramètres.

Description

Le domaine d'entrée du programme est divisé en un nombre de catégories d'entrées. Il convient que les tests englobent les limites et les extrêmes des catégories. Les tests vérifient que les limites du domaine d'entrée de la spécification coïncident avec celles du programme. L'utilisation de la valeur zéro, dans une traduction directe aussi bien qu'indirecte est souvent susceptible d'erreur, elle exige une attention spéciale:

– diviseur par zéro;
– caractères ASCII blancs;
– pile ou élément de liste vide;
– matrice nulle;
– entrée de table nulle.

References:

Verification – The Practical Problems. B.J.T. Webb and D.J. Mannering, SARSS 87, Nov. 1987, Altrincham, England, Elsevier Applied Science, ISBN 1-85166-167-0, 1987.

An Experience in Design and Validation of Software for a Reactor Protection System. S. Bologna, E. de Agostino *et al.*, IFAC Workshop, SAFECOMP 1979, Stuttgart, 16-18 May 1979, Pergamon Press 1979.

B.3 Avalanche/Stress Testing (referenced by dt6)

Aim

To burden the test object with an exceptionally high workload in order to show that the test object would stand normal workloads easily.

Description

There are a variety of test conditions which can be applied for avalanche/stress testing. Some of these test conditions are listed below:

– if working in a polling mode then the test object gets much more input changes per time unit as under normal conditions;
– if working on demand then the number of demands per time unit to the test object is increased beyond normal conditions;
– if the size of a database plays an important role then it is increased beyond normal conditions;
– influential devices are tuned to their maximum speed or lowest speed respectively;
– for extreme cases, all influential factors, as far as possible, are put to the boundary conditions at the same time.

Under these test conditions the time behaviour of the test object can be evaluated. The influence of load changes can be observed. The correct dimension of internal buffers or dynamic variables, stacks etc., can be checked.

B.4 Boundary Value Analysis (referenced by dt2, dt3 and dt8)

Aim

To remove software errors occurring at parameter limits or boundaries.

Description

The input domain of the program is divided into a number of input classes. The tests should cover the boundaries and extremes of the classes. The tests check that the boundaries in the input domain of the specification coincide with those in the program. The use of the value zero, in a direct as well as in an indirect translation, is often error-prone and demands special attention:

– zero divisor;
– blank ASCII characters;
– empty stack or list element;
– null matrix;
– zero table entry.

En principe, les limites de l'entrée ont une correspondance directe avec les limites de la gamme de sortie. Il convient d'écrire des scénarios de test de manière à forcer la sortie dans ses valeurs limites. Examiner aussi, s'il est possible de spécifier un scénario de test dans lequel la sortie dépasserait les valeurs limites de la spécification.

Si la sortie est une séquence de données, par exemple un tableau imprimé, il convient d'apporter une attention toute spéciale au premier et au dernier élément et aux listes contenant aucun, 1 et 2 éléments.

Référence:

The Art of Software Testing. G. Myers, Wiley et Sons, New York, Etats-Unis, 1979.

B.5 Rattrapage par régression (référencé par l'article 9)

Objectif

Permettre une exploitation fonctionnelle correcte en présence d'un ou de plusieurs défauts.

Description

Si un défaut a été détecté, le système est réinitialisé sur un état interne précédent dont la cohérence a déjà été éprouvée. Cette méthode implique une sauvegarde fréquente de l'état interne à ce que l'on appelle des points de reprises bien définis. Il est permis d'effectuer cette opération de façon globale (pour la base de données complète) ou incrémentale (seulement les changements entre les points de reprise). Il faut alors que le système compense les changements qui se sont produits dans l'intervalle en utilisant la journalisation (trace des actions), la compensation (tous les effets de ces changements sont annulés) ou l'interaction (manuelle) externe.

B.6 Schémas de cause et de conséquence (référencé par l'article 14 et td3)

Objectif

Modéliser, sous forme de schémas, la séquence d'événements pouvant se développer dans un système comme conséquence d'une combinaison d'événements de base.

Description

Cette technique peut être considérée comme une combinaison de l'analyse de l'arbre des causes et de l'arbre des événements. Commençant à partir d'un événement critique, un graphe de cause et de conséquence est tracé vers l'arrière et vers l'avant. Dans la direction arrière, il équivaut à un arbre des causes avec l'événement critique figurant tout en haut. Dans la direction avant, les conséquences possibles de cet événement sont identifiées. Le graphe peut contenir des symboles de noeuds qui décrivent les conditions de propagation le long des différents branches partant du noeud. Des temporisations peuvent également être incluses. Ces conditions peuvent aussi être décrites à l'aide des arbres des causes. Les lignes de propagation peuvent être combinées à des symboles logiques afin de rendre le schéma plus compact. Un ensemble de symboles standards est défini pour être utilisé dans les schémas de cause et de conséquence. Les schémas peuvent être utilisés pour calculer la probabilité d'une occurrence de certaines conséquences critiques.

Référence:

The Cause Consequence Diagram Method as a Basis for Quantitative Accident Analysis. D.S. Nielsen, Riso-M-1374, 1971.

Normally the boundaries for input have a direct correspondence to the boundaries for the output range. Test cases should be written to force the output to its limited values. Consider also if it is possible to specify a test case which causes output to exceed the specification boundary values.

If output is a sequence of data, for example a printed table, special attention should be paid to the first and the last elements and to lists containing none, 1 and 2 elements.

Reference:

The Art of Software Testing. G. Myers, Wiley and Sons, New York, USA, 1979.

B.5 Backward Recovery (referenced by clause 9)

Aim

To provide correct functional operation in the presence of one or more faults.

Description

If a fault has been detected, the system is reset to an earlier internal state, the consistency of which has been proven before. This method implies saving of the internal state frequently at so-called well-defined checkpoints. This may be done globally (for the complete database) or incrementally (changes only between checkpoints). Then the system has to compensate for the changes which have taken place in the meantime by using journalling (audit trail of actions), compensation (all effects of these changes are nullified) or external (manual) interaction.

B.6 Cause Consequence Diagrams (referenced by clause 14 and dt3)

Aim

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

Description

It can be regarded as a combination of fault tree and event-tree analysis. Starting from a critical event, a cause consequence graph is traced backwards and forwards. In the backwards direction it is equivalent to a fault tree with the critical event as the given top event. In the forward direction the possible consequences arising from an event are identified. The graph can contain vertex symbols which describe the conditions for propagation along different branches from the vertex. Time delays can also be included. These conditions can also be described with fault trees. The lines of propagation can be combined with logical symbols, to make the diagram more compact. A set of standard symbols are defined for use in cause consequence diagrams. The diagrams can be used to compute the probability of occurrence of certain critical consequences.

Reference:

The Cause Consequence Diagram Method as a Basis for Quantitative Accident Analysis. D.S. Nielsen, Riso-M-1374, 1971.

B.7 Outils certifiés et compilateurs certifiés (référencé par l'article 10)

Objectif

Des outils sont nécessaires pour aider les développeurs dans les différentes phases du développement logiciel. Il convient, chaque fois que possible, que les outils soient certifiés de sorte que l'on puisse accorder un certain degré de confiance dans la validité des données en sortie.

Description

Un outil certifié est un outil qui a été déterminé comme ayant une qualité particulière. La certification d'un outil sera généralement effectuée par un organisme indépendant, souvent national, utilisant des critères définis indépendamment, en principe des normes nationales ou internationales. Il convient, idéalement, que les outils utilisés dans toutes les phases de développement (définition, conception, codage, tests et validation) et ceux utilisés dans la gestion de configuration fassent l'objet d'une certification. A ce jour, seuls les compilateurs (traducteurs) sont régulièrement soumis aux procédures de certification fixées par des organismes nationaux de certification qui testent les compilateurs (traducteurs) par rapport aux normes internationales. les compilateurs pour Ada et Pascal, par exemple.

Références:

Pascal Validation Suite. Distributeur au Royaume-Uni: BSI Quality Assurance, PO Box 375, Milton Keynes, MK14 6LL.

Ada Validation Suite. Distributeur au Royaume-Uni: National Computing Centre (NCC), Oxford Road. Manchester, Angleterre.

B.8 Listes de contrôle (référencé par l'article 14 et td8)

Objectif

Stimuler l'évaluation critique sur tous les aspects du système plutôt que d'imposer des prescriptions spécifiques.

Description

Ensemble de questions qui doit être établi par la personne élaborant la liste de contrôle. La plupart des questions sont d'ordre général et il faut que le Chargé d'Evaluation les interprète de la façon la mieux adaptée au système spécifique en cours d'évaluation.

Pour tenir compte de la grande diversité des logiciels et des systèmes qui doivent être validés, la plupart des listes de contrôle contiennent des questions applicables à de nombreux types de systèmes. En conséquence, il n'est pas exclu que certaines questions figurant sur la liste ne soient pas applicables au système en question. Si tel est le cas, elles doivent être ignorées. De la même façon, il est admis que, pour un système spécifique, il soit nécessaire de compléter la liste de contrôle standard par des questions directement adaptées au système en question.

En tout cas, il convient qu'il soit clair que l'utilisation des listes de contrôle dépend surtout de la compétence et du jugement de l'ingénieur qui sélectionne et met la liste de contrôle en application. En conséquence, il convient que les décisions prises par l'ingénieur par rapport à la liste de contrôle sélectionnée (aux listes de contrôle sélectionnées) et à toute question complémentaire ou superflue soient entièrement documentées et justifiées. L'objectif consiste à assurer que l'application des listes de contrôle puisse être passée en revue et que les mêmes résultats soient obtenus à moins que des critères différents ne soient utilisés.

B.7 Certified Tools and Certified Translators (referenced by clause 10)

Aim

Tools are necessary to help developers in the different phases of software development. Wherever possible, tools should be certified so that some level of confidence can be assumed regarding the correctness of the outputs.

Description

A certified tool is one which has been determined to be of a particular quality. The certification of a tool will generally be carried out by an independent, often national, body, against independently set criteria, typically national or international standards. Ideally, the tools used in all development phases (definition, design, coding, testing and validation) and those used in configuration management, should be subject to certification. To date only compilers (translators) are regularly subject to certification procedures; these are laid down by national certification bodies and they exercise compilers (translators) against international standards. such as those for Ada and Pascal.

References:

Pascal Validation Suite. UK distributor: BSI Quality Assurance, PO Box 375, Milton Keynes, MK14 6LL.

Ada Validation Suite. UK distributor: National Computing Centre (NCC), Oxford Road., Manchester, England.

B.8 Checklists (referenced by clause 14 and dt8)

Aim

To provide a stimulus to critical appraisal of all aspects of the system rather than to lay down specific requirements.

Description

A set of questions to be completed by the person performing the checklist. Many of the questions are of a general nature and the Assessor must interpret them as seems most appropriate to the particular system being assessed.

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

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

Lorsque l'on complète une liste de contrôle, l'objectif est d'être aussi concis que possible. Quand une justification approfondie s'avère nécessaire, il convient de l'apporter par renvoi à une documentation complémentaire. Il convient d'utiliser des mentions telles que Satisfaisant, Insuffisant, Peu concluant ou un ensemble limité de symboles similaires pour enregistrer les résultats de chaque question. Cette concision simplifie beaucoup le processus permettant d'arriver à une conclusion globale à partir des résultats de l'évaluation de la liste de contrôle.

Références:

Programmable Electronic Systems in Safety Related Applications. Health and Safety Executive, Her Majesty's Stationary Office, Londres 1987.

Guidelines for the Assessment of the Safety and Reliability of High Integrity Industrial Computer Systems. EWICS TC7, WP 489, 6 octobre 1987.

The Art of Software Testing. G. Myers, Wiley et Sons, New York, Etats-Unis, 1979.

Logiciel pour les calculateurs utilisés dans les systèmes de sûreté des centrales nucléaires. CEI 60880, 1986.

B.9 Analyse de flux de contrôle (référencé par td8)

Objectif

Détecter des structures faibles et potentiellement incorrectes dans le programme.

Description

L'analyse de flux de contrôle identifie les zones suspectes du code qui ne respectent pas les bonnes pratiques de programmation. Le programme est analysé pour former un graphe orienté qui peut être analysé pour

– un code inaccessible, par exemple, branchements inconditionnels qui rendent les blocs de code inaccessibles;
– un code arborescent, qui est un code bien structuré dont le graphe de contrôle est réductible en un simple noeud par réductions de graphes successifs. Un code faiblement structuré ne peut être réduit qu'à une nodosité composée de plusieurs noeuds.

Références:

RXVP80 – The Verification and Validation System for FORTRAN. Users Manual. General Research Corporation, Santa Barbara, Californie, Etats-Unis.

Information Flow and Data Flow of While Programs. J.F. Bergeretti et B.A. Carre, ACMTrans. on Prog. Lang. and Syst., 1985.

B.10 Analyse des défaillances de mode commun (référencé par l'article 14)

Objectif

Identifier dans les systèmes ou les sous-systèmes redondants les défaillances potentielles qui amoindriraient les avantages de la redondance en raison de l'apparition simultanée des mêmes défaillances dans les parties redondantes.

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

References:

Programmable Electronic Systems in Safety Related Applications. Health and Safety Executive, Her Majesty's Stationary Office, London 1987.

Guidelines for the Assessment of the Safety and Reliability of High Integrity Industrial Computer Systems. EWICS TC7, WP 489, 6th October 1987.

The Art of Software Testing. G. Myers, Wiley and Sons, New York, USA, 1979.

Software for computers in the safety systems of nuclear power stations. IEC 60880, 1986.

B.9 Control Flow Analysis (referenced by dt8)

Aim

To detect poor and potentially incorrect program structures.

Description

Control Flow Analysis identifies suspect areas of code which do not follow good programming practice. The program is analysed to form a directed graph which can be analysed for

– inaccessible code, for instance, unconditional jumps which leaves blocks of code unreachable;

– knotted code, which is well-structured code whose control graph is reducible by successive graph reductions to a single node. Poorly structured code can only be reduced to a knot composed of several nodes.

References:

RXVP80 – The Verification and Validation System for FORTRAN. Users Manual. General Research Corporation, Santa Barbara, California, USA.

Information Flow and Data Flow of While Programs. J.F. Bergeretti and B.A. Carre, ACM Trans. on Prog. Lang. and Syst., 1985.

B.10 Common Cause Failure Analysis (referenced by clause 14)

Aim

To identify potential failures in redundant systems or redundant subsystems which would undermine the benefits of redundancy because of the appearance of the same failures in the redundant parts at the same time.

Description

Les systèmes informatiques conçus pour protéger la sécurité d'une unité de production utilisent souvent la redondance dans le matériel et la décision à la majorité. Cette technique est utilisée pour éviter les défaillances aléatoires des composants qui risqueraient d'empêcher le traitement correct des données dans un système informatique.

Certaines défaillances, pourtant, peuvent être communes à plusieurs composants. Par exemple, si un système informatique est installé dans une même salle, des imperfections dans la climatisation risqueraient éventuellement de réduire les avantages de la redondance. Il en va de même pour des effets externes sur le système informatique tels que les incendies, les inondations, les interférences électromagnétiques, les accidents d'avion et les tremblements de terre. Il n'est pas exclu, en outre, que le système informatique soit affecté par des incidents liés à l'exploitation et à la maintenance. Il est donc essentiel de proposer des procédures adéquates et bien documentées pour l'exploitation et la maintenance. La formation approfondie du personnel d'exploitation et de maintenance est également essentielle.

Les effets internes contribuent aussi largement aux défaillances ayant une cause commune (CCF). Elles peuvent découler d'erreurs de conception dans des composants communs ou identiques et dans leurs interfaces, aussi bien que du vieillissement des composants. L'analyse CCF doit examiner le système à la recherche de défaillances potentielles communes. Les méthodes d'analyse CCF sont le contrôle général de la qualité, les examens de la conception, la vérification et les tests effectués par une équipe indépendante et l'analyse des incidents réels avec le retour d'expérience acquise sur des systèmes similaires. Cependant, le domaine d'application de l'analyse va au-delà du matériel. Même si des «logiciels variés» sont utilisés dans les chaînes complexes d'un système informatique redondant, il n'est pas exclu qu'un certain degré de standardisation dans les approches logicielles puisse éventuellement donner lieu à une analyse CCF. Il pourrait s'agir, par exemple, d'erreurs dans la spécification commune.

Quand les défaillances ayant une cause commune ne se produisent pas exactement en même temps, des précautions peuvent être prises en utilisant des méthodes de comparaison entre les chaînes redondantes qui peuvent mener à la détection d'une défaillance avant que celle-ci ne devienne commune à toutes les chaînes. Il convient que l'analyse CCF prenne en compte cette technique.

Références:

Review of Common Cause Failures. I.A. Watson, UKAEA, Centre for Systems Reliability, Wigshaw Lane, WA3 4NE, Angleterre, NCSR R 27, juillet 1981.

Common-Mode Failures in Redundancy Systems. I.A. Watson et G.T. Edwards Nuclear Technology Vol. 46, De. 1979.

Programmable Electronic Systems in Safety Related Applications. Health and Safety Executive, Her Majesty's Stationary Office.

B.11 Analyse du flux de données (référencé par td8)

Objectif

Détecter des structures faibles et potentiellement incorrectes dans le programme.

Description

Computer systems intended to take care of the safety of a plant often use redundancy in hardware and majority voting. This technique is used to avoid random component failures, which would tend to prevent the correct processing of data in a computer system.

However, some failures can be common to more than one component. For example, if a computer system is installed in one single room, shortcomings in the air-conditioning might reduce the benefits of redundancy. The same is true for other external effects on the computer system such as: fire, flooding, electromagnetic interference, plane crashes, and earthquakes. The computer system may also be affected by incidents related to operation and maintenance. It is essential, therefore, that adequate and well-documented procedures are provided for operation and maintenance. Extensive training of operating and maintenance personnel is also essential.

Internal effects are also major contributors to Common Cause Failures (CCF). They can stem from design errors in common or identical components and their interfaces, as well as ageing of components. CCF-Analysis has to search the system for such potential common failures. Methods of CCF-Analysis are general quality control, design reviews, verification and testing by an independent team, and analysis of real incidents with feedback of experience from similar systems. The scope of the analysis, however, goes beyond hardware. Even if 'diverse software' is used in difficult chains of a redundant computer system, there might be some commonality in the software approaches which could give rise to CCF (errors in the common specification, for example).

When CCFs do not occur exactly at the same tine, precautions can be taken by means of comparison methods between the redundant chains which should lead to detection of a failure before this failure is common to all chains. CCF analysis should take this technique into account.

References:

Review of Common Cause Failures. I.A. Watson, UKAEA, Centre for Systems Reliability, Wigshaw Lane, WA3 4NE, England, NCSR R 27, July 1981.

Common-Mode Failures in Redundancy Systems. I.A. Watson and G.T. Edwards Nuclear Technology Vol. 46, De. 1979.

Programmable Electronic Systems in Safety Related Applications. Health and Safety Executive, Her Majesty's Stationary Office.

B.11 Data Flow Analysis (referenced by dt8)

Aim

To detect poor and potentially incorrect program structures.

Description

L'analyse du flux de données combine l'information obtenue à partir de l'analyse du flux de contrôle à l'information concernant les variables qui sont lues ou écrites dans différentes portions du code. L'analyse peut rechercher

– des variables qui sont lues avant d'être écrites. il est fort probable qu'il s'agisse d'une erreur et en tout cas d'une mauvaise pratique de programmation;
– des variables qui sont écrites plusieurs fois sans être lues, ce qui pourrait indiquer qu'un code a été omis;
– des variables qui sont écrites mais jamais lues, ce qui pourrait indiquer qu'un code est redondant.

Il existe une extension de l'analyse du flux de données, connue sous le nom d'analyse du flux d'informations dans laquelle les flux de données réels (à la fois à l'intérieur et entre les procédures) sont comparés au projet de conception. Cette analyse est, en principe, implémentée par un outil informatisé dans lequel les flux de données projetés sont définis à l'aide d'un commentaire structuré qui peut être lu par l'outil.

Références:

RXVP80 – The Verification and Validation System for FORTRAN. Users Manual. General Research Corporation, Santa Barbara, Californie, Etats-Unis.

Information Flow and Data Flow of While Programs. J.F. Bergeretti et B.A. Carre, ACMTrans. on Prog. Lang. and Syst., 1985.

B.12 Diagrammes des flux de données (référencé par td5 et td7)

Objectif

Décrire le flux de données dans un programme sous forme de schéma.

Description

Les organigrammes de données documentent la façon dont les données d'entrée sont transformées en sorties, chaque étape du schéma représentant une transformation distincte.

Les organigrammes de données sont constitués de trois composants:

1. des flèches annotées – représentent le flux de données en entrée et en sortie des centres de transformation avec des annotations spécifiant le type de données dont il s'agit;
2. des bulles annotées – représentent les centres de transformation avec des annotations spécifiant la transformation;
3. opérateurs (AND, XOR) – Ces opérateurs sont utilisés pour relier les flèches annotées.

Les organigrammes de données décrivent la façon dont une entrée est transformée en sortie. Comme il convient, ils ne contiennent pas des informations de contrôle ou des informations de séquencement. Chaque bulle peut être considérée comme une boîte noire autonome qui, dès que ses entrées sont disponibles, les transforme en ses sorties.

L'un des principaux avantages des organigrammes de données est qu'ils montrent les transformations sans émettre aucune hypothèse sur la manière dont ces transformations sont mises en oeuvre.

Description

Data Flow Analysis combines the information obtained from the control flow analysis with information about which variables are read or written in different portions of code. The analysis can check for

– variables that are read before they are written. This is very likely to be an error, and is certainly bad programming practice;

– variables that are written more than once without being read. This could indicate omitted code;

– variables that are written but never read. This could indicate redundant code.

There is an extension of data flow analysis known as information flow analysis, where the actual data flows (both within and between procedures) are compared with the design intent. This is normally implemented with a computerised tool where the intended data flows are defined using a structured comment that can be read by the tool.

References:

RXVP80 – The Verification and Validation System for FORTRAN. Users Manual. General Research Corporation, Santa Barbara, California, USA.

Information Flow and Data Flow of While Programs. J.F. Bergeretti and B.A. Carre, ACMTrans. on Prog. Lang. and Syst., 1985.

B.12 Data Flow Diagrams (referenced by dt5 and dt7)

Aim

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

Description

Data Flow Diagrams document how data input is transformed to output, with each stage in the diagram representing a distinct transformation.

Data flow diagrams are made up of three components:

1. annotated arrows – represent data flow in and out of the transformation centres with the annotations specifying what the data is;
2. annotated bubbles – represent transformation centres with the annotation specifying the transformation;
3. operators (AND, XOR) – these operators are used to link the annotated arrows.

Data flow diagrams describe how an input is transformed to an output. They do not, and should not, include control information or sequencing information. Each bubble can be considered as a stand-alone black box which, as soon as its inputs are available, transforms them to its outputs.

One of the principle advantages of data flow diagrams is that they show transformations without making any assumptions about how these transformations are implemented.

La meilleure approche vis-à-vis de la préparation des organigrammes de données est de prendre en considération les entrées système et d'évoluer vers les sorties système. Il faut que chaque bulle représente une transformation distincte – il convient que sa sortie soit, d'une certaine façon, différente de son entrée. Il n'existe aucune règle permettant de déterminer la structure globale du schéma et la construction d'un organigramme de données est l'un des aspects créatifs de la conception système. A l'instar de toute conception, il s'agit d'un processus itératif composé de tentatives précoces affinées par étapes pour produire un schéma final.

Référence:

Software Engineering. I. Sommerville, Addison-Wesley 1982. ISBN 0-201-13795-X.

B.13 Enregistrement et analyse des données (référencé par les articles 10 et 16)

Objectif

Enregistrer toutes les données, décisions et raisonnements d'un projet logiciel pour faciliter la vérification, la validation, l'évaluation et la maintenance.

Description

Des comptes rendus détaillés qui concernent à la fois le projet lui-même et les questions individuelles sont actualisés pendant un projet. Par exemple, un ingénieur est tenu de conserver des comptes rendus pouvant inclure

– l'effort développé sur les modules individuels,
– les tests effectués sur chaque module,
– les décisions et le raisonnement les sous-tendant,
– la réalisation des jalons du projet,
– les problèmes et leurs solutions.

Pendant le projet et au moment de sa conclusion, ces comptes rendus peuvent être analysés pour établir une grande variété d'informations. En particulier, l'enregistrement des données est très important pour la maintenance de systèmes informatiques puisque le raisonnement sous-tendant certains décisions prises au cours du projet de développement n'est pas toujours connu des ingénieurs de maintenance.

B.14 Tables de décision (Tables de vérité) (référencé par l'article 14 et td7)

Objectif

Fournir une spécification et une analyse claires et cohérentes des relations et combinaisons logiques complexes.

Description

Ces méthodes connexes utilisent des tables à deux dimensions pour décrire de façon concise les relations logiques entre les variables booléennes du programme.

La concision et la nature tabulaire des deux méthodes en font des moyens parfaitement adaptés pour l'analyse des combinaisons logiques complexes exprimées dans un code.

Les deux méthodes sont potentiellement exécutables si elles sont utilisées comme spécifications.

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

Reference:

Software Engineering. I. Sommerville, Addison-Wesley 1982. ISBN 0-201-13795-X.

B.13 Data Recording and Analysis (referenced by clauses 10 and 16)

Aim

To record all data, decisions and rationale in the software project to allow for easier verification, validation, assessment and maintenance.

Description

Detailed records are maintained during a project, both on a project and individual basis. For instance, an engineer would be required to keep records which could include

– effort expanded on individual modules,
– testing performed on each module,
– decisions and their rationale,
– achievement of project milestones,
– problems and their solutions.

During and at the conclusion of the project these records can be analysed to establish a wide variety of information. In particular, data recording is very important for the maintenance of computer systems as the rationale for certain decisions made during the development project is not always known by the maintenance engineers.

B.14 Decision Tables (Truth Tables) (referenced by clause 14 and dt7)

Aim

To provide a clear and coherent specification and analysis of complex logical combinations and relationships.

Description

These related methods use two dimensional tables to concisely describe logical relationships between Boolean program variables.

The conciseness and tabular nature of both methods makes them appropriate as a means of analysing complex logical combinations expressed in code.

Both methods are potentially executable if used as specifications.

B.15 Programmation défensive (référencé par l'article 9)

Objectif

Générer des programmes qui détectent un flux de contrôle, un flux de données ou des valeurs de données anormaux pendant leur exécution et qui réagissent envers eux d'une manière prédéterminée et acceptable.

Description

Beaucoup de techniques peuvent être utilisées au cours de la programmation pour rechercher les anomalies du contrôle ou des données. Ces techniques peuvent être appliquées systématiquement tout au long de la programmation d'un système pour diminuer la vraisemblance d'un traitement erroné des données.

On peut identifier deux zones de chevauchement des techniques défensives.

Le logiciel intrinsèquement protégé contre les erreurs est conçu pour prendre en charge ses propres imperfections de conception. Il n'est pas exclu que celles-ci soient provoquées par une simple erreur de conception ou de codage ou par des prescriptions erronées. Certaines techniques défensives sont énumérées ci-dessous:

– il est recommandé de vérifier l'étendue des variables;
– chaque fois que possible, il convient de vérifier la plausibilité des valeurs;
– il convient de vérifier le type, la dimension et l'étendue des paramètres des procédures à l'entrée de la procédure.

Ces trois premières recommandations aident à assurer que les nombres manipulés par le programme sont fondés, à la fois en termes de fonction du programme et de signification physique des variables.

Il convient de séparer les paramètres en lecture seule et en lecture-écriture et de vérifier leur accès. Il convient que les fonctions traitent tous les paramètres comme s'ils étaient en lecture seule. Il convient que les constantes littérales ne soient pas accessibles en écriture. Ceci aide à détecter un écrasement accidentel ou une utilisation erronée des variables.

Un logiciel tolérant aux erreurs est conçu pour «s'attendre» à des défaillances dans son propre environnement ou à un emploi hors des conditions nominales ou prévues et se comporter d'une manière prédéfinie. Ces techniques incluent ce qui suit:

– il convient de vérifier la plausibilité des variables d'entrée et des variables intermédiaires avec la signification physique;
– il convient de vérifier l'effet des variables de sortie, de préférence par l'observation directe des changements d'état du système associé;
– il convient que le logiciel vérifie sa propre configuration. Ceci peut inclure à la fois l'existence et l'accessibilité du matériel attendu et aussi l'état complet du logiciel lui-même. Ces vérifications sont particulièrement importantes pour conserver l'intégrité après des procédures de maintenance.

Certaines de ces techniques de programmation défensive telles que la vérification des séquences du flux de contrôle traitent également les défaillances externes.

B.15 Defensive Programming (referenced by clause 9)

Aim

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

Description

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

Two overlapping areas of defensive techniques can be identified.

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

– variables should be range checked;
– where possible, values should be checked for plausibility;
– parameters to procedures should be type, dimension and range checked at procedure entry.

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

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

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

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

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

Références:

Dependability of Critical Computer Systems – Partie 1. E.F. Redmill (ed) Elsevier Applied Science 1988.

Dependability of Critical Computer systems – Partie 2. E.F. Redmill (ed) Elsevier Applied Science 1988.

Software Engineering Aspects of Real-time Programming Concepts. E. Schoitsch, Computer Physics Communications 41 (1986) North Holland, Amsterdam.

B.16 Normes de conception et de codage (référencé par td1)

Objectif

Assurer une présentation uniforme des documents de conception et du code généré, imposer une programmation homogène et une méthode de conception standard.

Description

Les participants conviennent des règles à respecter dès le commencement du projet. Pour le moins, ces règles doivent inclure les éléments suivants:

– la méthode de développement et les normes associées de codage à respecter, par exemple JSP, MASCOT, Réseaux de Pétri, etc.;
– les détails de la modularisation, par exemple, la forme des interfaces, la taille des modules;
– l'utilisation de l'encapsulage, l'héritage et la polymorphie dans le cas des langages orientés objet;
– l'utilisation ou le rejet de certaines structures de langage, par exemple GOTO, EQUIVALENCE, les objets dynamiques, les données dynamiques, la récursivité, les pointeurs, les sorties, etc.; et
– la définition de l'objet et du moment du commentaire ainsi que la manière de commenter.

Ces règles sont élaborées pour faciliter le développement, la vérification, l'évaluation et la maintenance. Elles doivent donc désigner les outils disponibles, en particulier les analyseurs et les outils de réingénierie.

B.17 Programmation diversifiée (référencé par l'article 9)

Objectif

Détecter et masquer les défauts résiduels de conception du logiciel pendant l'exécution d'un programme de manière à empêcher les défaillances critiques de sécurité du système et à continuer en direction d'une plus grande fiabilité.

Description

Dans la programmation diversifiée, une spécification de programme donnée est implémentée N fois de diverses manières. Les mêmes valeurs d'entrée sont données aux N versions et les résultats produits par les N versions sont comparés. Le résultat, s'il est considéré valide, est transmis aux sorties informatiques.

Les N versions peuvent tourner en parallèle sur des ordinateurs distincts, ou encore toutes les versions peuvent être exécutées sur le même ordinateur et les résultats soumis à une décision interne. Différentes stratégies de décisions peuvent être utilisées sur les N versions selon les prescriptions de l'application.

References:

Dependability of Critical Computer Systems – Part 1. E.F. Redmill (ed) Elsevier Applied Science 1988.

Dependability of Critical Computer systems – Part 2. E.F. Redmill (ed) Elsevier Applied Science 1988.

Software Engineering Aspects of Real-time Programming Concepts. E. Schoitsch, Computer Physics Communications 41 (1986) North Holland, Amsterdam.

B.16 Design and Coding Standards (referenced by dt1)

Aim

To ensure a uniform layout of the design documents and the produced code, enforce egoless programming and to enforce a standard design method.

Description

The rules to be adhered to are agreed at the outset of the project between the participants. These rules must comprise as a minimum the following items:

– the development method and the related coding standards to be followed, for example JSP, MASCOT, Petri-Nets, etc;
– details of modularisation, for example, interface shapes, module sizes;
– use of encapsulation, inheritance and polymorphy, in case of object-oriented languages;
– use or avoidance of certain language constructs, for example GOTO, EQUIVALENCE, dynamic objects, dynamic data, recursion, pointers, exits, etc.; and
– the what, where and how to comment.

These rules are made to allow for ease of development, verification, assessment and maintenance. Therefore, they shall address the available tools, in particular analysers and reverse engineering tools.

B.17 Diverse Programming (referenced by clause 9)

Aim

Detect and mask residual software design faults during execution of a program, in order to prevent safety critical failures of the system, and to continue operation for high reliability.

Description

In diverse programming a given program specification implemented N times in different ways. The same input values are given to the N versions, and the results produced by the N versions are compared. If the result is considered to be valid, the result is transmitted to the computer outputs.

The N versions can run in parallel on separate computers, alternatively all versions can be run on the same computer and the results subjected to an internal vote. Different voting strategies can be used on the N versions depending on the application requirements.

Si le système a un état sûr, il est possible d'exiger l'accord complet (les N versions s'accordent); dans le cas contraire, une valeur intrinsèquement sûre de la sortie est utilisée. Pour les systèmes à simple déclenchement, la décision peut être influencée, en faveur de la sécurité. Dans ce cas, la sécurité serait de déclencher le système, si l'une ou l'autre version l'exigeait. En principe, cette approche n'utilise que deux versions (N=2).

Pour les systèmes sans état sûr, les stratégies de décision à la majorité peuvent être employées. Dans les cas où il n'existe aucun accord collectif, les approches probabilistes peuvent être utilisées afin de maximiser la chance de sélection de la valeur correcte, par exemple en prenant la valeur moyenne, par le maintien en l'état temporaire des sorties jusqu'au retour de l'accord, etc.

Cette technique n'élimine pas les défauts résiduels de conception du logiciel, mais propose une mesure permettant de les détecter et de les masquer avant qu'ils puissent avoir une incidence sur la sécurité.

Références:

Dependable Computing: From Concepts to Design Diversity. A. Avizienis, J.C. Laprie, Proc. IEEE, Vol. 74, 5, mai 1986.

A Theoretical Basis for the Analysis of Multi-version Software subject to Co-incident Failures. D.E. Eckhardt et L.D. Lee, IEEE Trans. SE-11, No. 12, 1985.

B.18 Reconfiguration dynamique (référencé par l'article 9)

Objectif

Conserver la fonctionnalité du système malgré un défaut interne.

Description

Il faut que l'architecture logique du système soit telle qu'elle puisse être configurée sur un sous-ensemble des ressources disponibles du système. Il est nécessaire que l'architecture soit capable de détecter une défaillance dans une ressource physique, puis de reconfigurer l'architecture logique sur les ressources limitées qui fonctionnent toujours. Bien que le concept soit plus traditionnellement limité au rétablissement pour cause d'unités matérielles défaillantes, il est également applicable à des unités logicielles défaillantes s'il y a «une redondance d'exécution» suffisante pour permettre une nouvelle tentative du logiciel ou s'il existe des données redondantes suffisantes pour que la défaillance individuelle et isolée revête une faible importance.

Bien que traditionnellement appliquée au matériel, cette technique est en cours de développement pour être appliquée au logiciel et donc au système dans son ensemble. Il faut qu'elle soit prise en considération dès la première étape de conception du système.

Références:

Critical Issues in the Design of a Reconfigureable Control Computer. H. Schmid, J. Lam, R. Naro et K. Weir, FTCS 14 juin 1984, IEEE 1984.

Assigning Processes to Processors: A Fault-tolerant Approach. juin 1984 G. Kar et C.N. Nikolaou, Watson Research Centre, Yorktown.

If the system has a safe state, then it is feasible to demand complete agreement (all N agree) otherwise a fail-safe output value is used. For simple trip systems the vote can be biased in the safe direction. In this case the safe action would be to trip if either version demanded a trip. This approach typically uses only two versions (N=2).

For systems with no safe state, majority voting strategies can be employed. For cases where there is no collective agreement, probabilistic approaches can be used in order to maximise the chance of selecting the correct value, for example, taking the middle value, temporary freezing of outputs until agreement returns, etc.

This technique does not eliminate residual software design faults, but it provides a measure to detect and mask before they can affect safety.

References:

Dependable Computing: From Concepts to Design Diversity. A. Avizienis, J.C. Laprie, Proc. IEEE, Vol. 74, 5, May 1986.

A Theoretical Basis for the Analysis of Multi-version Software subject to Co-incident Failures. D.E. Eckhardt and L.D. Lee, IEEE Trans. SE-11, No. 12, 1985.

B.18 Dynamic Reconfiguration (referenced by clause 9)

Aim

To maintain system functionality despite an internal fault.

Description

The logical architecture of the system has to be such that it can be mapped onto a subset of the available resources of the system. The architecture needs to be capable of detecting a failure in a physical resource and then remapping the logical architecture back onto the restricted resources left functioning. Although the concept is more traditionally restricted to recovery from failed hardware units, it is also applicable to failed software units if there is sufficient 'run-time redundancy' to allow a software re-try or if there is sufficient redundant data to make the individual and isolated failure a little import.

Although traditionally applied to hardware, this technique is being developed for application to software and, thus, the total system. It must be considered at the first system design stage.

References:

Critical Issues in the Design of a Reconfigureable Control Computer. H. Schmid, J. Lam, R. Naro and K. Weir, FTCS 14 June 1984, IEEE 1984.

Assigning Processes to Processors: A Fault-tolerant Approach. June 1984, G. Kar and C.N. Nikolaou, Watson Research Centre, Yorktown

B.19 Tests de classes d'équivalence et de partition des données (référencé par td2 et td3)

Objectif

Tester le logiciel de façon adéquate en utilisant un minimum de cas de test. Les cas de test sont obtenus en sélectionnant les partitions du domaine d'entrée requis pour tester le logiciel.

Description

Cette stratégie d'essai repose sur la relation d'équivalence des entrées qui détermine une partition du domaine d'entrée.

Les scénarios de test sont sélectionnés dans le but de couvrir tous les sous-ensembles de cette partition. Un scénario de test minimum est pris dans chaque classe d'équivalence.

Les deux possibilités de base existant pour le partitionnement d'entrée sont les suivantes:

– les classes d'équivalence peuvent être définies au travers de la spécification. L'interprétation de la spécification peut être orientée entrée, par exemple les valeurs sélectionnées sont traitées de la même façon, ou orientée sortie, par exemple l'ensemble de valeurs conduisant au même résultat fonctionnel; et
– les classes d'équivalence sont éventuellement définies sur la structure interne du programme. Dans ce cas, les résultats des classes d'équivalence sont déterminés à partir de l'analyse statique du programme, par exemple l'ensemble de valeurs conduisant au même chemin en cours d'exécution.

Référence:

The Art of Software Testing. G. Myers, Wiley et Sons, New York, Etats-Unis, 1979.

B.20 Codes correcteurs et détecteurs d'erreurs (référencé par l'article 9)

Objectif

Détecter et corriger des erreurs dans les informations sensibles.

Description

Pour une information de n bits, un bloc codé de k bits est généré. Il permet la détection et la correction des erreurs. Les différents types de code incluent

– les codes de Hamming;
– les codes cycliques;
– les codes polynomiaux.

Références:

The Technology of Error Correcting Codes. E.R. Berlekamp, Proc. de l'IEEE, 68, 5, 1980.

A Short Course on Error Correcting Codes. N.J.A. Sloane, Springer-Verlag, Vienne, 1975.

B.21 Supposition d'erreurs (référencé par td2 et td8)

Objectif

Supprimer des erreurs de programmation courantes.

B.19 Equivalence Classes and Input Partition Testing (referenced by dt2 and dt3)

Aim

To test the software adequately using a minimum of test data. The test data is obtained by selecting the partitions of the input domain required to exercise the software.

Description

This testing strategy is based on the equivalence relation of the inputs, which determines a partition of the input domain.

Test cases are selected with the aim of covering all subsets of this partition. At least one test case is taken from each equivalence class.

There are two basic possibilities for input partitioning which are:

– equivalence classes may be defined on the specification. The interpretation of the specification may be either input oriented, for example the values selected are treated in the same way or output oriented, for example the set of values leading to the same functional result; and

– equivalence classes may be defined on the internal structure of the program. In this case the equivalence class results are determined from static analysis of the program, for example the set of values leading to the same path being executed.

Reference:

The Art of Software Testing. G. Myers, Wiley and Sons, New York, USA, 1979.

B.20 Error Detecting and Correcting Codes (referenced by clause 9)

Aim

To detect and correct errors in sensitive information.

Description

For an information of n bits, a coded block of k bits is generated which enables errors to be detected and corrected. Different types of code include

– hamming codes;

– cyclic codes;

– polynomial codes.

References:

The Technology of Error Correcting Codes. E.R. Berlekamp, Proc. of the IEEE, 68, 5, 1980.

A Short Course on Error Correcting Codes. N.J.A. Sloane, Springer-Verlag, Wien, 1975.

B.21 Error Guessing (referenced by dt2 and dt8)

Aim

To remove common programming errors.

Description

Il n'est pas exclu que l'expérience des tests et l'intuition combinées à la connaissance et à la curiosité envers le système soumis au test ajoutent certains scénarios de test inclassables dans l'ensemble des scénarios de test prévus. Il n'est pas exclu que des valeurs ou des combinaisons de valeurs spéciales soient sujettes à erreurs. Il est permis de dériver certains scénarios de test intéressants à partir des listes de contrôle d'inspection. Il n'est pas exclu d'examiner aussi si le système est suffisamment robuste. Peut-on pousser les boutons du panneau avant trop rapidement ou trop fréquemment ? Que se passe-t-il si l'on pousse deux boutons simultanément ?

B.22 Insertion d'erreurs (référencé par td2)

Objectif

Vérifier si un ensemble de scénarios de test est adéquat.

Description

Certains types d'erreurs connus sont insérés dans le programme et le programme est exécuté avec les scénarios de test dans les conditions de test. Si seules quelques-unes des erreurs insérées sont trouvées, l'ensemble des scénarios de test n'est pas adéquat. Le rapport entre les erreurs insérées trouvées et le nombre total d'erreurs insérées est une estimation du rapport entre les erreurs réelles trouvées et le nombre total d'erreurs. Cela donne une possibilité d'estimation des erreurs restantes et donc de l'effort de test qu'il reste à fournir.

$$\frac{\text{Erreurs insérées trouvées}}{\text{Nombre total d'erreurs trouvées}} = \frac{\text{Erreurs réelles trouvées}}{\text{Nombre total d'erreurs réelles}}$$

Il est admis que la détection de toutes les erreurs insérées indique que l'ensemble des scénarios de test est adéquat ou que les erreurs insérées étaient trop faciles à trouver. Cette méthode comporte des limitations car, pour obtenir un résultat utilisable, il faut que le type d'erreur ainsi que la position des insertions correspondent à la distribution statistique des erreurs réelles.

B.23 Analyse par arbre des événements (référencé par l'article 14)

Objectif

Modéliser, sous forme de schéma, la séquence des événements qui peuvent se développer dans un système après un événement déclencheur et, par ce moyen, indiquer comment des conséquences graves peuvent survenir.

Description

En haut du schéma sont écrites les conditions de la séquence ayant un rapport avec le déroulement suivant l'événement déclencheur qui constitue la cible de l'analyse. En commençant sous l'événement déclencheur, on tire un trait vers la première condition de la séquence. Là le schéma se divise en une branche «oui» et une branche «non», en décrivant la manière dont les développements futurs dépendent de la condition. Pour chacune de ces branches, on continue vers la prochaine condition de la même façon. Toutes les conditions, toutefois, ne sont pas pertinentes à toutes les branches. On continue jusqu'à la fin de la séquence et chaque branche de l'arbre structuré de cette façon représente une conséquence possible. L'arborescence des événements peut être utilisé pour calculer la probabilité des diverses conséquences reposant sur la probabilité et sur le nombre de conditions dans la séquence.

Description

Testing experience and intuition combined with knowledge and curiosity about the system under test may add some uncategorised test cases to the designed test case set. Special values or combinations of values may be error-prone. Some interesting test cases may be derived from inspection checklists. It may also be considered whether the system is robust enough. Can the buttons be pushed on the front-panel too fast or too often? What happens if two buttons are pushed simultaneously?

B.22 Error Seeding (referenced by dt2)

Aim

To ascertain whether a set of test cases is adequate.

Description

Some known error types are inserted in the program, and the program is executed with the test cases under test conditions. If only some of the seeded errors are found, the test case set is not adequate. The ratio of found seeded errors to the total number of seeded errors is an estimate of the ratio of found real errors to total number errors. This gives a possibility of estimating the number of remaining errors and thereby the remaining test effort.

$$\frac{\text{Seeded errors found}}{\text{Total number of seeded errors}} = \frac{\text{Real errors found}}{\text{Total number of real errors}}$$

The detection of all the seeded errors may indicate either that the test case set is adequate, or that the seeded errors were too easy to find. The limitations of the method are that, in order to obtain any usable results, the error types as well as the seeding positions must reflect the statistical distribution of real errors.

B.23 Event Tree Analysis (referenced by clause 14)

Aim

To model, in a diagrammatic form, the sequence of events that can develop in a system after an initiating event, and thereby indicate how serious consequences can occur.

Description

On the top of the diagram is written the sequence conditions that are relevant in the development following the initiating event which is the target of the analysis. Starting under the initiating event, one draws a line to the first condition in the sequence. There the diagram branches off into a 'yes' and a 'no' branch, describing how the future developments depend on the condition. For each of these branches, one continues to the next condition in a similar way. Not all conditions are, however, relevant for all branches. One continues to the end of the sequence, and each branch of the tree constructed in this way represents a possible consequence. The event tree can be used to compute the probability of the various consequences based on the probability and number of conditions in the sequence.

Référence:

Event Trees and their Treatment on PC Computers. N. Limnious et J.P. Jeannette, Reliability Engineering, Vol. 18, No. 3 1987.

B.24 Inspection de Fagan (référencé par td8)

Objectif

Mettre à jour des erreurs dans toutes les phases du développement du programme.

Description

L'audit «formel» des documents d'assurance qualité avait pour objectif de trouver des erreurs ou des omissions. Ce processus de contrôle se compose de cinq phases: la planification, la préparation, l'inspection, la correction et le suivi. Chacune de ces phases comporte son propre objectif distinct. Il faut que le développement complet du système (spécification, conception, codage et essais) soit contrôlé.

Référence:

Design and Code Inspections to Reduce Errors in Program Development. M.E. Fagan, IBM Systems Journal, No. 3, 1976.

B.25 Programmation par assertion (référencé par l'article 9)

Objectif

Détecter des défauts résiduels de conception du logiciel pendant l'exécution d'un programme, de manière à empêcher les défaillances critiques dans la sécurité du système et à continuer en direction d'une plus grande fiabilité.

Description

La méthode de programmation par assertion repose sur le concept de vérification d'une précondition (avant qu'une séquence d'instructions soit exécutée, la validité des conditions initiales est vérifiée) et d'une postcondition (les résultats sont vérifiés après l'exécution d'une séquence d'instructions). Si la précondition ou la postcondition n'est pas conforme, le traitement s'arrête avec une erreur.

Par exemple,

```
assertion <précondition>;
    action 1;
        :
        :
    action x;
assertion <postcondition>;
```

Références:

A Discipline of Programming. E.W. Dijkstra, Prentice-Hall, 1976.

The Science of Programming. D. Gries, Springer-Verlag, 1981.

Software Development – A Rigorous Approach. C.B. Jones, Prentice-Hall, 1980.

Reference:

Event Trees and their Treatment on PC Computers. N. Limnious and J.P. Jeannette, Reliability Engineering, Vol. 18, No. 3 1987.

B.24 Fagan Inspections (referenced by dt8)

Aim

To reveal errors in all phases of the program development.

Description

A 'formal' audit on quality assurance documents aimed at finding errors and omissions. The inspection process consists of five phases: planning, preparation, inspection, rework and follow-up. Each of these phases has its own separate objective. The complete system development (specification, design, coding and testing) must be inspected.

Reference:

Design and Code Inspections to Reduce Errors in Program Development. M.E. Fagan, IBM Systems Journal, No. 3, 1976.

B.25 Failure Assertion Programming (referenced by clause 9)

Aim

To detect residual software design faults during execution of a program, in order to prevent safety critical failures of the system and to continue operation for high reliability.

Description

The assertion programming method follows the idea of checking a pre-condition(before a sequence of statements is executed, the initial conditions are checked for validity) and a post-condition (results are checked after the execution of a sequence of statements). If either the pre-condition or the post-condition is not fulfilled, the processing stops with an error.

For example,

```
assert <pre-condition>;
    action 1;
        :
        :
    action x;
assert <post-condition>;
```

References:

A Discipline of Programming. E.W. Dijkstra, Prentice-Hall, 1976.

The Science of Programming. D. Gries, Springer-Verlag, 1981.

Software Development – A Rigorous Approach. C.B. Jones, Prentice-Hall, 1980.

B.26 SEEA – Analyse des effets des erreurs du logiciel (référencé par les articles 9, 11 et 14)

Objectif

Identifier les modules logiciels et leur criticité; proposer des moyens permettant de détecter des erreurs logicielles et d'améliorer la robustesse du logiciel; évaluer la somme de validation requise sur les divers composants logiciels.

Description

L'analyse se fait en trois phases:

– identification des modules logiciels critiques;
– détermination de la profondeur de l'analyse (au niveau d'une simple ligne d'instruction, d'un groupe d'instructions, d'un module, etc.) requise pour chaque module logiciel, à partir de sa spécification;
– analyse des erreurs du logiciel.

Le résultat de cette phase est présenté dans un tableau répertoriant les informations suivantes:

– nom du module;
– erreur examinée;
– conséquences de l'erreur au niveau du module;
– conséquences au niveau du système;
– critère de sécurité violé;
– gravité de l'erreur;
– moyen proposé de détection d'erreur;
– critère violé si le moyen de détection proposé est mis en application;
– gravité résiduelle si le moyen de détection est mis en application;
– synthèse.

La synthèse identifie les scénarios non sûrs restants et l'effort de validation nécessaire étant donné l'importance de chaque module.

L'analyse SEEA, en tant qu'analyse en profondeur effectuée par une équipe indépendante, constitue une puissance méthode de mise à jour des bugs.

Références:

Railway fixed equipment and rolling stock. Data processing. Software dependability – Adapted methods for software safety analysis, French standard NF F 71-013, décembre 1990.

IRSE International Technical Committee Report No. 1

«SAFETY SYSTEM VALIDATION with regard to cross acceptance of signalling systems by the railways»

P. THIREAU
«Méthodologie d'Analyse des Effets des Erreurs Logiciel (AEEL) appliquée à l'étude d'un logiciel de haute sécurité»
5ème Conférence internationale sur la fiabilité et la maintenabilité.
Biarritz – France – 1986

B.26 SEEA – Software Error Effect Analysis (referenced by clauses 9, 11 and 14)

Aim

To identify software modules, their criticality; to propose means for detecting software errors and enhancing software robustness; to evaluate the amount of validation needed on the various software components.

Description

The analysis is done in three phases:

- vital software modules identification;
- determination of the depth of the analysis (at the level of a single instruction line, a group of instructions, a module, etc.) needed for each software module, from its specification.
- software error analysis.

The result of this phase is a table listing the following information:

- module name;
- error considered;
- consequences of the error at the module level;
- consequences at the system level;
- violated safety criterion;
- error criticality;
- proposed error detection means;
- violated criterion if the detection means is implemented;
- residual criticality if the detection means is implemented.
- synthesis.

The synthesis identifies the remaining unsafe scenarios and the validation effort needed given the criticality of each module.

SEEA, being an in-depth analysis carried out by an independent team, is a powerful bug-finding method.

References:

Railway fixed equipment and rolling stock. Data processing. Software dependability – Adapted methods for software safety analysis, French standard NF F 71-013, December 1990.

IRSE International Technical Committee Report No. 1

"SAFETY SYSTEM VALIDATION with regard to cross acceptance of signalling systems by the railways"

P. THIREAU
"Méthodologie d'Analyse des Effets des Erreurs Logiciel (AEEL) appliquée à l'étude d'un logiciel de haute sécurité"
5th International Conference on Reliability and Maintainability.
Biarritz – France – 1986

D.J. REIFER
«Software failures modes and effects analysis»
Transaction on reliability IEE – Vol. R28 No. 3
août 79 – Pages 247/249

J.R. FRAGOLA et J.F. SPAMM
«The software error effects analysis, a qualitative design tool»
IEEE Symposium Computer on Software Reliability
1973 – Pages 90/93

B.27 Détection des défauts et diagnostic (référencé par l'article 9)

Objectif

Détecter dans un système les défauts qui peuvent conduire à une défaillance, et fournir ainsi la base de contre-mesures de manière à minimiser les conséquences des défaillances.

Description

La détection des défauts est le processus consistant à rechercher des états erronés dans un système (qui sont provoqués, comme il est expliqué plus haut, par un défaut à l'intérieur du (sous-) système à vérifier). L'objectif principal de la détection des défauts consiste à inhiber l'effet des résultats faux. Un système qui fournit des résultats corrects ou pas de résultats du tout, est appelé «autocorrecteur».

La détection de défauts repose sur les principes de la redondance (principalement pour détecter les défauts matériels) et de la diversité (défauts logiciels). Un certain type de vote est nécessaire pour déterminer le bien fondé des résultats. Les méthodes spéciales applicables sont la programmation par assertion, la programmation en N versions et la technique sécurité contrôlée, et au niveau matériel on peut introduire des capteurs, des boucles de contrôle, des codes de contrôle d'erreur, etc.

Il est admis que la détection de défauts soit réalisée au moyen de contrôles du domaine des valeurs ou du domaine du temps sur différents niveaux, spécialement au niveau physique (température, tension etc.), logique (codes de détection d'erreur), fonctionnel (assertions) ou externe (contrôles de plausibilité). Il est admis que les résultats de ces contrôles soient stockés et associés aux données affectées pour permettre le suivi des défaillances.

Les systèmes complexes sont composés de sous-systèmes. L'efficacité de la détection des défauts, du diagnostic et de la compensation des défauts dépend de la complexité des interactions entre les sous-systèmes qui influencent la propagation des défauts.

Le diagnostic des défauts isole le plus petit sous-système pouvant être identifié. Des sous-systèmes plus petits autorisent un diagnostic plus détaillé des défauts (identification des états erronés).

B.28 Analyse par arbre des causes (référencé par les articles 9 et 14)

Objectif

Apporter une aide dans l'analyse des événements, ou des combinaisons d'événements, qui conduiront à une conséquence risquée ou grave.

D.J. REIFER
"Software failures modes and effects analysis"
Transaction on reliability IEE – Vol. R28 No. 3
August 79 – Pages 247/249

J.R. FRAGOLA and J.F. SPAMM
"The software error effects analysis, a qualitative design tool"
IEEE Symposium Computer on Software Reliability
1973 – Pages 90/93

B.27 Fault Detection and Diagnosis (referenced by clause 9)

Aim

To detect faults in a system, which might lead to a failure, thus providing the basis for counter-measures in order to minimise the consequences of failures.

Description

Fault detection is the process of checking a system for erroneous states (which are caused, as explained before, by a fault within the (sub)system to be checked). The primary goal of fault detection is to inhibit the effect of wrong results. A system which delivers either correct results, or no results at all, is called "self-checking".

Fault detection is based on the principles of redundancy (mainly to detect hardware faults) and diversity (software faults). Some sort of voting is needed to decide on the correctness of results. Special methods applicable are assertion programming, N-version programming and the safety-bag technique and on hardware level by introducing sensors, control loops, error checking codes, etc.

Fault detection may be achieved by checks in the value domain or in the time domain on different levels, especially on the physical (temperature, voltage etc.), logical (error-detecting codes), functional (assertions) or external level (plausibility checks). The results of these checks may be stored and associated with the data affected to allow failure tracking.

Complex systems are composed of subsystems. The efficiency of fault detection, diagnosis and fault compensation depends on the complexity of the interactions among the subsystems, which influences the propagation of faults.

Fault diagnosis isolates the smallest subsystem that may be identified. Smaller subsystems allow a more detailed diagnosis of faults (identification of erroneous states).

B.28 Fault Tree Analysis (referenced by clauses 9 and 14)

Aim

To aid in the analysis of events, or combinations of events, that will lead to a hazard or serious consequence.

Description

En commençant par un événement qui serait la cause immédiate d'un danger ou d'une conséquence grave («l'événement du haut»), l'analyse est conduite le long d'un chemin arborescent. Les combinaisons des causes sont décrites avec les opérateurs logiques (et, ou, etc.). Les causes intermédiaires sont analysées de la même façon et ainsi de suite jusqu'aux événements de base où s'arrête l'analyse.

La méthode est graphique et un ensemble de symboles standardisés est utilisé pour dessiner l'arbre des causes. Cette méthode est surtout adaptée à l'analyse des systèmes matériels, mais il y a eu aussi des tentatives pour appliquer cette approche à l'analyse des défaillances logicielles.

Références:

System Reliability Engineering Methodology: A Discussion of the State of the Art. J.B. Fusseland, J.S. Arend, Nuclear Safety 20(5), 1979.

Fault Tree Handbook. W.E. Vesely *et al.*, NUREG-0942, Division of System Safety Office of Nuclear Reactor Regulation, U.S. Nuclear Regulatory Commission Washington, D.C. 20555, 1981.

Reliability Technology. A.E. Greene et A.J. Bourne, Wiley-Interscience, 1972.

B.29 Automates à états finis/Schémas de transition d'état (référencé par td5 et td7)

Objectif

Définir ou mettre en oeuvre la structure de contrôle d'un système.

Description

De nombreux systèmes peuvent être définis par rapport à leurs états, leurs entrées et leurs actions. Ainsi, quand le système est dans l'état S1, la réception de l'entrée I provoquerait éventuellement l'exécution de l'action A et le passage à l'état S2. En définissant les actions d'un système pour chaque entrée dans chaque état, nous pouvons définir complètement un système. Le modèle résultant du système porte le nom d'automate à état fini. Il est souvent représenté par un schéma appelé schéma de transition d'état montrant comment le système passe d'un état à un autre, ou par une matrice dont les dimensions sont l'état et l'entrée et dont les cellules contiennent l'action et le nouvel état résultant de la réception de l'entrée dans l'état donné.

Quand un système est compliqué ou s'il a une structure naturelle, cette situation peut être reproduite dans plusieurs niveaux d'automates à états finis.

Il est possible de vérifier l'état complet (le système doit avoir une action et un nouvel état pour chaque entrée dans chaque état), la cohérence (un seul état est défini pour chaque paire état/entrée) et l'accessibilité (s'il est possible ou non de passer d'un état à l'autre par une quelconque séquence d'entrées) d'une spécification ou une conception exprimée sous forme d'automate à état fini. Ces propriétés sont importantes pour les systèmes critiques et elles peuvent être vérifiées. Les outils permettant de supporter ces contrôles sont faciles à écrire. Il existe aussi des algorithmes autorisant la génération automatique des scénarios de test afin de vérifier l'implémentation de l'automate à état fini ou l'animation d'un modèle d'automate à état fini.

Description

Starting at an event which would be the immediate causes of a hazard or serious consequence (the 'top event') analysis is carried out along a tree path. Combinations of causes are described with logical operators (and, or, etc.). Intermediate causes are analysed in the same way, and so on back to basic events where analysis stops.

The method is graphical, and a set of standardised symbols are used to draw the fault tree. It is mainly intended for the analysis of hardware systems, but there have also been attempts to apply this approach to software failure analysis.

References:

System Reliability Engineering Methodology: A Discussion of the State of the Art. J.B. Fusseland, J.S. Arend, Nuclear Safety 20(5), 1979.

Fault Tree Handbook. W.E. Vesely *et al.*, NUREG-0942, Division of System Safety Office of Nuclear Reactor Regulation, U.S. Nuclear Regulatory Commission Washington, D.C. 20555, 1981.

Reliability Technology. A.E. Greene and A.J. Bourne, Wiley-Interscience, 1972.

B.29 Finite State Machines/State Transition Diagrams (referenced by dt5 and dt7)

Aim

To define or implement the control structure of a system.

Description

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

Where a system is complicated or has a natural structure this can be reflected in a layered FSM.

A specification or design expressed as an FSM can be checked for completeness (the system must have an action and new state for every input in every state), for consistency (only one state change is defined for each state/input pair) and reachability (whether or not it is possible to get from one state to another by any sequence of inputs). These are important properties for critical systems and they can be checked. Tools to support these checks are easily written. Algorithms also exist that allow the automatic generation of test cases for verifying an FSM implementation or for animating an FSM model.

Référence:

Introduction to the theory of Finite State Machines. A. Gill, McGraw-Hill, 1962.

B.30 Méthodes formelles (référencé par les articles 8 et 10 et td5)

Objectif

Développer les logiciels d'une façon reposant sur les mathématiques. Ceci inclut les techniques de conception formelle et de codage formel.

Description

Les méthodes formelles proposent un moyen de développer une description d'un système à une certaine étape de sa spécification de développement, de sa conception ou de son code. La description qui en découle revêt une forme mathématique et peut être soumise à une analyse mathématique en vue de détecter diverses catégories d'incohérences ou d'erreurs. En outre, la description peut, dans certains cas, être analysée par la machine avec une rigueur similaire à la vérification de la syntaxe d'un programme par un compilateur ou animée de manière à afficher différents aspects du comportement du système décrit. L'animation peut renforcer la confiance envers le système et assurer qu'il répond à la prescription réelle ainsi qu'à la prescription spécifiée de façon formelle.

Une méthode formelle proposera en général une notation (en principe une forme quelconque de mathématique discrète est utilisée), une technique permettant de dériver une description dans cette notation ainsi que diverses formes d'analyse permettant de vérifier une description au regard de différentes propriétés de correction.

Plusieurs exemples de méthodes formelles sont décrites dans les paragraphes suivants de cette bibliographie. Les méthodes formelles décrites sont les suivantes: CCS, CSP, HOL, LOTOS, OBJ, Logique temporelle, VDM et Z.

B.30.1 CCS – Algèbre des Systèmes de Transmission

Objectif

La méthode CCS est un moyen de décrire et de raisonner sur le comportement des systèmes de processus parallèles et communicants.

Description

Similaire à la méthode CSP, la méthode CCS est une algèbre appliquée au comportement de systèmes. La conception du système est modélisée comme un réseau de processus indépendants fonctionnant en série ou en parallèle. Les processus peuvent communiquer via des ports (similaires au canaux CSP), la communication ne se produit que lorsque les deux processus sont prêts. Le non-déterminisme peut être modélisé. En commençant par une représentation très abstraite de l'ensemble du système (connue sous le nom de trace), il est possible de réaliser un affinement par étapes du système pour arriver à composer des processus de communication dont le comportement global est celui requis par l'ensemble du système. De la même façon, il est possible d'adopter une méthode ascendante, combinant des processus et déduisant les propriétés du système résultant à l'aide de règles d'inférence liées aux règles de composition.

Reference:

Introduction to the theory of Finite State Machines. A. Gill, McGraw-Hill, 1962.

B.30 Formal Methods (referenced by clauses 8 and 10 and dt5)

Aim

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

Description

Formal methods provide a means of developing a description of a system at some stage in its development specification, design or code. The resulting description takes a mathematical form and can be subjected to mathematical analysis to detect various classes of inconsistency or incorrectness. Moreover, the description can in some cases be analysed by machine with a rigour similar to the syntax checking of a source program by a compiler, or animated to display various aspects of the behaviour of the system described. Animation can give extra confidence that the system meets the real requirement as well as the formally specified requirement.

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

Several examples of Formal Methods are described in the following subsections of this bibliography. The Formal Methods described are CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM and Z.

B.30.1 CS – Calculus of Communicating Systems

Aim

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

Description

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

Références:

A Calculus of Communicating Systems. R. Milner, Report number ECS-LFCS-86-7, Laboratory for Foundations of Computer Science, Département informatique, Université d'Edimbourg, Royaume-Uni.

The Specification of Complex Systems. B. Cohen, W.T. Harwood, M.I. Jackson. Addison-Wesley, 1986.

B.30.2 CSP – Processus Séquentiels de Communication

Objectif

La méthode CSP est une technique appliquée à la spécification des systèmes logiciels concurrents, autrement dit des systèmes de processus de communication fonctionnant simultanément.

Description

La méthode CSP propose un langage pour la spécification des systèmes de processus et une preuve permettant de vérifier que l'implémentation des processus satisfait à leurs spécifications (décrites comme une trace – séquences permises d'événements).

Un système est modélisé comme un réseau de processus indépendants. Chaque processus est décrit au travers de tous ses comportements possibles. Un système est modélisé en composant des processus en série ou en parallèle. Les processus peuvent communiquer (synchroniser ou échanger des données) via des canaux, la communication n'ayant lieu que lorsque les deux processus sont prêts. La synchronisation relative des événements peut être modélisée.

La théorie sous-tendant les processus CSP a été directement incorporée dans l'architecture du transputeur INMOS, et le langage OCCAM autorise l'implémentation directe d'un système spécifié CSP sur un réseau de transputeurs.

Référence:

Communicating Sequential Processes. CAR. Hoare, Prentice-Hall, 1985.

B.30.3 HOL – Logique d'Ordre Supérieur

Objectif

Il s'agit d'un langage logique devant servir de base à la spécification et à la vérification du matériel.

Description

HOL (Higher Order Logic) fait référence à une notation logique spécifique et à son système de support machine, les deux ayant été développés dans les laboratoires informatiques de l'Université de Cambridge. La notation logique s'inspire principalement de la Simple théorie des types de Church (Simple Theory of Types) et le système du support machine repose sur le système des fonctions LCF (Logique des fonctions calculables – Logic of Computable Functions).

Référence:

HOL: A Machine Orientated Formulation of Higher Order Logic. M. Gordon, University of Cambridge Technical Report, Numéro 68.

References:

A Calculus of Communicating Systems. R. Milner, Report number ECS-LFCS-86-7, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK.

The Specification of Complex Systems. B. Cohen, W.T. Harwood, M.I. Jackson. Addison-Wesley, 1986.

B.30.2 CSP – Communicating Sequential Processes

Aim

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

Description

CSP provides a language for the specification of systems of processes and proof for verifying that the implementation of processes satisfies their specifications (described as a trace – permissible sequences of events).

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

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

Reference:

Communicating Sequential Processes. CAR. Hoare, Prentice-Hall, 1985

B.30.3 HOL – Higher Order Logic

Aim

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

Description

HOL (Higher Order Logic) refers to a particular logic notation and its machine support system, both of which were developed at the University of Cambridge Computer Laboratory. The logic notation is mostly taken from Church's Simple Theory of Types and the machine support system is based upon the LCF (Logic of Computable Functions) system.

Reference:

HOL: A Machine Orientated Formulation of Higher Order Logic. M. Gordon, University of Cambridge Technical Report, Number 68.

B.30.4 LOTOS

Objectif

Le langage LOTOS est un moyen de décrire et de raisonner sur le comportement des systèmes de processus parallèles et communicants.

Description

Le langage LOTOS (Language for Temporal Ordering Specification) repose sur CCS en intégrant des caractéristiques supplémentaires issues des algèbres connexes CSP et CIRCAL (Circuit Calculus). Il surmonte la faiblesse de CCS en matière de manipulation des structures des données et des expressions des valeurs en la combinant avec un deuxième composant basé sur le langage des types de données abstraits ACT ONE. Le composant LOTOS de définition du processus pourrait, toutefois, être utilisé avec d'autres formalismes pour la description des types de données abstraits.

Référence:

Information Processing Systems – Open Systems Inter-connection – LOTOS – A Formal Description Technique based on the Temporal Ordering of Observational Behaviour. ISO/DIS 8807, 20 juillet 1987.

B.30.5 OBJ

Objectif

Fournir une spécification précise du système avec une remontée de l'information utilisateur et une validation du système préalable à l'implémentation.

Description

OBJ est un langage de spécification algébrique. Les utilisateurs spécifient des exigences en terme d'équation algébrique. Les aspects de comportement, ou constructifs, du système sont spécifiés en termes d'opérations agissant sur les types de données abstraits (ADT). Un type de données abstrait est semblable à un ensemble ADA dans lequel le comportement de l'opérateur est visible tandis que les détails d'implémentation sont «cachés».

Une spécification OBJ et l'implémentation par étapes consécutives relèvent des mêmes techniques de preuves formelles que les autres approches formelles. En outre, puisque les aspects constructifs de la spécification OBJ sont exécutables par la machine, il est simple d'accomplir la validation du système à partir de la spécification elle-même. L'exécution est essentiellement l'évaluation d'une fonction par une substitution d'équation (réécriture) qui continue jusqu'à ce qu'une valeur de sortie spécifique soit obtenue. Cette exécutabilité permet aux utilisateurs finals du système envisagé d'avoir un «aperçu» du système final au stade de la spécification du système sans avoir la nécessité de connaître les techniques de spécification formelles sous-jacentes.

Comme pour toutes les autres techniques ADT, le langage OBJ n'est applicable qu'aux systèmes séquentiels ou aux aspects séquentiels des systèmes concurrents. Ce langage a été largement utilisé pour la spécification des applications industrielles de petite et de grande échelle.

Références:

An introduction to OBJ; A language for Writing and Testing Specifications. J. A. Goguen et J. Tardo, Specification of Reliable Software, IEEE Press 1979, republié dans Software Specification Techniques, N. Gehani, A. McGettrick (eds), Addison-Wesley, 1985.

Algebraic Specification for Practical Software Production. C. Rattray, Cogan Press, 1987.

B.30.4 LOTOS

Aim

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

Description

LOTOS (Language for Temporal Ordering Specification) is based on CCS with additional features from the related algebras CSP and CIRCAL (Circuit Calculus). It overcomes the weakness of CCS in the handling of data structures and value expressions by combining it with a second component based on the abstract data type language ACT ONE. The process definition component of LOTOS could, however, be used with other formalisms for the description of abstract data types.

Reference:

Information Processing Systems – Open Systems Inter-connection – LOTOS – A Formal Description Technique based on the Temporal Ordering of Observational Behaviour. ISO/DIS 8807, July 20, 1987.

B.30.5 OBJ

Aim

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

Description

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

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

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

References:

An introduction to OBJ; A language for Writing and Testing Specifications. J.A. Goguen and J. Tardo, Specification of Reliable Software, IEEE Press 1979, reprinted in Software Specification Techniques, N. Gehani, A. McGettrick (eds), Addison-Wesley, 1985.

Algebraic Specification for Practical Software Production. C. Rattray, Cogan Press, 1987.

An Algebraic Approach to the Standardisation and Certification of Graphics Software. R. Gnatz, Computer Graphics Forum 2(2/3), 1983.

DTI STARTS Guide. 1987, NCC, Oxford Road, Manchester, Royaume-Uni.

B.30.6 Logique temporelle

Objectif

Expression directe des prescriptions en matière de sécurité et de fonctionnement et démonstration formelle que ces propriétés sont préservées dans les étapes de développement suivantes.

Description

La logique standard des prédicats (du premier ordre) ne contient aucun concept de temps. La logique temporelle étend la logique du premier ordre en ajoutant des opérateurs modaux (par exemple, «Dorénavant» et «Finalement»). Ces opérateurs peuvent être utilisés pour qualifier des assertions sur le système. Par exemple, il se peut que des propriétés de sécurité soient tenues de contenir «dorénavant», tandis qu'il sera peut-être demandé à d'autres états du système d'être atteints «finalement» à partir de quelque autre état d'initialisation. Les formules temporelles sont interprétées sur des séquences d'états (comportements). Ce qui constitue un «état» dépend du niveau choisi de description. Cela peut faire référence à l'ensemble du système, à un composant du système ou à un programme informatique. Les intervalles temporels quantifiés et les contraintes ne sont pas explicitement gérés dans la logique temporelle. La synchronisation absolue doit être gérée par la création d'états temporels supplémentaires comme partie intégrante de la définition de l'état.

Références:

Temporal Logic of Programs. F. Kroger EATCS Monographs on Computer Science, Vol. 8, Springer Verlag, 1987.

Design for Safety using Temporal Logic. J. Gorski. SAFECOMP 86, Sarlat France, Pergamon Press, octobre 1986.

Logics for Computer Programming. D. Gabay. Ellis Horwood.

B.30.7 VDM – Méthode de Développement de Vienne

Objectif

Spécification et mise en oeuvre des programmes séquentiels.

Description

La méthode VDM est une technique de spécification à base mathématique et une technique de raffinement des implémentations d'une manière autorisant la preuve de leur correction par rapport à la spécification.

La technique de spécification est basée sur un modèle en ce sens que l'état du système est modélisé en termes de structures de la théorie des ensembles sur lesquelles sont définis des invariants (prédicats), et les opérations sur cet état sont modélisées en spécifiant leurs préconditions et postconditions en terme d'état du système. Il peut être montré que les opérations préservent les invariants du système.

L'implémentation de la spécification est réalisée par la réification de l'état du système en termes de structures de données dans le langage cible et par le raffinement des opérations en terme d'un programme dans le langage cible. Les étapes de réification et de raffinement donnent naissance à des obligations de preuve qui établissent leur validité. Que ces obligations soient ou non menées à bien est un choix réservé au concepteur.

An Algebraic Approach to the Standardisation and Certification of Graphics Software. R. Gnatz, Computer Graphics Forum 2(2/3), 1983.

DTI STARTS Guide. 1987, NCC, Oxford Road, Manchester, UK.

B.30.6 Temporal Logic

Aim

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

Description

Standard First Order Predicate Logic contains no concept of time. Temporal logic extends First Order logic by adding modal operators (e.g. 'Henceforth' and 'Eventually'). These operators can be used to qualify assertions about the system. For example, safety properties might be required to hold 'henceforth', whilst other desired system states might be required to be attained 'eventually' from some other initiating state. Temporal formulas are interpreted on sequences of states (behaviours). What constitutes a 'state' depends on the chosen level of description. It can refer to the whole system, a system component or the computer program. Quantified time intervals and constraints are not handled explicitly in temporal logic. Absolute timing has to be handled by creating additional time states as part of the state definition.

References:

Temporal Logic of Programs. F. Kroger EATCS Monographs on Computer Science, Vol. 8, Springer Verlag, 1987.

Design for Safety using Temporal Logic. J. Gorski. SAFECOMP 86, Sarlat France, Pergamon Press, October 1986.

Logics for Computer Programming. D. Gabay. Ellis Horwood.

B.30.7 VDM – Vienna Development Method

Aim

The systematic specification and implementation of sequential programs.

Description

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

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

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

La méthode VDM est principalement utilisée au stade de la spécification mais peut être employée dans les stades de conception et d'implémentation conduisant au code source. Elle peut aussi être appliquée aux programmes séquentiels ou aux processus séquentiels dans les systèmes concurrents.

Références:

Software Development – A Rigorous Approach. C.B. Jones. Prentice-Hall, 1980.

Formal Specification and Software Development. D. Bjorner et C.B. Jones. Prentice-Hall, 1982.

Systematic Software Development using VDM. C.B. Jones. Prentice-Hall, 1986.

The Specification of Complex Systems. B. Cohen, W.T. Harwood et M.I. Jackson. Addison-Wesley, 1986.

B.30.8 Z et B

Objectif

Z est une notation de langage de spécification destinée aux systèmes séquentiels et une technique de conception permettant au développeur d'évoluer d'une spécification Z vers des algorithmes exécutables d'une manière permettant d'établir la preuve de leur validité par rapport à la spécification.

La notation Z est surtout utilisée au stade de la spécification, mais une méthode a été mise au point pour passer de la spécification à la conception et à l'implémentation. Cette notation est parfaitement adaptée au développement des systèmes séquentiels orientés de données.

B est une méthode associée.

Description

A l'instar de la méthode VDM, la technique de spécification est basée sur un modèle en ce sens que l'état du système est modélisé en termes de structures de la théorie des ensembles sur lesquelles sont définis des invariants (prédicats), et les opérations sur cet état sont modélisées en spécifiant leurs préconditions et postconditions en terme d'état du système. Il peut être montré que les opérations préservent les invariants du système, démontrant par là leur cohérence. La partie formelle de la spécification est divisée en schémas permettant la structuration des spécifications par le biais du raffinement.

En principe, une spécification Z est un mélange de Z formel et de texte explicatif informel en langage naturel. (Le texte formel par lui-même est parfois trop laconique pour permettre une lecture facile et il est souvent nécessaire d'expliquer ses objectifs, en revanche le langage naturel informel peut aisément devenir vague et imprécis.)

A la différence de la méthode VDM, Z est une notation plutôt qu'une méthode complète. Toutefois, une méthode associée (appelée B) a été développée qui peut être utilisée conjointement à Z. La méthode B repose sur le principe du raffinement par étapes.

Références:

The Z Notation – A Reference Manual. J.M. Spivey, Prentice-Hall, 1988.

Specification Case Studies. Edited by I. Hayes, Prentice-Hall, 1987.

Specification of the UNIX Filestore. C. Morgan et B. Sufrin. IEEE Transactions on Software Engineering, SE-10, 2 mars 1984.

The B Book – Assigning programs to meanings. J.R. Abrial, Cambridge United Press, 1996.

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

References:

Software Development – A Rigorous Approach. C.B. Jones. Prentice-Hall, 1980.

Formal Specification and Software Development. D. Bjorner and C.B. Jones. Prentice-Hall, 1982.

Systematic Software Development using VDM. C.B. Jones. Prentice-Hall, 1986.

The Specification of Complex Systems. B. Cohen, W.T. Harwood and M.I. Jackson. Addison-Wesley, 1986.

B.30.8 Z and B

Aim

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

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

B is an associated method.

Description

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

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

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

References:

The Z Notation – A Reference Manual. J.M. Spivey, Prentice Hall, 1988.

Specification Case Studies. Edited by I. Hayes, Prentice-Hall, 1987.

Specification of the UNIX Filestore. C. Morgan and B. Sufrin. IEEE Transactions on Software Engineering, SE-10, 2 March 1984.

The B Book – Assigning Programs to Meanings. J.R. Abrial, Cambridge United Press, 1996.

B.31 Preuve formelle (référencé par l'article 11)

Objectif

En utilisant des modèles et des règles théoriques et mathématiques, il est possible de prouver la validité d'un programme sans l'exécuter.

Description

Un certain nombre d'assertions sont déclarées à divers endroits du programme et elles sont utilisées comme préconditions et postconditions vers les différents chemins du programme. La preuve consiste à montrer que le programme transfère les préconditions dans les postconditions selon un ensemble de règles logiques, et que le programme prend fin.

Plusieurs méthodes formelles sont décrites dans la présente bibliographie, par exemple, CCS, CSP, HOL, LOTOS, OBJ, Logique temporelle, VDM et Z. Leurs descriptions figurent dans l'article B.30.

Référence:

Can Program Proving be made Practical. J. Dahl, Research Report, ISBN 82-90230-26-5 No. 33, Oslo, mai.

B.32 Rattrapage par progression (référencé par l'article 9)

Objectif

Permettre une exploitation fonctionnelle correcte en présence d'un ou de plusieurs défauts.

Description

Si un défaut a été détecté, l'état actuel du système est manipulé pour obtenir un état qui deviendra cohérent un peu plus tard. Ce concept est spécialement adapté aux systèmes en temps réel, dotés d'une petite base de données et d'une grande vitesse de changement de l'état interne. On suppose que, au minimum, une partie de l'état du système s'impose éventuellement sur l'environnement et que seule une partie des états du système est influencée (contrainte) par l'environnement.

B.33 Dégradation contrôlée

Objectif

Maintenir la disponibilité des fonctions cruciales du système malgré les défaillances en éliminant les fonctions moins cruciales.

Description

Cette technique attribue des priorités aux diverses fonctions que le système doit exécuter. La conception assure, s'il arrivait que les ressources soient insuffisantes pour exécuter toutes les fonctions du système, que les fonctions de plus haute priorité seraient alors exécutées de préférence aux fonctions moins prioritaires. Par exemple, il est admis que des fonctions de consignation des erreurs et des événements soient moins prioritaires que les fonctions de contrôle du système. Le contrôle du système pourrait continuer si le matériel associé à la consignation d'erreurs devait échouer. En outre, s'il arrivait que le matériel de contrôle du système échoue, mais pas le matériel de consignation d'erreurs, le matériel de consignation d'erreur prendrait en charge la fonction de contrôle. Cette technique est surtout appliquée au matériel mais elle est applicable à l'ensemble du système. Elle doit être prise en compte depuis la toute première phase de conception.

B.31 Normal Proof (referenced by clause 11)

Aim

Using theoretical and mathematical models and rules it is possible to prove the correctness of a program without executing it.

Description

A number of assertions are stated at various locations in the program, and they are used as pre- and post-conditions to various paths in the program. The proof consists of showing that the program transfers the pre-conditions into the post-conditions according to a set of logical rules, and that the program terminates.

Several Formal Methods are described in this bibliography, for instance, CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM and Z. The descriptions of these can be found in clause B.30.

Reference:

Can Program Proving be made Practical. J. Dahl, Research Report, ISBN 82-90230-26-5 No. 33, Oslo, May.

B.32 Forward Recovery (referenced by clause 9)

Aim

To provide correct functional operation in the presence of one or more faults.

Description

If a fault has been detected, the current state of the system is manipulated to obtain a state, which will be consistent some time later. This concept is especially suited for real-time systems with a small database and fast rate of change of the internal state. It is assumed that at least part of the system state may be imposed onto the environment, and only part of the system states are influenced (forced) by the environment.

B.33 Graceful Degradation

Aim

To maintain the more critical system functions available despite failures by dropping the less critical functions.

Description

This technique gives priorities to the various functions to be carried out by the system. The design then ensures that should there be insufficient resources to carry out all the system functions, then the higher priority functions are carried out in preference to the lower ones. For example, error and event logging functions may be lower priority than system control functions. System control would continue if the hardware associated with error logging were to fail. Further, should the system control hardware fail, but not the error logging hardware, then the error logging hardware would take over the control function. This is predominantly applied to hardware but is applicable to the total system. It must betaken into account from the top-most design phase.

Références:

Space Shuttle Software. C.T. Sheridan, Datamation, Vol. 24, juillet 1978

The Evolution of Fault-Tolerant Computing. Vol. 1 of Dependable Computing and Fault Tolerant Systems, Edited by A. Avizienis, H. Kopetz et J.C. Laprie, Springer-Verlag, ISBN 3-211-81941-X, 1987.

Fault Tolerance, Principle and Practices. Vol. 3 of [2] above, T. Anderson et P.A. Lee, Springer-Verlag, ISBN 3-211-82077-9, 1987.

B.34 Etude de risque et d'opérabilité HAZOP (Hazard and Operability Study)

Objectif

Etablir, au moyen d'une série d'examens systématiques des parties des composants du système informatique et de leur fonctionnement, des modes de défaillance qui amènent à des situations potentiellement dangereuses dans le système contrôlé.

En principe, les événements dangereux sur les systèmes contrôlés sont l'incendie, l'explosion, la diffusion de matériaux toxiques (chimiques ou nucléaires) ou une perte économique grave.

On suppose que les événements dangereux survenant dans le système sous contrôle ont été identifiés dans une analyse des risques distincte et la conséquence des événements dangereux classifiée par degrés de gravité.

L'analyse qui englobe toutes les étapes du cycle de vie du projet, de la spécification à la maintenance et la modification en passant par la conception, a pour objectif d'identifier à chaque étape les événements/modes de défaillance qui pourraient conduire à des dangers potentiels et donc de les éliminer.

Description

L'analyse est réalisée par une équipe d'ingénieurs (recouvrant les disciplines informatique, instrumentale, électrique, de processus, de sécurité et de fonctionnement), dirigés par un spécialiste formé aux techniques d'analyse des risques, au cours d'une série de réunions planifiées.

Il est important qu'une planification soit affectée dans le cadre du projet à ces réunions: chacune d'elle doit être prévue pour une durée minimum d'une demi-journée et par souci d'efficacité, leur nombre ne doit pas excéder quatre par semaine afin de permettre la mise à jour du flux de la documentation d'accompagnement.

Avant l'étude, des listes de contrôle approuvées pour l'examen systématique auront été compilées et, pour chaque section du système, devront amener à poser des questions telles que: Que se passe-t-il si cela se produit ? Comment cela peut-il arriver ? Quand cela peut-il arriver ? Est-ce important ? Quand des réponses positives sont apportées d'autres questions sont posées, du style: Que peut-on faire à ce sujet ? Quand cela doit-il être fait ? Existe-t-il une alternative ? etc.

Pour ce qui concerne la première partie de l'analyse, il est conseillé d'examiner dans son ensemble l'installation informatique. La deuxième partie et les autres parties impliquent un examen détaillé des parties pertinentes des systèmes informatiques eux-mêmes.

A chaque partie ou étape de l'analyse, l'objectif consiste à identifier les modes de défaillance qui conduisent à des dangers potentiels sur le système contrôlé et le degré de leur incidence. Les principales parties composantes du système informatique sont encore subdivisées et le cas échéant soumises à une analyse distincte.

References:

Space Shuttle Software. C.T. Sheridan, Datamation, Vol. 24, July 1978.

The Evolution of Fault-Tolerant Computing. Vol. 1 of Dependable Computing and Fault Tolerant Systems, Edited by A. Avizienis, H. Kopetz and J.C. Laprie, Springer-Verlag, ISBN 3-211-81941-X, 1987.

Fault Tolerance, Principle and Practices. Vol. 3 of [2] above, T. Anderson and P.A. Lee, Springer-Verlag, ISBN 3-211-82077-9, 1987.

B.34 Hazard and Operability Study (HAZOP)

Aim

To establish, by a series of systematic examinations of the component sections of the computer system and its operation, failure modes which lead to potentially hazardous situations in the controlled system.

Typical hazardous events on the controlled systems are fire, explosion, release of toxic material (chemical or nuclear) or serious economic loss.

It is assumed that the hazardous events on the system being controlled have been identified in a separate Hazard Analysis and the consequence of the hazardous events classified into degrees of seriousness.

The analysis which covers all stages of the project life cycle, from specification through design to maintenance and modification, and is intended to identify at each stage events/failure modes which could lead to potential hazards and thus eliminate them.

Description

The analysis is carried out by a team of engineers (covering computer, instrument, electrical, process, safety and operational disciplines) led by a trained specialist in hazard analysis techniques in a series of scheduled meetings.

It is important that a fixed time schedule is allocated within the project for the meetings; each one scheduled for at least half a day and for effectiveness no more than four per week to allow for maintaining the flow of accompanying documentation.

Prior to the study, agreed checklists for the systematic examination will have been compiled and for each section of the system leading questions such as: What if it happens? How can it happen? When can it happen? Does it matter? are asked. When positive answers are given further questions are asked, such as: What can be done about it? When must it be done? Is there an alternative? etc.

For the first part of the analysis it is suggested that the computer installation as a whole is examined. The second and subsequent parts involve detailed examination of the relevant parts of the computer systems itself.

At each part or stage of the analysis, the objective is to identify failure modes which lead to potential hazards on the controlled system and the degree of their effect. The major component parts of the computer system are further sub-divided and where necessary subjected to a separate analysis.

A des moments opportuns tout au long de l'analyse systématique, des réunions de revue des actions seront organisées.

Il est essentiel de conserver des comptes rendus complets des réunions, car ils constitueront une partie importante du dossier risque/sécurité du système.

Après quelques réunions, il est suggéré d'organiser une réunion de revue pour s'assurer que les actions sont suivies, que les modifications suggérées au cours des réunions d'étude ont été incorporées dans l'étude, etc.

Références:

HAZOP and HAZAN. T.A. Kletz, 1986, 2nd Edition, Institution of Chemical Engineers, 165-171 Railway Terrace, Rugby, CV1 3HQ, Royaume-Uni.

Reliability and Hazard Criteria for Programmable Electronic Systems in the Chemical Industry. E. Johnson, Proc. of «Safety and Reliability of PES», PES 3 Safety Symposium, B.K. Daniels (ed), 28-30 mai 1986, Guernsey Channel Islands, Elsevier Applied Science, 1986.

Reliability Engineering and Risk Assessment. E.J. Henlty et H. Kumamoto, Prentice-Hall, 1981.

Systems Reliability and Risk Analysis. E.G. Frenkel, Martinus Nijhogg 1984.

B.35 Analyse d'impact (référencé par l'article 16)

Objectif

Identifier l'effet provoqué par un changement ou une amélioration apporté à un système logiciel sur les autres modules de ce système ainsi que sur les autres systèmes.

Description

Avant d'apporter toute modification ou amélioration au logiciel, une analyse doit être effectuée en vue d'identifier l'impact de la modification ou de l'amélioration sur le logiciel ainsi que d'identifier les systèmes et modules logiciels affectés.

Une fois l'analyse terminée, une décision doit être prise concernant la nouvelle vérification du système logiciel. Cette vérification dépend du nombre de modules affectés, de la criticité de ces modules et de la nature du changement. Les décisions possibles sont les suivantes:

i) seul le module changé doit être revérifié;

ii) tous les modules affectés identifiés sont revérifiés;

iii) le système complet est revérifié.

B.36 Masquage d'informations/Encapsulage (référencé par td9)

Objectif

Augmenter la fiabilité et la maintenabilité du logiciel.

Description

Les données globalement accessibles par tous les composants logiciels peuvent être modifiées accidentellement ou incorrectement par l'un ou l'autre de ces composants. Il n'est pas exclu que tout changement apporté à ces structures de données nécessite un examen détaillé du code et des modifications importantes.

At suitable points throughout the systematic analysis, action review meetings will be arranged.

It is essential that comprehensive records of the meetings are kept, for they will form a substantial part of the system hazard/safety dossier.

After a number of meetings it is suggested that a review meeting be held to ensure that actions are followed up, modifications suggested during the Study meetings are incorporated into the study etc.

References:

HAZOP and HAZAN. T.A. Kletz, 1986, 2nd Edition, Institution of Chemical Engineers, 165-171 Railway Terrace, Rugby, CV1 3HQ, UK.

Reliability and Hazard Criteria for Programmable Electronic Systems in the Chemical Industry. E. Johnson, Proc. of 'Safety and Reliability of PES', PES 3 Safety Symposium, B.K. Daniels (ed), 28-30 May 1986, Guernsey Channel Islands, Elsevier Applied Science, 1986.

Reliability Engineering and Risk Assessment. E.J. Henlty and H. Kumamoto, Prentice-Hall, 1981.

Systems Reliability and Risk Analysis. E.G. Frenkel, Martinus Nijhogg 1984.

B.35 Impact Analysis (referenced by clause 16)

Aim

To identify the effect that a change or an enhancement to a software system will have to other modules in that software system as well as to other systems.

Description

Prior to a modification or enhancement being performed on the software, an analysis shall be undertaken to identify the impact of the modification or enhancement on the software and to also identify the affected software systems and modules.

After the analysis has been completed a decision is required concerning the reverification of the software system. This depends on the number of modules affected, the criticality of the affected modules and the nature of the change. The possible decisions are:

i) only the changed module to be reverified;

ii) all identified affected modules are reverified; and

iii) the complete system is reverified.

B.36 Information Hiding/Encapsulation (referenced by dt9)

Aim

To increase the reliability and maintainability of software.

Description

Data that is globally accessible to all software components can be accidentally or incorrectly modified by any of these components. Any changes to these data structures may require detailed examination of the code and extensive modifications.

Le masquage d'informations constitue une approche générale permettant de minimiser ces difficultés. Les structures de données clés sont «masquées» et ne peuvent être manipulées que par le biais d'un ensemble défini de procédures d'accès. Ceci permet de modifier des structures internes et d'ajouter de nouvelles procédures sans affecter le comportement fonctionnel du reste du logiciel. Par exemple, un répertoire de noms peut avoir des procédures d'accès Insertion, Suppression et Recherche. Les procédures d'accès et les structures internes peuvent être réécrites (par exemple, pour utiliser une différente méthode de consultation ou pour enregistrer les noms sur un disque dur) sans affecter le comportement logique du reste du logiciel utilisant ces procédures.

Ce concept de type de données abstrait est directement pris en charge dans un certain nombre de langages de programmation, mais le principe de base peut être appliqué quel que soit le langage utilisé.

Références:

Software Engineering: Planning for Change, D.A. Lamb, Prentice-Hall, 1988.

On the Design and Development of Program Families, D.L. Parnas, IEEE Trans. SE-2, mars 1976.

B.37 Tests d'interface (référencé par l'article 10)

Objectif

Démontrer que les interfaces des sous-programmes ne contiennent pas d'erreurs ou pas d'erreurs amenant à des défaillances dans une application particulière du logiciel ou détecter toutes les erreurs éventuellement significatives.

Description

Plusieurs niveaux de détail ou d'état complet des tests sont réalisables. Les niveaux les plus importants sont les tests de

– toutes les variables d'interface dans leurs positions extrêmes;
– toute variable d'interface individuellement sur ses valeurs extrêmes avec les autres variables d'interface sur leurs valeurs normales;
– toutes les valeurs du domaine de chaque variable d'interface avec les autres variables d'interface sur leurs valeurs normales;
– toutes les valeurs de toutes les variables combinées. Il n'est pas exclu que ceci ne soit réalisable que pour les petites interfaces;
– les conditions de test spécifiées applicables à chaque appel de chaque sous-routine.

Ces tests sont particulièrement importants si leurs interfaces ne contiennent pas d'assertions détectant les valeurs incorrectes des paramètres. Ils sont également essentiels après la génération de nouvelles configurations de sous-programmes préexistants.

B.38 Sous-ensemble de langage (référencé par l'article 10 et td4)

Objectif

Réduire la probabilité d'introduction de défauts de programmation et augmenter la probabilité de détection des défauts restants.

Information hiding is a general approach for minimising these difficulties. The key data structures are 'hidden' and can only be manipulated through a defined set of access procedures. This allows the internal structures to be modified or further procedures to be added without affecting the functional behaviour of the remaining software. For example, a name directory might have access procedures Insert, Delete and Find. The access procedures and internal data structures could be re-written (e.g. to use a different look-up method or to store the names on a hard disk) without affecting the logical behaviour of the remaining software using these procedures.

This concept of an abstract data type is directly supported in a number of programming languages, but the basic principle can be applied whatever programming language is used.

References:

Software Engineering: Planning for Change, D.A. Lamb, Prentice Hall, 1988.

On the Design and Development of Program Families, D.L. Parnas, IEEE Trans. SE-2, March 1976.

B.37 Interface Testing (referenced by clause 10)

Aim

To demonstrate that interfaces of subprograms do not contain any errors or any errors that lead to failures in a particular application of the software or to detect all errors that may be relevant.

Description

Several levels of detail or completeness of testing are feasible. The most important levels are testing

– all interface variables at their extreme positions;
– all interface variable individually at their extreme values with other interface variables at normal values;
– all values of the domain of each interface variable with other interface variables at normal values;
– all values of all variables in combination. This may only be feasible for small interfaces;
– the specified test conditions relevant to each call of each subroutine.

These tests are particularly important if their interfaces do not contain assertions that detect incorrect parameter values. They are also important after new configurations of pre-existing subprograms have been generated.

B.38 Language Subset (referenced by clause 10 and dt4)

Aim

To reduce the probability of introducing programming faults and increase the probability of detecting any remaining faults.

Description

Le langage est examiné pour identifier les structures de programmation qui sont sujettes à erreurs ou difficiles à analyser, en utilisant par exemple des méthodes d'analyse statique. Un sous-ensemble de langage excluant ces structures est alors défini.

B.39 Mémorisation des cas exécutés (référencé par l'article 9)

Objectif

Forcer le logiciel dans un état sûr s'il exécute un chemin non autorisé.

Description

Pendant le processus d'autorisation, tous les détails importants de chaque exécution du programme sont enregistrés. Pendant le fonctionnement normal, chaque exécution du programme est comparée à l'ensemble des exécutions autorisées. Si elle diffère, une action de sécurité est décidée.

L'enregistrement de l'exécution peut être la séquence de chemins de décision-à-décision individuels (DD paths) ou la séquence d'accès individuel aux tableaux, aux enregistrements ou aux volumes, ou les deux.

Différentes méthodes de stockage des chemins d'exécution sont possibles. Les méthodes de codage de Nash peuvent être utilisées pour réduire la séquence d'exécution à un grand nombre ou à une séquence de nombres. Pendant le fonctionnement normal, il faut vérifier la valeur du chemin d'exécution par rapport aux modèles enregistrés avant toute opération de sortie.

Puisqu'il existe un très grand nombre de combinaisons possibles des chemins de décision-à-décision au cours d'un seul programme, il n'est parfois pas réalisable de traiter les programmes comme un tout. Dans ce cas, la technique peut être appliquée au niveau du module.

Référence:

Fail-safe Software – Some Principles and a Case Study. W. Ehrenberger, Proc. SARSS 1987, Altringham, Manchester, Royaume-Uni, Elsevier Applied Science, 1987.

B.40 Bibliothèque de modules et de composants sécurisés/vérifiés (référencé par l'article 10)

Objectif

Eviter la nécessité d'une nouvelle validation ou conception extensive des modules logiciels et des composants matériels pour chaque nouvelle application. Avantager également les conceptions qui n'ont pas été validées formellement ou rigoureusement mais pour lesquelles on dispose d'un historique opérationnel considérable.

Description

Les systèmes électroniques programmables correctement conçus et structurés sont constitués d'un certain nombre de composants et de modules matériels et logiciels qui sont clairement distincts et qui interagissent entre eux de façons clairement définies.

Description

The language is examined to identify programming constructs which are either error-prone or difficult to analyse, for example, using static analysis methods. A language subset is then defined which excludes these constructs.

B.39 Memorising Executed Cases (referenced by clause 9)

Aim

To force the software to fail safe if it executes an unlicensed path.

Description

During licensing a record is made of all relevant details of each program execution. During normal operation each program execution is compared with the set of the licensed executions. If it differs, a safety action is taken.

The execution record can be the sequence of the individual decision-to-decision paths (DDpaths) or the sequence of the individual accesses to arrays, records or volumes, or both.

Different methods of storing execution paths are possible. Nash-coding methods can be used to map the execution sequence onto a single large number or sequence of numbers. During normal operation the execution path value must be checked against the stored cases before any output operation occurs.

Since the possible combinations of decision-to-decision paths during one program is very large, it may not be feasible to treat programs as a whole. In this case, the technique can be applied at module level.

Reference:

Fail-safe Software – Some Principles and a Case Study. W. Ehrenberger, Proc. SARSS 1987, Altringham, Manchester, UK, Elsevier Applied Science, 1987.

B.40 Library of Trusted/Verified Modules and Components (referenced by clause 10)

Aim

To avoid the need for software modules and hardware component designs to be extensively revalidated or redesigned for each new application. Also to advantage designs which have not been formally or rigorously validated but for which considerable operational history is available.

Description

Well-designed and structured PESs are made up of a number of hardware and software components and modules which are clearly distinct and which interact with each other in clearly defined ways.

Les différents systèmes électroniques conçus pour des applications diverses contiendront un nombre de modules ou de composants identique ou très similaire. La création d'une bibliothèque regroupant ces modules généralement applicables permet à plusieurs applications de partager une grande partie des ressources nécessaires à la validation.

En outre, l'utilisation de ces modules dans plusieurs applications apporte une preuve empirique de la réussite de leur utilisation opérationnelle. Cette preuve empirique améliore considérablement la confiance éventuelle des utilisateurs envers les modules.

B.41 Modèles de Markov (référencé par l'article 14)

Objectif

Evaluer la fiabilité, la sécurité ou la disponibilité d'un système.

Description

Un graphe du système est dessiné. Il représente le statut du système par rapport aux états de défaillance (les états de défaillance sont représentés par les noeuds du graphe). Les arcs entre les noeuds qui représentent les événements de défaillance ou de réparation sont pondérés par les taux de défaillance ou de réparation correspondants. Il est à noter que les événements, les états et les taux de défaillance peuvent être détaillés de manière à obtenir une description précise du système, par exemple défaillances détectées ou non détectées, manifestation d'une défaillance plus importante, etc.

La technique de Markov est adaptée à la modélisation des systèmes redondants dans lesquels le niveau de redondance varie avec le temps à cause de la défaillance et de la réparation des composants. Il est difficile d'adapter des méthodes classiques telles que AMDE et Arbre des causes à la modélisation des effets des défaillances tout au long du cycle de vie du système puisqu'il n'existe aucune formule combinatoire simple pour le calcul des probabilités correspondantes.

Pour les cas les plus simples, les formules qui décrivent les probabilités du système sont aisément disponibles dans les livres ou peuvent être calculées manuellement. Pour les cas plus complexes, il existe certaines méthodes de simplification (réduction d'états). Pour les cas très complexes, les résultats peuvent être calculés par simulation informatique du graphe.

Références:

The Theory of Stochastic Processes. R.E. Cox et H.D. Miller, Methuen et Co. Ltd, London, Royaume-Uni, 1968.

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

Reliability Handbook. B.A. Koslov et L.A. Ushakov, Holt Rinehart et Winston Inc., New York 1970.

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

B.42 Métriques (référencé par les articles 11 et 14)

Objectif

Prédire les attributs des programmes à partir des propriétés du logiciel lui-même plutôt que de le faire à partir de son développement ou de son historique de test.

Different PESs designed for differing applications will contain a number of modules or components which are the same or very similar. Building up a library of such generally applicable modules allows much of the resource necessary for validating the designs to be shared by more than one application.

Furthermore, the use of such modules in multiple applications provides empirical evidence of successful operational use. This empirical evidence justifiably enhances the trust which users are likely to have in the modules.

B.41 Markov Models (referenced by clause 14)

Aim

To evaluate the reliability, safety or availability of a system.

Description

A graph of the system is constructed. The graph represents the status of the system with regard to its failure states (the failure states are represented by the nodes of the graph). The edges between nodes, which represent the failure events or repair events, are weighted with the corresponding failure rates or repair rates. Note that the failure events, states and rates can be detailed in such a way that a precise description of the system is obtained, e.g. detected or undetected failures, manifestation of a larger failure, etc.

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

In the simplest cases, the formulae which describe the probabilities of the system are readily available in the literature or can be calculated manually. In more complex cases, some methods of simplification (absorbing states) exist. For very complex cases results can be calculated by computer simulation of the graph.

References:

The Theory of Stochastic Processes. R.E. Cox and H.D. Miller, Methuen and Co. Ltd, London, UK, 1968.

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

Reliability Handbook. B.A. Koslov and L.A. Ushakov, Holt Rinehart and Winston Inc., New York, 1970.

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

B.42 Metrics (referenced by clauses 11 and 14)

Aim

To predict the attributes of programs from properties of the software itself rather than from its development or test history.

Description

Ces modèles évaluent certaines propriétés structurelles du logiciel et les associent à un attribut recherché tel que la fiabilité ou la complexité. Des outils logiciels sont requis pour évaluer la plupart des mesures. Certaines des métriques pouvant être appliquées sont énumérées ci-dessous:

– complexité théorique du graphe: cette mesure pouvant être appliquée au début du cycle de vie pour évaluer les compromis repose sur la complexité du graphe de contrôle du programme, représenté par un nombre cyclomatique;
– nombre de chemins permettant d'activer un certain module (accessibilité): plus le module est accessible, plus il est facile à mettre au point;
– science logicielle: cette mesure calcule la longueur du programme en comptant le nombre d'opérateurs et d'opérandes. Elle fournit une mesure de la complexité et estime les ressources de développement;
– nombre d'entrées et de sorties par module: la minimisation du nombre de points d'entrée/sortie est une caractéristique clé de la conception structurée et des techniques de programmation.

Références:

A Complexity Measure. T.J. McCabe, IEEE Trans. on Software Engineering, Vol. SE-2, No. 4, décembre 1976.

Models and Measurements for Quality Assessments of Software. S.N. Mohanty, ACM Computing Surveys, Vol. 11, No. 3, septembre 1979.

Elements of Software Science. M.H. Halstead, Elsevier, North Holland, New York, 1977.

B.43 Approche modulaire (référencé par td9)

Objectif

Décomposition d'un système logiciel en petites parties compréhensibles de manière à limiter la complexité du système.

Description

Une approche modulaire ou modularisation contient plusieurs règles applicables aux phases de conception, de codage et de maintenance d'un projet logiciel. Ces règles varient selon la méthode de conception employée au cours de la conception. La plupart des méthodes intègre les règles suivantes:

– un module doit n'avoir qu'une seule tâche ou fonction bien définie à remplir;
– les connexions entre les modules doivent être limitées et strictement définies, la cohérence à l'intérieur d'un module doit être forte;
– les collections de sous-programmes doivent être construites de manière à fournir plusieurs niveaux de modules;
– la taille des sous-programmes doit être limitée à une certaine valeur spécifique, en principe 2 à 4 tailles d'écran;
– les sous-programmes ne doivent comporter qu'une seule entrée et une seule sortie;
– les modules doivent communiquer avec d'autres modules via leurs interfaces. Lorsqu'on utilise des variables globales ou communes, elles doivent être bien structurées, l'accès doit être contrôlé et leur utilisation justifiée dans chaque instance;
– toutes les interfaces des modules doivent être pleinement documentées;
– chaque module doit masquer quelque chose de son environnement;

Description

These models evaluate some structural properties of the software and relate this to a desired attribute such as reliability or complexity. Software tools are required to evaluate most of the measures. Some of the metrics which can be applied are given below:

– Graph Theoretic Complexity: this measure can be applied early in the life cycle to assess trade-offs, and is based on the complexity of the program control graph, represented by its cyclomatic number;
– number of ways to activate a certain module (accessibility): the more a module can be accessed, the more likely it is to be debugged;
– Software Science: this measure computes the program length by counting the number of operators and operands. It provides a measure of complexity and estimates development resources;
– number of entries and exits per module: minimising the number of entry/exit points is a key feature of structured design and programming techniques.

References:

A Complexity Measure. T.J. McCabe, IEEE Trans. on Software Engineering, Vol. SE-2, No. 4, Dec. 1976.

Models and Measurements for Quality Assessments of Software. S.N. Mohanty, ACM Computing Surveys, Vol. 11, No. 3, Sep. 1979.

Elements of Software Science. M.H. Halstead, Elsevier, North Holland, New York, 1977.

B.43 Modular Approach (referenced by dt9)

Aim

Decomposition of a software systems into small comprehensible parts in order to limit the complexity of the system.

Description

A Modular Approach or modularisation contains several rules for the design, coding and maintenance phases of a software project. These rules vary according to the design method employed during design. Most methods contain the following rules:

– a module shall have a single well-defined task or function to fulfil;
– connections between modules shall be limited and strictly defined, coherence in one module shall be strong;
– collections of subprograms shall be built providing several levels of modules;
– subprogram sizes shall be restricted to some specified value typically 2 to 4 screen sizes;
– subprograms shall have a single entry and a single exit only;
– modules shall communicate with other modules via their interfaces. Where global or common variables are used they shall be well structured, access shall be controlled and their use shall be justified in each instance;
– all module interfaces shall be fully documented;
– each module shall hide something from its environment;

– toutes les interfaces des modules doivent contenir le nombre de paramètres minimum nécessaire pour la fonction des modules; et

– une restriction adaptée du nombre des paramètres doit être spécifiée; en règle générale ils sont au nombre de 5.

B.44 Simulation de Monte Carlo

Objectif

Simuler dans le logiciel les phénomènes de l'environnement réel à l'aide de nombres aléatoires.

Description

Les simulations de Monte Carlo sont utilisées pour résoudre des problèmes. Ces problèmes se divisent en deux catégories:

– probabiliste, dans laquelle les nombres aléatoires sont utilisés pour générer un phénomène stochastique de l'environnement réel; et

– déterministe, dans laquelle les problèmes sont convertis mathématiquement en un problème probabiliste équivalent.

La simulation de Monte Carlo injecte des flux de nombres aléatoires pour simuler le bruit sur un signal analytique ou pour ajouter des biais ou des tolérances. Cette simulation est exécutée pour produire un grand échantillon à partir duquel sont obtenus des résultats statistiques.

Quand on utilise les simulations de Monte Carlo, il faut veiller à s'assurer que les biais, les tolérances ou le bruit sont des valeurs raisonnables.

Un principe général des simulations de Monte Carlo consiste à énoncer de nouveau et à reformuler le problème de manière à ce que les résultats obtenus soient aussi précis que possible plutôt que de s'attaquer au problème comme il avait été énoncé initialement.

B.45 Modélisation des performances (référencé par td2 et td5)

Objectif

Assurer que la capacité de fonctionnement du système est suffisante pour satisfaire aux exigences spécifiées.

Description

La spécification des exigences inclut les prescriptions relatives à la capacité de traitement et de réponse de fonctions spécifiques, combinées parfois à des contraintes applicables à l'utilisation des ressources totales du système. La conception suggérée du système est comparée aux besoins requis en

– définissant un modèle des processus du système et de leurs interactions,

– identifiant l'utilisation des ressources par chaque processus (par exemple le temps processeur, la largeur de bande des communications, les périphériques de stockage, etc.),

– identifiant la distribution des demandes exigées du système dans des conditions moyennes et dans les conditions les plus mauvaises,

– calculant la capacité de traitement moyenne et la plus défavorable et les temps de réponse pour chaque fonction du système.

– any modules interface shall contain the minimum number of parameters necessary for the modules function; and
– a suitable restriction of parameter number shall be specified, typically 5.

B.44 Monte Carlo Simulation

Aim

To simulate phenomenon of the real world in the software using random numbers.

Description

Monte Carlo simulations are used to solve problems. These problems fall into two classes:

– probabilistic, where random numbers are used to generate a real-world stochastic phenomenon; and
– deterministic, which are mathematically translated into an equivalent probabilistic problem.

Monte Carlo simulation injects a random number streams to simulate noise on an analytic signal or to add random biases or tolerances. The Monte Carlo simulation is run to produce a large sample from which statistical results are obtained.

When using Monte Carlo simulations care must be taken to ensure that the biases, tolerances or noise are reasonable values.

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

B.45 Performance Modelling (referenced by dt2 and dt5)

Aim

To ensure that the working capacity of the system is sufficient to meet the specified requirements.

Description

The requirements specification includes throughput and response requirements for specific functions, perhaps combined with constraints on the use of total system resources. The proposed system design is compared against the stated requirements by

– defining a model of the system processes, and their interactions,
– identifying the use of resources by each process, for example, processor time, communications bandwidth, storage devices, etc.),
– identifying the distribution of demands placed upon the system under average and worst-case conditions,
– computing the mean and worst case throughput and response times for the individual system functions.

Pour les systèmes simples, il n'est pas exclu qu'une solution analytique soit possible. En revanche, pour les systèmes plus complexes, une certaine forme de simulation est indispensable pour obtenir des résultats précis.

Avant la modélisation détaillée, on peut utiliser un contrôle du «budget des ressources» plus simple qui additionne les exigences applicables aux ressources de tous les processus.

Si les exigences excèdent la capacité prévue du système, la conception n'est pas réalisable. Même si la conception réussit ce contrôle, il n'est pas exclu que la modélisation des performances montre que des retards et des temps de réponse excessifs se produisent en raison du manque de ressources. Pour éviter cette situation, les ingénieurs imaginent souvent des systèmes utilisant une certaine fraction (50 % par exemple) des ressources totales de manière à limiter la probabilité manque de ressources.

B.46 Exigences en matière de performance (référencé par td6)

Objectif

Etablir que les exigences du système logiciel en matière de performance ont été observées.

Description

Une analyse est effectuée sur les spécifications des exigences du système et du logiciel afin d'identifier toutes les exigences générales et spécifiques, explicites et implicites en matière de performance.

A tour de rôle, toutes les exigences de performance sont examinées pour déterminer

– les critères de réussite auxquels il faut satisfaire,
– si une mesure de la satisfaction aux critères de réussite peut être définie,
– la précision potentielle de telles mesures,
– les étapes du projet au cours desquelles les mesures peuvent être estimées, et
– les étapes du projet au cours desquelles les mesures peuvent être effectuées.

La faisabilité de chaque exigence de performance est alors analysée de manière à obtenir une liste des exigences de performance, des critères de réussite et des mesures potentielles. Les principaux objectifs sont les suivants:

i) chaque exigence de performance est associée à une mesure au moins;
ii) chaque fois que possible, on sélectionne des mesures précises et efficaces qui peuvent être utilisées aussitôt que possible dans le processus de développement;
iii) les exigences essentielles et optionnelles des performances et les critères de réussite sont identifiés; et
iv) chaque fois possible, on doit tirer parti de la possibilité d'utiliser une seule mesure pour plusieurs prescriptions de performance.

B.47 Tests probabilistes (référencé par les articles 11 et 13)

Objectif

Obtenir un chiffre quantitatif concernant les propriétés de fiabilité du logiciel à l'étude. Il est admis que ce chiffre concerne les niveaux de confiance et de signification associées à

i) une probabilité de défaillance par demande,
ii) une probabilité de défaillance pendant une certaine période de temps, et
iii) une probabilité de limitation d'erreur.

For simple systems, an analytic solution may be possible whilst for more complex systems, some form of simulation is required to obtain accurate results.

Before detailed modelling, a simpler 'resource budget' check can be used which sums the resources requirements of all the processes. If the requirements exceed designed system capacity, the design is infeasible.

Even if the design passes this check, performance modelling may show that excessive delays and response times occur due to resource starvation. To avoid this situation engineers often design systems to use some fraction (e.g. 50 %) of the total resources so that the probability of resource starvation is reduced.

B.46 Performance Requirements (referenced by dt6)

Aim

To establish that the performance requirements of a software system have been satisfied.

Description

An analysis is performed of both the system and the software requirements specifications to identify all general and specific, explicit and implicit performance requirements.

Each performance requirement is examined in turn to determine

– the success criteria to be obtained,
– whether a measure against the success criteria can be obtained,
– the potential accuracy of such measurements,
– the project stages at which the measurements can be estimated, and
– the project stages at which the measurements can be made.

The practicability of each performance requirement is then analysed in order to obtain a list of performance requirements, success criteria and potential measurements. The main objectives are:

i) each performance requirement is associated with at least one measurement;
ii) where possible, accurate and efficient measurements are selected which can be used as early in the development process as possible;
iii) essential and optional performance requirements and success criteria are identified; and
iv) where possible, advantage shall be taken of the possibility of using a single measurement for more than one performance requirement.

B.47 Probabilistic Testing (referenced by clauses 11 and 13)

Aim

To gain a quantitative figure about the reliability properties of the investigated software. This figure may address the related levels of confidence and significance and

i) a failure probability per demand,
ii) a failure probability during a certain period of time, and
iii) a probability of error containment.

A partir de ces chiffres, il est permis de dériver d'autres paramètres, parmi lesquels

– la probabilité d'une exécution sans défaillance,
– la probabilité de survie,
– la disponibilité,
– le MTBF ou le taux de défaillance, et
– la probabilité d'une exécution sûre.

Description

Les considérations probabilistes reposent sur un test probabiliste ou sur une expérience d'exploitation. En général, il existe un grand nombre de scénarios de test pour les cas d'exploitation observés.

Pour faciliter les tests, des aides automatiques sont en principe utilisés. Ils concernent les détails de la fourniture de données de test et la surveillance des sorties des tests. Des tests poussés sont exécutés sur les grands ordinateurs hôtes avec le contexte approprié de simulation des processus. Les données de test sont sélectionnées à la fois selon des points de vue systématiques et aléatoires. Les premiers concernent le contrôle global du test, par exemple, en garantissant un profil des données de test. La sélection aléatoire prend chaque scénario de test individuellement en détail.

Les conditions de test, les exécutions des tests et les surveillances des tests sont déterminées individuellement selon les objectifs détaillés des tests décrits plus haut. Les autres conditions importantes sont définies par les conditions mathématiques préalables à remplir pour autoriser l'évaluation du test dans l'optique de l'objectif voulu du test.

Il est admis que des chiffres probabilistes sur le comportement de tout objet de test soient également dérivés de l'expérience d'exploitation. Sous réserve de remplir les mêmes conditions, on peut appliquer les mêmes mathématiques que pour l'évaluation des résultats des tests.

B.48 Simulation du processus (référencé par td3)

Objectif

Tester la fonction d'un système logiciel ainsi que de son interface avec le monde extérieur, sans lui permettre de modifier la situation réelle d'aucune façon.

Description

La création, aux seules fins de tests, d'un système qui simule le comportement du système à contrôler par le système soumis au test.

Il est admis que la simulation soit de type logiciel seulement ou une combinaison de logiciel et de matériel. Elle doit

– fournir toutes les entrées du système soumis au test qui existeront quand le système sera installé,
– répondre aux sorties du système d'une façon représentant fidèlement l'installation contrôlée,
– prévoir des entrées opérateur afin de fournir toutes les perturbations que le système soumis au test est tenu de gérer.

Quand le logiciel est soumis aux tests, il est admis que la simulation soit une simulation du matériel cible avec ses entrées et sorties.

From these figures other parameters may be derived such as

– probability of failure-free execution,
– probability of survival,
– availability,
– MTBF or failure rate, and
– probability of safe execution.

Description

Probabilistic considerations are either based on a probabilistic test or on operating experience. Usually the number of test cases of observed operating cases is very large.

In order to facilitate testing, usually automatic aids are taken. They concern the details of test data provision and test output supervision. Large tests are run on large host computers with the appropriate process simulation periphery. Test data is selected both according to systematic and random view points. The first concern the overall test control, for example, guarantee a test data profile. The random selection takes the individual test cases in detail.

Individual test harnesses, test executions and test supervisions are determined by the detailed test aims as described above. Other important conditions are given through the mathematical prerequisites to be fulfilled in order to enable the test evaluation in view of the intended test aim.

Probabilistic figures about the behaviour of any test object may also be derived from operating experience. Provided the same conditions are met, the same mathematics can be applied as for the evaluation of test results.

B.48 Process Simulation (referenced by dt3)

Aim

To test the function of a software system, together with its interface to the outside world, without allowing it to modify the real world in any way.

Description

The creation of a system, for testing purposes only, which mimics the behaviour of the system to be controlled by the system under test.

The simulation may be software only or a combination of software and hardware. It must

– provide all the inputs of the system under test which will exist when the system is installed,
– respond to outputs from the system in a way which faithfully represents the controlled plant,
– have provision for operator inputs to provide any perturbations with which the system under test is required to cope.

When software is being tested the simulation may be a simulation of the target hardware with its inputs and outputs.

Références:

A Software Simulator – An Aid to Plant Commissioning. S. Nunns, EWICS TC7.

Physical Fault Simulation. F. Morillon, EWICS Numéro de document WP460.

B.49 Prototypage/Animation (référencé par td3 et td5)

Objectif

Vérifier la faisabilité de l'implémentation du système par rapport aux contraintes définies. Communiquer au client l'interprétation du système par l'auteur des spécifications afin de repérer les méprises.

Description

Un sous-ensemble de fonctions, de contraintes et de prescriptions de performance du système est sélectionné. Un prototype est élaboré à l'aide d'outils de haut niveau. A cette étape, des contraintes telles que l'ordinateur cible, le langage d'implémentation, la taille du programme, la maintenabilité, la fiabilité et la disponibilité n'ont pas à être prises en compte. Le prototype est évalué par rapport aux critères du client et il est permis de modifier les exigences du système à la lumière de cette évaluation.

Références:

Proc. Working Conference on Prototyping. Namur octobre 1983, Budde *et al.*, Springer-verlag, 1984.

Using an executable specification language for an information system. S. Urban *et al.*, IEEE Trans. Software Engineering, Vol. SE-11 No. 7, juillet 1985.

B.50 Bloc de rattrapage (référencé par l'article 9)

Objectif

Améliorer la vraisemblance du programme exécutant la fonction pour laquelle il est conçu.

Description

Plusieurs sections de programme différentes sont écrites, souvent indépendamment, dans le but que chacune d'elle exécute la même fonction souhaitée. Le programme final est construit à partir de ces sections. La première section, appelée section principale, est exécutée en premier. Cette exécution est suivie d'un test d'acceptation du résultat calculé. Si le test est réussi, le résultat est alors accepté et transmis aux parties suivantes du système. Si le test échoue, tous les effets secondaires de la première section sont réinitialisés et la seconde section, appelée première alternative, est exécutée. Cette exécution, également suivie d'un test d'acceptation, est traitée comme dans le premier cas. Une deuxième, une troisième et même des alternatives supplémentaires peuvent être proposées le cas échéant.

Référence:

System Structure for Software Fault Tolerance. B. Randall, IEEE Trans. Software Engineering, Vol. SE-1, No. 2, 1975.

References:

A Software Simulator – An Aid to Plant Commissioning. S. Nunns, EWICS TC7.

Physical Fault Simulation. F. Morillon, EWICS Document number WP460.

B.49 Prototyping/Animation (referenced by dt3 and dt5)

Aim

To check the feasibility of implementing the system against the given constraints. To communicate the specifier's interpretation of the system to the customer, in order to locate misunderstandings.

Description

A sub-set of system functions, constraints, and performance requirements are selected. A prototype is built using high-level tools. At this stage, constraints such as the target computer, implementation language, program size, maintainability, reliability and availability need not be considered. The prototype is evaluated against the customer's criteria and the system requirements may be modified in the light of this evaluation.

References:

Proc. Working Conference on Prototyping. Namur Oct 1983, Budde *et al.*, Springer-verlag, 1984.

Using an executable specification language for an information system. S. Urban *et al.*, IEEE Trans. Software Engineering, Vol. SE-11 No. 7, July 1985.

B.50 Recovery Block (referenced by clause 9)

Aim

To increase the likelihood of the program performing its intended function.

Description

Several different program sections are written, often independently, each of which is intended to perform the same desired function. The final program is constructed from these sections. The first section, called the primary, is executed first. This is followed by an acceptance test of the result it calculates. If the test is passed then the result is accepted and passed on to subsequent parts of the system. If it fails, any side-effects of the first are reset and the second section, called the first alternative, is executed. This too is followed by an acceptance test and is treated as in the first case. A second, third or even more alternatives can be provided if desired.

Reference:

System Structure for Software Fault Tolerance. B. Randall, IEEE Trans. Software Engineering, Vol. SE-1, No. 2, 1975.

B.51 Schéma bloc de la fiabilité (référencé par l'article 14)

Objectif

Modéliser, sous forme de schéma, l'ensemble d'événements qui doivent avoir lieu et les conditions qui doivent être remplies pour que l'opération d'un système ou d'une tâche soit réussie.

Description

La cible de l'analyse est représentée sous forme d'un chemin de réussite composé de blocs, de lignes et de portes logiques. Un chemin de réussite commence d'un côté du schéma et continue via les blocs et les portes vers l'autre côté du schéma. Un bloc représente une condition ou un événement, et le chemin peut se poursuivre si la condition est vraie ou si l'événement a lieu. Si le chemin arrive à une porte, il continue si la logique de la porte est remplie. S'il atteint un noeud, il est admis qu'il continue le long de toutes les lignes de sortie. S'il existe au moins un chemin de réussite dans le schéma le système analysé fonctionne correctement.

Références:

System Reliability Engineering Methodology: A Division of the State of the Art. J.B. Fusseland et J.S. Arend, Nuclear Safety 20(5), 1979.

Fault Tree Handbook. W.E. Vesely *et al*., NUREG-0942, Division of System Safety Office at Nuclear Reactor Regulation, U.S. Nuclear Regulatory Commission, Washington, D.C. 20555, 1981.

B.52 Temps de réponse et contraintes de place mémoire (référencé par td6)

Objectif

Assurer que le système satisfera à ses exigences temporelles et de mémoire.

Description

La spécification des exigences destinées au système et au logiciel intègre des exigences relatives à la mémoire et à la réponse de fonctions spécifiques, parfois combinées à des contraintes sur l'utilisation des ressources totales du système. Une analyse est effectuée pour identifier la répartition des demandes dans des conditions moyennes et dans les conditions les plus défavorables. Cette analyse exige des estimations de l'usage des ressources et du temps requis pour chaque fonction du système. Ces estimations peuvent être obtenues de diverses manières, par exemple, par comparaison avec un système existant ou par prototypage et évaluation des performances des systèmes temps-réel critiques.

B.53 Rattrapage par réexécution (référencé par l'article 9)

Objectif

Tenter un rétablissement fonctionnel à partir d'une condition de défaut détectée, à l'aide de mécanismes de tentative de réexécution.

Description

Au cas où une condition de défaut ou d'erreur est détectée, des tentatives sont effectuées pour rétablir la situation en réexécutant le même code. Le rattrapage par réexécution peut être aussi complet qu'une procédure de réinitialisation et de redémarrage ou n'être qu'une petite tâche de replanification et de relance, après expiration d'un délai ou action d'un chien de garde.

B.51 Reliability Block Diagram (referenced by clause 14)

Aim

To model, in a diagrammatic form, the set of events that must take place and conditions which must be fulfilled for a successful operation of a system or a task.

Description

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

References:

System Reliability Engineering Methodology: A Division of the State of the Art. J.B. Fusseland J.S. Arend, Nuclear Safety 20(5), 1979.

Fault Tree Handbook. W.E. Vesely *et al.*, NUREG-0942, Division of System Safety Office at Nuclear Reactor Regulation, U.S. Nuclear Regulatory Commission, Washington, D.C. 20555, 1981.

B.52 Response Timing and Memory Constraints (referenced by dt6)

Aim

To ensure that the system will meet its temporal and memory requirements.

Description

The requirements specification for the system and the software includes memory and response requirements for specific functions, perhaps combined with constraints on the use of total system resources. An analysis is performed which will identify the distribution demands under average and worst-case conditions. This analysis requires estimates of the resource usage and elapsed time of each system function. These estimates can be obtained in several ways, for example, comparison with an existing system or the prototyping and bench-marking of time critical systems.

B.53 Re-try Fault Recovery Mechanisms (referenced by clause 9)

Aim

To attempt functional recovery from a detected fault condition by re-try mechanisms.

Description

In the event of a detected fault or error condition, attempts are made to recover the situation by re-executing the same code. Recovery by re-try can be as complete as a re-boot and a re-start procedure or a small re-scheduling and re-starting task, after a software time-out or a task watchdog action.

Les techniques de rattrapage par réexécution sont communément utilisées pour le rétablissement des défauts ou des erreurs de transmission et les conditions de tentative de réexécution pourraient être indiquées par une erreur du protocole de communication (somme de contrôle) ou par une temporisation de la réponse d'acquittement de la communication.

Référence:

The Theory and Practise of Reliable System Design. D.P. Siewiorek et R.S. Schwarz, Digital Press.

B.54 Sécurité contrôlée (Safety Bag) (référencé par l'article 9)

Objectif

Protéger le logiciel contre les défauts résiduels de spécification et d'implémentation qui nuisent à la sécurité.

Description

La Sécurité Contrôlée repose sur un moniteur externe, mis en oeuvre sur un ordinateur indépendant d'après une spécification différente. Ce contrôle a pour seul objectif de s'assurer que l'ordinateur principal exécute des actions sûres, mais pas nécessairement correctes. Le contrôle supervise en permanence l'ordinateur principal. Il empêche le système d'entrer dans un état «non sûr». En outre, si le contrôle détecte que l'ordinateur principal entre dans un état potentiellement risqué, le système doit être ramené à un état «sûr» soit par le contrôle, soit par l'ordinateur principal lui-même.

Référence:

Using AI Techniques to Improve Software Safety. Proc. IFAC SAFECOMP 88, Sarlat, France, octobre 1986, Pergamon Press 1986.

B.55 Analyse des chemins insidieux (référencé par td8)

Objectif

Détecter un chemin ou un flux logique inattendu dans un système qui, dans certaines conditions, initialise une fonction non souhaitée ou inhibe une fonction souhaitée.

Description

Il est admis qu'un chemin insidieux d'un schéma se compose de matériel, de logiciel, d'actions de l'opérateur ou de toute combinaison de ces éléments. Les chemins insidieux ne sont pas le résultat d'une défaillance matérielle mais de conditions latentes créées par inadvertance dans le système et codées dans les programmes logiciels. Ils peuvent provoquer un dérangement du système dans certaines conditions.

Les catégories de chemins insidieux sont les suivantes:

– des chemins insidieux qui provoquent le flux d'un courant, d'une énergie ou d'une séquence logique le long d'un chemin inattendu ou dans une direction imprévue;

– des synchronisations insidieuses dans lesquelles les événements se produisent en séquence inattendue ou conflictuelle;

– des indications insidieuses qui provoquent un affichage ambigu ou erroné des conditions de fonctionnement du système et qui risquent d'aboutir à une action indésirable de l'opérateur;

Re-try techniques are commonly used in communication fault or error recovery and re-try conditions could be flagged from a communication protocol error (check sum etc.) or from a communication acknowledgement response time-out.

Reference:

The Theory and Practise of Reliable System Design. D.P. Siewiorek and R.S. Schwarz, Digital Press.

B.54 Safety Bag (referenced by clause 9)

Aim

To protect against residual specification and implementation faults in software which adversely affect safety.

Description

A safety bag is an external monitor, implemented on an independent computer to a different specification. This safety bag is solely concerned with ensuring that the main computer performs safe, not necessarily correct, actions. The safety bag continuously monitors the main computer. The safety bag prevents the system from entering an unsafe state. In addition, if it detects that the main computer is entering a potentially hazardous state, the system has to be brought back to a safe state either by the safety bag or the main computer.

Reference:

Using AI Techniques to Improve Software Safety. Proc. IFAC SAFECOMP 88, Sarlat, France, Oct. 1986, Pergamon Press 1986.

B.55 Sneak Circuit Analysis (referenced by dt8)

Aim

To detect an unexpected path or logic flow within a system which, under certain conditions initiates an undesired function or inhibits a desired function.

Description

A sneak circuit path may consist of hardware, software, operator actions, or combinations of these elements. Sneak circuits are not the result of hardware failure but are latent conditions inadvertently designed into the system or coded into the software programs, which can cause it to malfunction under certain conditions.

Categories of sneak circuits are:

– Sneak Paths which cause current, energy, or logical sequence to flow along an unexpected path or in an unintended direction;

– Sneak Timing in which events occur in an unexpected or conflicting sequence;

– Sneak Indications which cause an ambiguous or false display of system operating conditions, and thus may result in an undesired action by the operator;

– des labels insidieux qui annotent de façon incorrecte ou imprécise les fonctions du système, par exemple les entrées système, les commandes, les affichages, les bus, etc. et qui risquent donc d'induire en erreur un opérateur susceptible par là même d'appliquer un stimulus incorrect au système.

L'analyse des chemins insidieux s'appuie sur l'identification de structures topologiques de base dans le matériel ou le logiciel (par exemple, six structures de base sont identifiées pour le logiciel). L'analyse a lieu grâce à une liste de contrôle comportant des questions sur l'utilisation et les relations entre les composants topologiques de base.

Références:

Sneak Analysis and Software Sneak Analysis. S.G. Godoy et G.J. Engels, J. Aircraft Vol. 15, No. 8, 1978.

Sneak Circuit Analysis. J.P. Rankin, Nuclear Safety, Vol. 14, No. 5, 1973.

B.56 Gestion de la configuration du logiciel (référencé par l'article 15)

Objectif

La gestion de la configuration du logiciel vise à assurer la cohérence des livraisons des groupes de logiciel développés à mesure que ces livraisons évoluent. La gestion de la configuration s'applique, en général, tant au développement matériel que logiciel.

Description

La gestion de la configuration du logiciel est une technique utilisée tout au long du développement. Pour l'essentiel, elle exige l'enregistrement de la production de toute version de chaque livraison «importante» et de toute relation entre les différentes versions des différentes livraisons. Les enregistrements résultants permettent au développeur de déterminer l'effet, sur les autres livraisons, d'un changement apporté à une livraison (spécialement sur l'un de ses composants). En particulier, les systèmes ou sous-systèmes peuvent être reconstruits de façon fiable à partir d'ensembles cohérents de versions de composants.

Références:

Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs. MIL-STD-483.

Software Configuration Management. J.K. Buckle, Macmillan Press, 1982.

Software Configuration Management. W.A. Babich, Addison-Wesley, 1986.

Configuration Management Requirements for Defence Equipment. Ministère de la défense du Royaume-Uni, Standard 05-57 numéro 1, 1980.

B.57 Langages de programmation à fort typage (référencé par l'article 10)

Objectif

Réduire la probabilité de défauts en utilisant un langage autorisant un haut niveau de contrôle par le compilateur.

– Sneak Labels which incorrectly or imprecisely label system functions, for example, system inputs, controls, displays, buses, etc., and thus may mislead an operator into applying an incorrect stimulus to the system.

Sneak circuit analysis, relies on the recognition of basic topological patterns with the hardware or software structure (e.g. six basic patterns are identified for software). Analysis takes place with the aid of a checklist of questions about the use and relationships between the basic topological components.

References:

Sneak Analysis and Software Sneak Analysis. S.G. Godoy and G.J. Engels, J. Aircraft Vol. 15, No. 8, 1978.

Sneak Circuit Analysis. J.P. Rankin, Nuclear Safety, Vol. 14, No. 5, 1973.

B.56 Software Configuration Management (referenced by clause 15)

Aim

Software Configuration management aims to ensure the consistency of groups of development deliverables as those deliverables change. Configuration Management, in general, applies to both hardware and software development.

Description

Software Configuration Management is a technique used throughout development. In essence, it requires the recording of the production of every version of every "significant" deliverable and of every relationship between different versions of the different deliverables. The resulting records allow the developer to determine the effect on other deliverables of a change to one deliverable (especially on of its components). In particular, systems or subsystems can be reliably re-built from consistent sets of component versions.

References:

Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs. MIL-STD-483.

Software Configuration Management. J.K. Buckle, Macmillan Press, 1982.

Software Configuration Management. W.A. Babich, Addison-Wesley, 1986.

Configuration Management Requirements for Defence Equipment. UK Ministry of Defence Standard 05-57 Issue 1, 1980.

B.57 Strongly Typed Programming Languages (referenced by clause 10)

Aim

Reduce the probability of faults by using a language which permits a high level of checking by the compiler.

Description

De tels langages, en général, autorisent que des types de données définis par l'utilisateur soient définis à partir des types de données du langage de base (ENTIER, REEL, par exemple). Ces types peuvent alors être utilisés exactement de la même façon que les types de base, mais des contrôles stricts sont imposés pour assurer que le type correct est utilisé. Ces contrôles sont imposés sur l'ensemble du programme, même si celui-ci est construit à partir d'unités compilées distinctes. Les contrôles assurent aussi que le nombre et le type des arguments des procédures correspondent même lorsqu'ils sont référencés à partir de modules compilés distincts.

Les langages de programmation à fort typage supportent aussi d'autres aspects de la bonne pratique du génie logiciel tels que des structures de contrôle facilement analysables (par exemple, IF .. THEN .. ELSE, DO .. WHILE, etc.) qui conduisent à des programmes bien structurés.

Des exemples types de langage à fort typage sont Pascal, Ada et Modula 2.

Références:

Reference Manual for the Ada Programming Language, ANSI/MIL-STD-815A 1983.

In search of Effective Diversity: a Six Language Study of Fault-Tolerant Flight Control Software, A. Avizienis, M. R. Lyu, W. Schutz, The Eighteenth International Symposium on Fault Tolerant Computing, Tokyo, Japon 27-30 juin 1988, IEEE Computer Society Press, ISBN 0-8186-0867-6.

B.58 Tests structurels (référencé par td2)

Objectif

Effectuer des tests permettant de tester certains sous-ensembles de la structure du programme.

Description

Basé sur une analyse du programme, un ensemble de données d'entrée est choisi de manière à ce qu'une grande fraction des éléments du programme sélectionnés soit testée. Les éléments testés du programme peuvent varier en fonction du niveau de rigueur requis.

Instruction: il s'agit du test le moins rigoureux puisqu'il est possible d'exécuter toutes instructions du code sans tester les deux branches d'une instruction conditionnelle.

Branches: il convient de contrôler les deux côtés de chaque branche. Il n'est pas exclu que cette vérification soit peu pratique pour certains types de codes défensifs.

Conditions composées: chaque condition d'une branche conditionnelle composée (c'est-à-dire reliée par AND/OR est testée).

LCSAJ (linear code sequence and jump): une section linéaire de code et saut est une quelconque section linéaire de code comprenant des branchements conditionnels se terminant par un branchement. Beaucoup de sous-chemins potentiels seront irréalisables en raison des contraintes sur les données d'entrée imposées par l'exécution du code précédent.

Flux de données: les chemins d'exécution sont sélectionnés sur la base de l'usage des données, par exemple un chemin dans lequel la même variable est lue et écrite à la fois.

Description

Such languages usually allow user-defined data types to be defined from the basic language data types (such as INTEGER, REAL). These types can then be used in exactly the same way as the basic types, but strict checks are imposed to ensure the correct type is used. These checks are imposed over the whole program, even if this is built from separately compiled units. The checks also ensure that the number and the type of procedure arguments match even when referenced from separately compiled modules.

Strongly typed languages also support other aspects of good software engineering practice such as easily analysable control structures (e.g. IF .. THEN .. ELSE, DO .. WHILE, etc.) which lead to well-structured programs.

Typical examples of strongly typed languages are Pascal, Ada and Modula 2.

References:

Reference Manual for the Ada Programming Language, ANSI/MIL-STD-815A 1983.

In search of Effective Diversity: a Six Language Study of Fault-Tolerant Flight Control Software, A. Avizienis, M.R. Lyu, W. Schutz, The Eighteenth International Symposium on Fault Tolerant Computing, Tokyo, Japan 27-30 June 1988, IEEE Computer Society Press, ISBN 0-8186-0867-6.

B.58 Structure Based Testing (referenced by dt2)

Aim

To apply tests which exercise certain subsets of the program structure.

Description

Based on an analysis of the program a set of input data is chosen in such a way that a large fraction of selected program elements are exercised. The program elements exercised can vary depending upon the level of rigour required.

Statements: this is the least rigorous test since it is possible to execute all code statements without exercising both branches of a conditional statement.

Branches: both sides of every branch should be checked. This may be impractical for some types of defensive code.

Compound Conditions: every condition in a compound conditional branch (i.e. linked by AND/OR is exercised).

LCSAJ: a linear code sequence and jump is any linear sequence of code statements including conditional jumps terminated by a jump. Many potential sub-paths will be infeasible due to constraints on the input data imposed by the execution of earlier code.

Data Flow: the execution paths are selected on the basis of data usage, for example a path where the same variable is both written and wrote.

Graphe d'appel: un programme est composé de sous-routines qui sont éventuellement appelées à partir d'autres sous-routines. Le graphe d'appel est l'arborescence des appels des sous-routines du programme. Les essais sont destinés à couvrir tous les appels dans l'arborescence.

Chemin entier: exécute tous les chemins possibles du code. Des tests complets sont normalement irréalisables en raison du très grand nombre de chemins potentiels.

Référence:

Reliability of the Path Analysis Testing Strategy. W. Howden, IEEE Trans. Software Engineering, Vol. SE 3, 1976.

B.59 Schémas de structure (référencé par td5)

Objectif

Représenter la structure d'un programme sous forme de schéma.

Description

Les schémas de structure sont une notation venant compléter les organigrammes de données. Ils décrivent le système de programmation et la hiérarchie des parties qu'ils représentent sous forme d'un schéma arborescent. Ils documentent la manière dont les éléments des organigrammes de données peuvent être implémentés sous forme d'une hiérarchie d'unités de programmation.

Un schéma de structure décrit les relations entre les unités du programme sans inclure une quelconque information sur l'ordre d'activation de ces unités. Il est dessiné à l'aide des trois symboles suivants:

i) un rectangle annoté avec le nom de l'unité;

ii) une flèche reliant ces rectangles;

iii) une flèche entourée d'un cercle, annotée du nom des données entrant et sortant des éléments du schéma de structure. En principe, la flèche entourée d'un cercle est dessinée parallèlement à la flèche reliant les rectangles du tableau.

A partir de n'importe quel organigramme de données non ordinaire, il est possible de dériver un nombre de schémas de structures différentes.

Les schémas de structure dérivés des organigrammes de données représentent une structure de premier niveau du système dans laquelle chaque boîte du schéma correspond à une bulle de l'organigramme de données. Naturellement, des niveaux plus profonds peuvent être décrits en utilisant la même technique.

Références:

Software Engineering. I. Sommerville, Addison-Wesley 1982. ISBN 0-201-13795-X.

Structured Design. L.L. Constantine et E. Yourdon, Englewood Cliffs, New Jersey, Prentice-Hall, 1979.

Reliable Software Through Composite Design. G.J. Myers, New York, Van Nostrand, 1975.

Call Graph: a program is composed of subroutines which may be invoked from other subroutines. The call graph is the tree of subroutine invocations in the program. Tests are designed to cover all invocations in the tree.

Entire Path: execute all possible paths through the code. Complete testing is normally infeasible due to the very large number of potential paths.

Reference:

Reliability of the Path Analysis Testing Strategy. W. Howden, IEEE Trans. Software Engineering, Vol. SE 3, 1976.

B.59 Structure Diagrams (referenced by dt5)

Aim

To show the structure of a program diagrammatically.

Description

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

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

i) a rectangle annotated with the name of the unit;

ii) an arrow connecting these rectangles;

iii) a circled arrow, annotated with the name of data passed to and from elements in the structure chart. Normally, the circled arrow is drawn parallel to the arrow connecting the rectangles in the chart.

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

Structure charts derived from data flow diagrams represent a first level structure of the system, where each box on the structure chart represents a bubble in the data flow diagram. Naturally, deeper levels can be described using the same technique.

References:

Software Engineering. I. Sommerville, Addison-Wesley 1982. ISBN 0-201-13795-X.

Structured Design. L.L. Constantine and E. Yourdon, Englewood Cliffs, New Jersey, Prentice-Hall, 1979.

Reliable Software Through Composite Design. G.J. Myers, New York, Van Nostrand, 1975.

B.60 Méthode structurée (référencé par les articles 8 et 10)

Objectif

Le principal objectif des méthodologies structurées est de promouvoir la qualité du développement logiciel en portant l'attention sur les premières parties du cycle de vie. Les méthodes visent à accomplir cela à l'aide de procédures intuitives et précises et de notations (assistées par ordinateurs) permettant d'identifier l'existence d'exigences et de caractéristiques d'implémentation dans un ordre logique et d'une manière structurée.

Description

Il existe toute une gamme de méthodes structurées. Certaines parmi lesquelles les méthodes SSADM et LBMS sont conçues pour le traitement de données traditionnel et les fonctions de traitement transactionnel, tandis que d'autres (MASCOT, JSD, Yourdon en temps réel) sont plus orientées vers le contrôle des processus et les applications en temps réel (qui tendent plus à être à critique de sécurité).

Les méthodes structurées sont essentiellement des «supports de réflexion» permettant de percevoir et de fractionner systématiquement un problème ou un système. Leurs principales caractéristiques sont les suivantes:

– ordre logique de réflexion décomposant un vaste problème en étapes faciles à manier;
– identification du système total, y compris l'environnement ainsi que le système prescrit;
– décomposition des données et fonctions dans le système prescrit;
– listes de contrôle, autrement dit liste des types d'éléments exigeant une définition;
– faible investissement intellectuel – Simples, intuitives, pragmatiques.

Les notations correspondantes ont tendance à être précises dans l'identification d'un problème et des entités du système (par exemple, les processus et les flots de données), mais les fonctions de traitement exécutées par ces entités ont tendance à être exprimées par des notations informelles. Certaines méthodes utilisent toutefois en partie des notations (mathématiquement) formelles (JSD, par exemple, emploie des expressions régulières; Yourdon, SOM et SDL utilisent des automates à état finis). Cette précision non seulement réduit la possibilité de méprise, mais elle offre la possibilité du traitement automatique.

La visibilité est un autre avantage de la notation structurée, car elle permet à l'utilisateur de contrôler intuitivement la spécification ou la conception d'après sa connaissance élevée mais non formalisée.

La présente bibliographie décrit certaines de ces méthodologies structurées de façon plus détaillée, notamment, la méthodologie CORE (Controlled Requirements Expression), JSD (Jackson System Development), MASCOT, Yourdon en temps réel et SADT (Structured Analysis and Design Technique).

Références:

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

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

The Use of Structured Methods in the Development of Large Software-Based Avionic Systems. D.J. Hatley, Proceedings DASC, Baltimore, 1984.

B.60 Structured Methodology (referenced by clauses 8 and 10)

Aim

The main aim of Structured Methodologies is to promote the quality of software development by focusing attention on the early parts of the life cycle. The methods aim to achieve this through both precise and intuitive procedures and notations (assisted by computers) to identify the existence of requirements and implementation features in a logical order and a structured manner.

Description

A range of Structured Methodologies exist. Some such as SSADM, LBMS are designed for traditional data-processing and transaction processing functions, while others (MASCOT, JSD, real-time Yourdon) are more oriented to process-control and real-time applications (which tend to be more safety-critical).

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

– a logical order of thought, breaking a large problem into manageable stages;
– identification of total system, including the environment as well as the required system;
– decomposition of data and function in the required system;
– checklists, i.e. lists of the sort of things that need definition;
– low intellectual overhead – simple, intuitive, pragmatic.

The supporting notations tend to be precise for identifying problem and system entities (e.g. processes and data flows), but the processing functions performed by these entities tend to be expressed using informal notations. However some methods do make partial use of (mathematically) formal notations (for example, JSD makes use of regular expressions: Yourdon, SOM and SDL utilise finite state machines). This precision not only reduces the scope for misunderstanding, it provides scope for automatic processing.

Another benefit of structured notation is its visibility, which enables a specification or design to be checked intuitively by a user, against his powerful but unstated knowledge.

This bibliography describes some of these Structured Methodologies in more detail; for instance, Controlled Requirements Expression, Jackson System Development, MASCOT, real-time Yourdon and Structured Analysis and Design Technique (SADT).

References:

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

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

The Use of Structured Methods in the Development of Large Software-Based Avionic Systems. D.J. Hatley, Proceedings DASC, Baltimore, 1984.

B.60.1 Méthode CORE (Controlled Requirements Expression)

Objectif

Assurer que toutes les exigences sont identifiées et exprimées.

Description

Cette approche vise à établir un rapprochement entre le client/utilisateur final et l'analyste. Elle n'a pas la rigueur mathématique mais elle facilite le processus de communication – CORE est plus adaptée à l'expression des exigences qu'à la spécification. Cette approche est structurée et l'expression passe par différents niveaux de raffinement. Le processus CORE favorise une vue plus large du problème en apportant une connaissance de l'environnement dans lequel le système sera utilisé ainsi que les points de vue divers des différents types d'utilisateurs. CORE inclut des lignes directrices et des tactiques permettant de trouver des points pistes de réflexions à partir de la «vision générale». Ces pistes peuvent être corrigées ou identifiées explicitement et documentées. Ainsi, il n'est pas exclu que les spécifications ne soient pas complètes, mais les problèmes non résolus et les zones à haut risque sont identifiés et doivent être abordés dans la conception ultérieure.

B.60.2 JSD (Jackson System Development)

Objectif

Méthode de développement recouvrant le développement des systèmes logiciels depuis le besoin jusqu'au code, mettant particulièrement l'accent sur les systèmes en temps réel.

Description

La méthode JSD est un processus de développement par étapes dans lequel le développeur modélise les processus de la situation réelle sur lesquels doivent reposer les fonctions du système, détermine les fonctions requises et les insère dans le modèle, et transforme la spécification résultant en une spécification réalisable dans l'environnement cible. La méthode recouvre donc les phases classiques de définition, de conception et d'implémentation mais adopte une optique quelque peu différente par rapport aux méthodes traditionnelles en ce sens qu'elle n'est pas descendante.

Cette méthode met, en outre, largement l'accent sur le stade précoce d'identification des entités du monde extérieur en rapport avec le système en cours de construction et sur leur modélisation et sur ce qui peut leur arriver. Une fois que cette analyse du «monde extérieur» a été faite et qu'un modèle a été créé, les fonctions requises du système sont analysées afin de déterminer comment elles peuvent s'intégrer dans ce modèle. Le modèle de système résultant est enrichi des descriptions structurées de tous les processus du modèle et l'ensemble est alors transformé en programmes qui fonctionneront dans l'environnement logiciel et matériel cible.

Références:

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

System Development. M. Jackson, Prentice-Hall, 1983.

B.60.3 Méthodologie MASCOT

Objectif

Conception et implémentation de systèmes en temps réel.

B.60.1 Controlled Requirements Expression (CORE)

Aim

To ensure that all the requirements are identified and expressed.

Description

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

B.60.2 JSD – Jackson System Development

Aim

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

Description

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

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

References:

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

System Development. M. Jackson, Prentice-Hall, 1983.

B.60.3 MASCOT

Aim

The design and implementation of real-time systems.

Description

La méthode MASCOT (Modular Approach to Software Construction, Operation and Test) est une méthode de conception supportée par un système de programmation. Il s'agit d'une méthode systématique d'expression de la structure d'un système en temps réel, d'une façon indépendante du matériel cible ou du langage d'implémentation. Elle impose une approche disciplinée de la conception qui génère une structure hautement modulaire, en assurant une correspondance étroite entre les éléments fonctionnels de la conception et les éléments de construction apparaissant dans l'intégration du système. Un système est conçu en terme d'un réseau de processus parallèles qui communiquent par l'intermédiaire de canaux. Les canaux peuvent être des regroupements de données fixes ou des files d'attente (pipelines de données). Le contrôle d'accès aux canaux est défini indépendamment des processus, il est définit en termes de mécanismes d'accès qui appliquent aussi les règles de planification aux processus. Des versions récentes de MASCOT ont été imaginées en gardant l'implémentation Ada à l'esprit.

MASCOT supporte une stratégie d'acceptation qui repose sur le test et la vérification des modules simples et de plus grands ensembles de modules fonctionnellement associés. Une implémentation MASCOT est prévue pour être construite sur un noyau MASCOT – un ensemble de primitifs de planification qui souligne l'implémentation et supporte les mécanismes d'accès.

Référence:

MASCOT 3 User Guide. MASCOT Users Forum, RSRE, Malvern, Angleterre, 1987.

B.60.4 Yourdon en temps réel

Objectif

Cette méthode vise à représenter une méthode complète de développement logiciel composée de techniques de spécification et de conception orientées vers le développement des systèmes en temps réel.

Description

Le plan de développement sous-tendant cette technique suppose une évolution en trois phases du système en cours de développement. La première phase implique la construction d'un «modèle essentiel» qui décrit le comportement requis par le système. La deuxième implique la construction d'un modèle d'implémentation qui décrit les structures et les mécanismes qui, lorsqu'ils sont implémentés, incarnent le comportement requis. La troisième phase implique la construction réelle du système tant au niveau matériel que logiciel. Les trois phases correspondent à peu près aux phases classiques de définition, de conception et d'implémentation mais l'accent est plus largement mis sur le fait qu'à chaque phase, le développeur est engagé dans une activité de modélisation.

Le modèle essentiel se compose de deux parties: le modèle environnemental contenant une définition de la frontière entre le système et son environnement et une description des événements externes auxquels le système doit répondre, et le modèle de comportement qui contient les schémas définissant la transformation que le système réalise en réponse aux événements et une définition des données que le système doit contenir pour pouvoir répondre.

Le modèle d'implémentation se divise aussi en sous-modèles: dans ce cas, décrivant l'allocation des processus individuels aux processeurs et la décomposition des processus en modules.

Pour produire ces modèles, la technique combine un nombre de techniques bien connues: les diagramme de flux de données, le langage naturel structuré, les schémas de transition d'état et les réseaux de Pétri.

Description

MASCOT (Modular Approach to Software Construction, Operation and Test) is a design method supported by a programming system. It is a systematic method of expressing the structure of real-time system in a way that is independent of the target hardware or implementation language. It imposes a disciplined approach to design that yields a highly modular structure, ensuring a close correspondence between the functional elements in the design and the construction elements appearing in system integration. A system is designed in terms of a network of concurrent processes that communicate through channels. Channels can be either pools of fixed data or queues (pipelines of data). Control of access to channels is defined independently of the processes. It is defined in terms of access mechanisms that also enforce scheduling rules on the processes. Recent versions of MASCOT have been designed with Ada implementation in mind.

MASCOT supports an acceptance strategy based on the test and verification of single modules and larger collections of functionally related modules. A MASCOT implementation is intended to be built upon a MASCOT kernel – a set of scheduling primitives that underline the implementation and support the access mechanisms.

Reference:

MASCOT 3 User Guide. MASCOT Users Forum, RSRE, Malvern, England, 1987.

B.60.4 Real-time Yourdon

Aim

This aims at being a complete software development method consisting of specification and design techniques oriented towards the development of real-time systems.

Description

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

The essential model is in two parts: the environmental model containing a definition of the boundary between the system and its environment and a description of the external events to which the system must respond, and the behavioural model which contains schemas defining the transformation the system carries out in response to events and a definition of the data the system must hold in order to respond.

The implementation model also divides into sub-models: in this case covering the allocation of individual processes to processors and of the decomposition of the processes into modules.

To capture these models the technique combines a number of well-known techniques: data-flow diagrams, structured English, state transition diagrams and Petri Nets.

En complément, la méthode intègre des techniques de simulation d'une conception de système proposée soit sur le papier soit mécaniquement à partir de modèles qui sont dessinés.

Référence:

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

B.60.5 Méthodologie SADT (Analyse Structurée et Technique de Conception)

Objectif

Modéliser et identifier, sous forme de schémas utilisant des flux d'informations, les processus de prise de décision et les tâches de gestion associées à un système complexe.

Description

Dans la méthodologie SADT, le concept de l'actigramme joue un rôle primordial. Un actigramme se compose d'activités regroupées dans ce que l'on appelle des «boîtes d'action». Chaque boîte, dotée d'un nom unique, est reliée à d'autres boîtes par des relations de facteurs (représentées par des flèches) auxquelles il est également attribué un nom unique. Chaque boîte peut se décomposer hiérarchiquement en boîtes et en relations subsidiaires. Il existe quatre types de facteurs: les entrées, les commandes, les mécanismes et les sorties.

– Entrée: indiquée par une flèche qui pénètre à gauche dans une boîte d'action. Les entrées peuvent représenter des choses matérielles ou immatérielles et elles sont facilement manipulables par une ou par plusieurs activités d'une boîte d'action.
– Commandes: ce sont, en principe, des instructions, des procédures, des critères de choix, etc. Les commandes guident l'exécution d'une activité. Elles sont décrites par des flèches pénétrant dans une boîte d'action par le haut.
– Mécanisme: il s'agit d'une ressource telle que le personnel, les unités organisationnelles ou le matériel, indispensable à une activité pour exécuter sa tâche.
– Sortie: elle indique toute chose produite par une activité et est représentée par une flèche sortant de la boîte d'action du côté droit.

Quand les activités sont fortement liées les unes aux autres par de nombreuses relations de facteurs, il est peut-être préférable de considérer ces activités comme un groupe indivisible contenu dans une seule boîte d'action ne se prêtant pas elle-même à un détail plus approfondi de son contenu. Le principe directeur du regroupement des activités dans les boîtes d'action est que les boîtes résultantes sont reliées deux à deux par peu de facteurs.

La hiérarchie du modèle des actigrammes continue jusqu'à ce qu'il devienne inutile de détailler plus avant les boîtes d'action. On atteint cette étape quand les activités contenues dans les boîtes sont inséparables ou quand un plus grand niveau de détail des boîtes d'action s'inscrit hors du champ d'application de l'analyse du système.

Références:

Structured Analysis for Requirements Definition, D.T. Ross, K.E. Schoman Jr., IEEE Trans. Software Eng., Vol. SE-3, 1977, 6-15.

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

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

Structured Analysis and Design Technique – Application on Safety Systems, W. Heins, Risk Assessment and Control Courseware, Module B1, chapitre 11. 1989, Delft University of Technology, Safety Science Group, PO Box 5050, 2600 GB Delft, Pays-Bas.

Additionally the method contains techniques for simulating a proposed system design either on paper or mechanically from the models that are drawn up.

Reference:

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

B.60.5 SADT – Structured Analysis and Design Technique

Aim

To model and identify, in a diagrammatic form using information flows, the decision-making processes and the management tasks associated with a complex system.

Description

In SADT, the concept of an Activity-Factor Diagram plays a central role. An A/F diagram consists of activities grouped in so called 'action boxes'. Each action box has a unique name, and is linked to other action boxes by factor relations (drawn as arrows) which are also given unique names. Each action box can be hierarchically decomposed into subsidiary action boxes and relations. There are four types of factors: inputs, controls, mechanisms and outputs.

- Input: indicated by an arrow that enters an action box at the left-hand side. Inputs can represent material or immaterial things and they are suitable for manipulation by one or more activities in an action box.
- Control: are typically instructions, procedures, choice criteria and so on. Controls guide the execution of an activity and they are shown by arrows entering the top side of an action box.
- Mechanism: is a resource such as personnel, organisational units or equipment, that is needed for an activity to perform its task.
- Output: can denote anything that an activity produces, and it is pictured by an arrow leaving an action box at the right-hand side.

When activities are strongly related to each other by many factor relations then it is perhaps better to consider these activities as an indivisible group that is contained in one action box which does not lend itself to further detailing of its content. The guiding principle for grouping of activities into action boxes is that the resulting boxes are coupled pairwise by only a few factors.

The model hierarchy of A/F diagrams is pursued until a further detailing of the action boxes is meaningless. This stage is reached when the activities within the boxes are inseparable or when further detailing of the action boxes falls outside the scope of the system analysis.

References:

Structured Analysis for Requirements Definition, D.T. Ross, K.E. Schoman Jr., IEEE Trans. Software Eng., Vol. SE-3, 1977, 6-15.

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

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

Structured Analysis and Design Technique – Application on Safety Systems, W. Heins, Risk Assessment and Control Courseware, Module B1, chapter 11. 1989, Delft University of Technology, Safety Science Group, PO Box 5050, 2600 GB Delft, Netherlands.

B.61 Programmation structurée (référencé par l'article 10)

Objectif

Concevoir et implémenter le programme de manière à qu'il soit pratique à analyser. Il convient que cette analyse permette de découvrir tous les comportements significatifs du programme.

Description

Il convient que le programme soit caractérisé par une complexité structurelle minimale et qu'il évite les branchements complexes. Il est recommandé (lorsque c'est possible) d'associer simplement les contraintes et conditions de boucles aux paramètres d'entrée. Il convient de diviser le programme en modules de taille appropriée et que l'interaction de ces modules soit explicite. Il convient que les caractéristiques du langage de programmation qui favorisent l'approche ci-dessus soient utilisées de préférence à d'autres caractéristiques supposées être plus efficaces, excepté lorsque l'efficacité constitue une priorité absolue (par exemple, certains systèmes critiques de sécurité).

Références:

A Discipline of Programming. E.W. Dijkstra, Englewood Cliffs NJ, Prentice-Hall, 1976.

Assessing a Class of Software Tools. M.A. Hennell *et al.*, 7th International conference on Software Engineering, mars 1984, Orlando.

A Software Tool for Top-down Programming. D.C. Ince, Software – Practice and Experience, Vol. 13, No. 8, août 1983.

B.62 Langages de programmations adaptés (référencé par td4)

Objectif

Soutenir, autant que possible, les exigences de la présente Norme internationale, notamment la programmation défensive, le fort typage, la programmation structurée et les assertions. Il convient que le langage de programmation choisi conduise à un code facilement vérifiable avec un minimum d'effort et facilite le développement, la vérification et la maintenance du programme.

Description

Il convient que le langage soit défini intégralement et sans ambiguïté. Il est recommandé qu'il soit orienté utilisateur ou problème plutôt que machine. Les langages largement utilisés ou leurs sous-ensembles sont préférés aux langages spécialisés.

Outre les caractéristiques déjà référencées, il convient que le langage propose

– une structure de bloc,
– une vérification du temps de compilation,
– une vérification à l'exécution des types et des limites des tableaux, et
– une vérification des paramètres.

Il convient que le langage favorise

– l'utilisation de petits modules faciles à gérer,
– la limitation de l'accès aux données dans les modules définis,
– la définition des sous-domaines des variables, et
– tout autre type de structure limitant les erreurs.

B.61 Structured Programming (referenced by clause 10)

Aim

To design and implement the program in a way which makes practical the analysis of the program. This analysis should be capable of discovering all significant program behaviour.

Description

The program should contain the minimum of structural complexity. Complicated branching should be avoided. Loop constraints and branching should (where possible) be simply related to input parameters. The program should be divided into appropriately small modules, and the interaction of these modules should be explicit. Features of the programming language which encourage the above approach should be used in preference to other features which are (allegedly) more efficient, except where efficiency takes absolute priority (e.g. some safety-critical systems).

References:

A Discipline of Programming. E.W. Dijkstra, Englewood Cliffs NJ, Prentice-Hall, 1976.

Assessing a Class of Software Tools. M.A. Hennell *et al.*, 7th International conference on Software Engineering, March 1984, Orlando.

A Software Tool for Top-down Programming. D.C. Ince, Software – Practice and Experience, Vol. 13, No. 8, August 1983.

B.62 Suitable Programming Languages (referenced by dt4)

Aim

To support the requirements of this International Standard as much as possible, in particular, defensive programming, strong typing, structured programming and possibly assertions. The programming language chosen should lead to easily verifiable code with a minimum of effort and facilitate program development, verification and maintenance.

Description

The language should be fully and unambiguously defined. The language should be user- or problem-oriented rather than machine-oriented. Widely used languages or their subsets are preferred to special purpose languages.

In addition to the already referenced features the language should provide for

– block structure,
– translation time checking,
– run time type and array bound checking, and
– parameter checking.

The language should encourage

– the use of small and manageable modules,
– restriction of access to data in defined modules,
– definition of variable sub-ranges, and
– any other type of error-limiting constructs.

Il est souhaitable que le langage soit supporté par un compilateur adapté, des bibliothèques appropriées de modules préexistants, un outil de mise au point et des outils adaptés au contrôle de version et au développement.

Les caractéristiques qui rendent la vérification difficile et qu'il convient donc d'éviter sont les suivantes:

– les branchements inconditionnels à l'exception des appels de sous-routines;
– la récursivité;
– pointeurs, collections ou tout autre type d'objets ou de variables dynamiques;
– la gestion des interruptions au niveau du code source;
– les entrées ou sorties multiples des boucles, blocs ou sous-programmes;
– l'initialisation ou la déclaration implicite de variables;
– les enregistrements variables et l'équivalence;
– les paramètres procéduraux.

Les langages de bas niveau, notamment les langages assembleurs, posent des problèmes en raison de leur nature orientée machine.

B.63 Exécution symbolique (référencé par td8)

Objectif

Montrer l'accord entre le code source et la spécification.

Description

Le programme est exécuté en remplaçant le côté gauche par le côté droit dans toute assignation. Les branchements conditionnels et les boucles sont traduits en expressions booléennes. Le résultat final est une expression symbolique de chaque variable du programme. Ce résultat peut être vérifié en le comparant à l'expression attendue.

Références:

Formal Program Verification using Symbolic Execution. R.B. Dannenberg et G.W. Ernst, IEEE Trans. Software Engineering, Vol. SE-8, No. 1, 1982.

Symbolic Execution and Software Testing. J.C. King, Comm. ACM, Vol. 19, No. 7, 1976.

B.64 Réseaux de Pétri temporels (référencé par td5 et td7)

Objectif

Modéliser les aspects pertinents du comportement du système et évaluer et améliorer, le cas échéant, les exigences en matière de sécurité et de fonctionnement par le biais de l'analyse et de la reconception.

Description

Les réseaux de Pétri appartiennent à une catégorie de modèles théoriques de graphes, adaptés à la représentation de l'information et du flux de contrôle dans les systèmes présentant des caractéristiques de parallélisme et de comportement asynchrone.

It is desirable that the language is supported by a suitable translator, appropriate libraries of pre-existing modules, a debugger and tools for both version control and development.

Features which make verification difficult and therefore should be avoided are:

– unconditional jumps excluding subroutine calls;
– recursion;
– pointers, heaps or any type of dynamic variables or objects;
– interrupt handling at source code level;
– multiple entries or exits of loops, blocks or subprograms;
– implicit variable initialisation or declaration;
– variant records and equivalence; and
– procedural parameters.

Low-level languages, in particular, assembly languages, present problems due to their machine oriented nature.

B.63 Symbolic Execution (referenced by dt8)

Aim

To show the agreement between the source code and the specification.

Description

The program is executed substituting the left-hand side by the right-hand side in all assignments. Conditional branches and loops are translated into Boolean expressions. The final result is a symbolic expression for each program variable. This can be checked against the expected expression.

References:

Formal Program Verification using Symbolic Execution. R.B. Dannenberg and G.W. Ernst, IEEE Trans. Software Engineering, Vol. SE-8, No 1, 1982.

Symbolic Execution and Software Testing. J.C. King, Comm. ACM, Vol. 19, No. 7, 1976.

B.64 Time Petri Nets (referenced by dt5 and dt7)

Aim

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

Description

Petri nets belong to a class of graph theoretic models which are suitable for representing information and control flow in systems exhibiting concurrency and asynchronous behaviour.

Un réseau de Pétri est un réseau de places et de transitions. Les places sont «marquées» ou «non marquées». Une transition est «activée» quand toutes ses places d'entrée sont marquées. Quand elle est activée, elle est autorisée (mais pas obligée) à «tirer». Dans ce cas, les marques d'entrée sont retirées et chaque place de sortie de la transition est marquée .

Les dangers potentiels sont représentés sous forme d'états particuliers (marquages) dans le modèle. Les réseaux de Pétri étendus autorisent la modélisation des caractéristiques temporelles du système. S'il est vrai que les réseaux de Pétri «classiques» se concentrent sur les aspects du flux de contrôle, plusieurs extensions ont été proposées en vue d'incorporer le flot de données dans le modèle.

Références:

Net Theory and Applications. W. Brauer (ed), Lecture Notes in Computer Science, Vol. 84, Springer 1980.

Petri Net Theory and Modelling of Systems. J.L. Peterson, Prentice-Hall, 1981.

Safety Analysis using Petri Nets. N. Leveson et J. Stolzy, Proc. FTCS 15, Ann Arbor, Michigan, juin 1985, IEEE 1985.

A Tool for Requirements Specification and Analysis of Real Time Software Based on Timed Petri Nets. S. Bologna, F. Pisacane, C. Ghezzi, D. Mandrioli, Proc. SAFECOMP 88, 9-11 nov. 1988. Fulda Rép. Féd. d'Allemagne 1988.

B.65 Compilateur éprouvé à l'utilisation (référencé par l'article 10)

Objectif

Eviter toute difficulté provoquée par des défaillances du compilateur pouvant survenir au cours du développement, de la vérification et de la maintenance du logiciel.

Description

On utilise un compilateur dont le fonctionnement correct a déjà été démontré dans de nombreux projets. Les compilateurs non éprouvés en exploitation ou comportant des erreurs graves connues sont interdits.

Si le traducteur a montré des petites déficiences, les structures de langage associées sont consignées et soigneusement évitées lors d'un projet lié à la sécurité.

Une autre version de ce mode de travail consiste à limiter l'usage du langage aux seules caractéristiques couramment utilisées.

Cette recommandation repose sur l'expérience acquise au cours de nombreux projets. Il a été montré que des traducteurs immatures constituent un sérieux handicap à tout développement logiciel. Ils rendent le développement logiciel lié à la sécurité généralement irréalisable.

Actuellement, nous savons aussi qu'il n'existe aucune méthode permettant de prouver la validité de toutes les parties d'un compilateur.

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

The potential hazards are represented as particular states (markings) in the model. Extended Petri nets allow timing features of the system to be modelled. Although 'classical' Petri nets concentrate on control flow aspects, several extensions have been proposed to incorporate dataflow into the model.

References:

Net Theory and Applications. W. Brauer (ed), Lecture Notes in Computer Science, Vol. 84, Springer 1980.

Petri Net Theory and Modelling of Systems. J.L. Peterson, Prentice-Hall, 1981.

Safety Analysis using Petri Nets. N. Leveson and J. Stolzy, Proc. FTCS 15, Ann Arbor, Michigan, June 1985, IEEE 1985.

A Tool for Requirements Specification and Analysis of Real Time Software Based on Timed Petri Nets. S. Bologna, F. Pisacane, C. Ghezzi, D. Mandrioli, Proc. SAFECOMP 88, 9-11 Nov. 1988. Fulda Fed. Rep. of Germany 1988.

B.65 Translator Proven In Use (referenced by clause 10)

Aim

To avoid any difficulties due to translator failures which can arise during development, verification and maintenance of a software package.

Description

A translator is used, whose correct performance has been demonstrated in many projects already. Translators without operating experience or with any serious known errors are prohibited.

If the translator has shown small deficiencies the related language constructs are noted down and carefully avoided during a safety-related project.

Another version of this way of working is to restrict the usage of the language to its commonly used features only.

This recommendation is based on experience from many projects. It has been shown that immature translators are a serious handicap to any software development. They make a safety-related software development generally infeasible.

It is also known, presently, that no method exits to prove the correctness for all compiler parts.

B.66 Revues/Examens de la conception (référencé par td8)

Objectif

Détecter les erreurs présentes dans un produit du processus de développement aussi tôt et aussi économiquement que possible.

Description

La CEI a publié la Norme internationale 61160 qui donne des recommandations pour la mise en oeuvre des procédures de revues de conception en tant que moyen de stimulation pour l'amélioration d'un produit ou d'un processus. Elle comprend des lignes directrices pour la planification et la conduite de revues de conception, ainsi que des détails spécifiques concernant les contributions apportées par les spécialistes fiabilité, maintenance, logistique de maintenance et disponibilité. Cette norme comprend également des paragraphes sur les contributions d'autres spécialistes oeuvrant dans des domaines tels que la qualité, l'environnement, la sécurité (liée au produit et à l'utilisateur), les facteurs humains et les aspects juridiques.

A l'acception courante du terme «produit», il est indiqué d'inclure d'autres aspects tels que matériel et logiciel, listes constituant un catalogue, spécifications décrivant une entité, emballage de transport, installation et assemblage, instructions et manuels d'utilisation, étiquettes et avertissements, maintenance, listes de pièces de rechange, garanties, brochures commerciales et publicitaires.

Une revue du code consiste en une équipe de revue sélectionnant un petit ensemble de scénarios de test sur papier, représentatifs d'entrées et des sorties prévues correspondantes pour le programme. Les données de test sont alors manuellement analysées au travers de la logique du programme.

Références:

The Art of Software Testing. G. Myers, Wiley et Sons, New York, 1979.

Norme internationale CEI 61160: *Revue de conception formalisée* (1992).

B.67 Logique floue (référencé par l'article 10)

Objectif

La logique floue est une discipline mathématique, fondée sur la théorie des ensembles flous qui autorise des degrés intermédiaires entre les valeurs «vrai» et «faux». C'est la généralisation d'une logique binaire qui propose un modèle pour le raisonnement approximatif. Elle permet l'incorporation de l'intelligence humaine dans les systèmes automatiques. En reconnaissant la difficulté à définir des frontières précises, la logique floue limite la précision requise et mène, en conséquence, à des solutions simples et de haut niveau qui sont faciles à contrôler.

Description

La partie essentielle d'une solution en logique floue est un ensemble de règles linguistiques (règles IF – THEN) dans lequel les antécédents et les concluants sont associés à des ensembles flous.

EXEMPLE

IF la vitesse est grande et la distance_d'arrêt est moyenne

THEN l'accélérateur est proche de zéro et le freinage est faible

B.66 Walk-throughs/Design Reviews (referenced by dt8)

Aim

To detect errors in some product of the development process as soon and as economically as possible.

Description

The IEC has published an International Standard 61160 which makes recommendations for the implementation of design review procedures as a means of stimulating product and for process improvement. It includes guidelines for planning and conducting design reviews and specific details concerning contributions by reliability, maintenance, maintenance support and availability specialists. This standard also includes subclauses on contributions by other specialists dealing with related subjects such as quality, environment, safety (product and user), human factors, and legal matters.

In addition to the common understanding of "product", it is indicated to include other aspects, e.g. hardware and software, catalogue listings, specifications describing the item, shipping package, installation and assembly, instructions and operating manuals, labels and warnings, maintenance, spare parts lists, warranties, sales brochures and advertising literature.

A code walk-through consists of a walk-through team selecting a small set of paper test cases, representative sets of inputs and corresponding expected outputs for the program. The test data is then manually traced through the logic of the program.

References:

The Art of Software Testing. G. Myers, Wiley and Sons, New York, 1979.

International Standard IEC 61160: *Formal design review* (1992).

B.67 Fuzzy Logic (referenced by clause 10)

Aim

Fuzzy Logic is a mathematical discipline, based on the fuzzy set theory that allows for degrees of truth and falseness. It is a generalisation of bi-level logic which provides a model for approximate reasoning. It enables the incorporation of human intelligence into automatic systems. By acknowledging the difficulty of defining precise boundaries, Fuzzy Logic reduces the required precision, and hence leads to high-level and simple solutions which are easy to control.

Description

The essential part of a Fuzzy Logic solution is a set of linguistic rules (IF – THEN rules) where the antecedents and the consequents are associated with fuzzy sets.

EXAMPLE

IF speed is fast and distance_to_stop is medium

THEN accelerator is near zero and brake is light

Dans cet exemple, «vitesse» est une variable linguistique caractérisée pour un ensemble flou «grande», qui peut être interprété comme «la vitesse est supérieure à 70 km/h environ». Si la vitesse est inférieure à 60 km/h, la valeur du membre «grande» est 0. Si la vitesse est supérieure à 80 km/h, la valeur du membre «grande» est 1. Si la vitesse se situe entre 60 et 80 km/h, la valeur du membre «grande» varie entre 0 et 1.

La logique de prise de décision repose sur les catégories mathématiques des opérateurs: les normes triangulaires et les co-normes triangulaires.

Les systèmes basés sur la logique floue diffèrent des autres systèmes experts par beaucoup d'aspects tels que: (1) le petit nombre de règles, (2) l'utilisation exclusive du chaînage avant, (3) le non-chaînage des inférences (toutes les règles sont exécutées en parallèle dans une seule itération), (4) les règles statistiquement définies, (5) la vitesse d'exécution due à la simplicité des solutions et (6) le déterminisme.

Au cours des dernières années, la logique floue a trouvé de nombreuses applications dans des domaines allant de la finance à l'ingénierie des séismes. En particulier, le contrôle flou a émergé en vue de contrôler les systèmes hautement non linéaires ou les systèmes dont la description mathématique est inconnue ou trop complexe pour être traitée de façon analytique, ou encore des systèmes dans lesquels les mesures disponibles sont de mauvaise qualité.

Parmi les applications remarquables de contrôle en logique floue se trouvent le contrôle des vols aériens et le contrôle des alimentations électriques et des réacteurs nucléaires. Plus récemment, le contrôle flou a été appliqué avec succès aux systèmes d'exploitation ferroviaire automatique.

Références:

Fuzzy Sets: Zadeh. Information and Control, 1965, vol. 8, p. 338-353

Fuzzy Logic in Control Systems: Fuzzy Logic Controller, Part I et II. Chuen Chien Lee. IEEE Transactions on systems, man, and cybernetics, vol. 20, No. 2, mars/avril 1990.

Industrial Applications of Fuzzy Control: M. Sugeno Ed., Amsterdam North Holland, 1985.

Automatic Train Operation System by Predictive Fuzzy Control: Yasunobu, Miyamoto, in Industrial Applications of Fuzzy Control, M. Sugeno Ed., Amsterdam North Holland, 1985.

An automatic operation method for control rods in BWR plants: Kinoshita, Fukusaki, Satoh, Miyake, Proc. Specialists Meeting on In-Core Instrumentation and Reactor Core Assessment. Cadarache, France 1988.

Use of rule-based system for process control. Bernard. IEEE Contr. Syst. Mag., Vol. 8, No. 5, p. 3-13, 1988.

B.68 Programmation orientée objet (référencé par l'article 10)

Objectif

Permettre un prototypage rapide, faciliter la réutilisation des composants logiciels, effectuer le masquage d'informations, réduire la vraisemblance d'erreurs pendant toute la durée du cycle de vie, réduire l'effort nécessaire pendant la phase de maintenance, décomposer des problèmes complexes en petits problèmes plus faciles à gérer, réduire les dépendances entre les composants, créer des applications plus facilement extensibles.

In this example, "speed" is a linguistic variable characterised by a fuzzy set "fast", which can be interpreted as "speed is above about 70 km/h". If speed is less than 60 km/h, the membership value of "fast" is 0. If speed is above 80 km/h, the memberships value of "fast" is 1. If speed is between 60 and 80 km/h, the membership value of "fast" varies between 0 and 1.

The decision making logic is based on mathematical classes of operators: the triangular norms and triangular co-norms.

Fuzzy Logic rule-based systems differ from other expert systems in many ways as: (1) the small number of rules, (2) the use of forward chaining only, (3) non-chaining of inferences (all rules are executed in parallel in one iteration), (4) the statistically defined rules, (5) the speed of execution due to simplicity of the solutions, and (6) the determinism.

During the past few years, Fuzzy Logic has found numerous applications in fields ranging from finance to earthquake engineering. In particular, fuzzy control has emerged to control highly non-linear systems, or systems which mathematical description is unknown or too complex to be treated analytically, or systems in which the available measurements are of poor quality.

Notable applications of Fuzzy Logic control include aircraft flight control, and power systems and nuclear reactor control. More recently, fuzzy control has been applied successfully to automatic train operation systems.

References:

Fuzzy Sets: Zadeh. Information and Control, 1965, vol. 8, p. 338-353

Fuzzy Logic in Control Systems: Fuzzy Logic Controller, Part I and II. Chuen Chien Lee. IEEE Transactions on systems, man, and cybernetics, vol. 20, No. 2, March/April 1990.

Industrial Applications of Fuzzy Control: M. Sugeno Ed., Amsterdam North Holland, 1985.

Automatic Train Operation System by Predictive Fuzzy Control: Yasunobu, Miyamoto, in Industrial Applications of Fuzzy Control, M. Sugeno Ed., Amsterdam North Holland, 1985.

An automatic operation method for control rods in BWR plants: Kinoshita, Fukusaki, Satoh, Miyake, Proc. Specialists Meeting on In-Core Instrumentation and Reactor Core Assessment. Cadarache, France 1988.

Use of rule-based system for process control. Bernard. IEEE Contr. Syst. Mag., Vol. 8, No. 5, p. 3-13, 1988.

B.68 Object Oriented Programming (referenced by clause 10)

Aim

To enable rapid prototyping, to more easily reuse existing software components, to achieve information hiding, to reduce the likelihood of errors during the whole life cycle, to reduce the necessary effort during the maintenance phase, to break down complex problems into more easily manageable small problems, to reduce the dependencies between software components, to create more easily extendible applications.

Description

La programmation orientée objet est essentiellement une nouvelle manière de penser les logiciels, reposant sur les abstractions qui existent en situation réelle plutôt que sur des abstractions de calcul. La programmation orientée objet organise le logiciel comme un ensemble d'objets qui incorporent à la fois la structure des données et le comportement. Cette façon de penser contraste avec la programmation classique dans laquelle la structure des données et le comportement ne sont que vaguement reliés.

Objet: un objet se compose d'une zone de données privées et d'un ensemble d'opérations – appelées méthodes – sur cet objet. Les méthodes peuvent être publiques ou privées. Aucun autre composant du logiciel n'est autorisé à lire ou à changer directement les données privées d'un objet. Tous les autres composants logiciels doivent utiliser les méthodes publiques sur cet objet pour lire ou écrire des données dans la zone des données privées d'un objet.

Classe d'objet: en spécifiant une classe d'objet (souvent sous forme d'une définition du type), l'instanciation de nombreux objets appartenant à la même classe est rendue possible, autrement dit toutes les instanciations comportent toutes, la zone des données privées et les méthodes définies dans la classe d'objet.

Héritage (multiple): une classe d'objet peut hériter de la zone de données privées et des méthodes une (ou plusieurs) superclasses (classes d'objet se plaçant à un niveau supérieur dans la hiérarchie des classes) et est autorisé à ajouter certaines données privées et certaines méthodes ou à modifier les implémentations des méthodes héritées. En utilisant l'héritage, il est possible de construire plusieurs arborescences des classes d'objets.

Polymorphisme: il n'est pas exclu que la même opération se comporte différemment sur différentes classes d'objets, par exemple, pour un objet terminal, l'opération d'écriture écrit des caractères sur ce terminal tandis qu'une opération d'écriture sur un fichier écrit des caractères dans ce fichier.

Références:

Object Oriented Software Construction, Bertrand Meyer, Angleterre: Prentice-Hall International, 1988.

Classification as a paradigm for computing, Peter Wegner, Technical Report CS-86-11, Brown University, mai 1986.

Learning language, Peter Wegner, Byte (McGraw-Hill publication), Mars 1989.

Tous les comptes rendus des conférences OOPSLA (Object-Oriented Programming Systems, Languages and Applications) et ECOOP (European Conference on Object-Oriented Programming).

B.69 Traçabilité (référencé par l'article 11)

Objectif

La traçabilité a pour objectif d'assurer qu'il est possible de montrer que toutes les exigences sont respectées et qu'aucun matériau non traçable n'a été introduit.

Description

La traçabilité par rapport aux exigences doit être un facteur important dans la validation d'un système et des moyens doivent être mis à disposition pour en permettre la démonstration durant toutes les phases du cycle de vie.

Description

Object oriented programming is a fundamentally new way of thinking about software based on abstractions that exist in the real world rather than based on computational abstractions. Object oriented programming organises software as a collection of objects that incorporate both data structure and behaviour. This is in contrast to conventional programming where data structure and behaviour are only loosely connected.

Object: an object consists of a private data area and set of operations – so called methods – on that object. Methods may be public or private. No other software component is allowed to read or change the private data of an object directly. Every other software component has to use the public methods on that object to read or write data in the private data area of an object.

Object Class: by specifying an object class (often in the form of a type definition) you enable the instantiation of numerous objects of the same class, i.e., all instantiations have the private data area and the methods defined in the object class.

(Multiple) Inheritance: an object class can inherit the private data area and the methods of one (or more) superclasses (object classes above it in the class hierarchy) with being allowed to add some private data, to add some methods or to modify the implementations of the inherited methods. Using Inheritance multiple-object class trees can be built.

Polymorphism: the same operation may behave differently on different object classes, for example, the write operation for a terminal object writes characters to that terminal and a write operation to a file object writes characters to that file.

References:

Object Oriented Software Construction, Bertrand Meyer, England: Prentice-Hall International, 1988.

Classification as a paradigm for computing, Peter Wegner, Technical Report CS-86-11, Brown University, May 1986.

Learning language, Peter Wegner, Byte (McGraw-Hill publication), March 1989.

All OOPSLA (Object-Oriented Programming Systems, Languages and Applications) and ECOOP (European Conference on Object-Oriented Programming) conference proceedings.

B.69 Traceability (referenced by clause 11)

Aim

The objective of Traceability is to ensure that all requirements can be shown to have been properly met and that no untraceable material has been introduced.

Description

Traceability to requirements shall be an important consideration in the validation of a system and means shall be provided to allow this to be demonstrated throughout all phases of the life cycle.

La traçabilité doit être considérée comme applicable à la fois aux exigences fonctionnelles et non fonctionnelles et doit particulièrement concerner

a) la traçabilité des exigences par rapport à la conception ou aux autres items qui y répondent,

b) la traçabilité des items de conception par rapport aux items d'implémentation qui les instancient,

c) la traçabilité des exigences et des items de conception par rapport aux items opérationnels et de maintenance qu'il est nécessaire d'appliquer pour que l'utilisation du système soit sûre et correcte,

d) la traçabilité des exigences, des items de conception, d'implémentation, de fonctionnement et de maintenance par rapport aux plans de vérification et de test et aux spécifications qui détermineront leur acceptabilité,

e) la traçabilité des plans de vérification et de test et des spécifications vis-à-vis des tests et autres rapports qui enregistrent les résultats de leur application.

Lorsque les exigences, la conception et d'autres items sont instanciés dans plusieurs documents séparés, la traçabilité doit être conservée dans les structures de documents et d'une manière hiérarchique.

La sortie du processus de traçabilité doit être soumise à la gestion de configuration formelle.

Traceability shall be considered applicable to both functional and non-functional requirements and shall particularly address

a) traceability of requirements to the design or other objects which fulfil them,

b) traceability of design objects to the implementation objects which instantiate them,

c) traceability of requirements and design objects to the operational and maintenance objects required to be applied in the safe and proper use of the system,

d) traceability of requirements, design, implementation, operation and maintenance objects, to the verification and test plans and specifications which will determine their acceptability,

e) traceability of verification and test plans and specifications to the test or other reports which record the results of their application.

Where requirements, design or other objects are instantiated as a number of separate documents, traceability shall be maintained within the document structures and in a hierarchical manner.

The output of the Traceability process shall be the subject of formal Configuration Management.

Standards Survey

The IEC would like to offer you the best quality standards possible. To make sure that we continue to meet your needs, your feedback is essential. Would you please take a minute to answer the questions overleaf and fax them to us at +41 22 919 03 00 or mail them to the address below. Thank you!

Customer Service Centre (CSC)

International Electrotechnical Commission
3, rue de Varembé
1211 Genève 20
Switzerland

or

Fax to: **IEC**/CSC at +41 22 919 03 00

Thank you for your contribution to the standards-making process.

Nicht frankieren
Ne pas affranchir

Non affrancare
No stamp required

A Prioritaire

RÉPONSE PAYÉE

SUISSE

Customer Service Centre (CSC)
International Electrotechnical Commission
3, rue de Varembé
1211 GENEVA 20
Switzerland

Q1 Please report on **ONE STANDARD** and **ONE STANDARD ONLY**. Enter the exact number of the standard: *(e.g. 60601-1-1)*

.....

Q2 Please tell us in what capacity(ies) you bought the standard *(tick all that apply).* I am the/a:

purchasing agent ❑
librarian ❑
researcher ❑
design engineer ❑
safety engineer ❑
testing engineer ❑
marketing specialist ❑
other.....

Q3 I work for/in/as a: *(tick all that apply)*

manufacturing ❑
consultant ❑
government ❑
test/certification facility ❑
public utility ❑
education ❑
military ❑
other.....

Q4 This standard will be used for: *(tick all that apply)*

general reference ❑
product research ❑
product design/development ❑
specifications ❑
tenders ❑
quality assessment ❑
certification ❑
technical documentation ❑
thesis ❑
manufacturing ❑
other.....

Q5 This standard meets my needs: *(tick one)*

not at all ❑
nearly ❑
fairly well ❑
exactly ❑

Q6 If you ticked NOT AT ALL in Question 5 the reason is: *(tick all that apply)*

standard is out of date ❑
standard is incomplete ❑
standard is too academic ❑
standard is too superficial ❑
title is misleading ❑
I made the wrong choice ❑
other .....

Q7 Please assess the standard in the following categories, using the numbers:
(1) unacceptable,
(2) below average,
(3) average,
(4) above average,
(5) exceptional,
(6) not applicable

timeliness.....
quality of writing.....
technical contents.....
logic of arrangement of contents .....
tables, charts, graphs, figures.....
other .....

Q8 I read/use the: *(tick one)*

French text only ❑
English text only ❑
both English and French texts ❑

Q9 Please share any comment on any aspect of the IEC that you would like us to know:

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Enquête sur les normes

La CEI ambitionne de vous offrir les meilleures normes possibles. Pour nous assurer que nous continuons à répondre à votre attente, nous avons besoin de quelques renseignements de votre part. Nous vous demandons simplement de consacrer un instant pour répondre au questionnaire ci-après et de nous le retourner par fax au +41 22 919 03 00 ou par courrier à l'adresse ci-dessous. Merci !

Centre du Service Clientèle (CSC)

Commission Electrotechnique Internationale
3, rue de Varembé
1211 Genève 20
Suisse

ou

Télécopie: **CEI**/CSC +41 22 919 03 00

Nous vous remercions de la contribution que vous voudrez bien apporter ainsi à la Normalisation Internationale.

Nicht frankieren
Ne pas affranchir

Non affrancare
No stamp required

A Prioritaire

RÉPONSE PAYÉE

SUISSE

Centre du Service Clientèle (CSC)
Commission Electrotechnique Internationale
3, rue de Varembé
1211 GENÈVE 20
Suisse

Q1 Veuillez ne mentionner qu'**UNE SEULE NORME** et indiquer son numéro exact: (*ex. 60601-1-1)*

.....

Q2 En tant qu'acheteur de cette norme, quelle est votre fonction?
(cochez tout ce qui convient)
Je suis le/un:

agent d'un service d'achat ❑
bibliothécaire ❑
chercheur ❑
ingénieur concepteur ❑
ingénieur sécurité ❑
ingénieur d'essais ❑
spécialiste en marketing ❑
autre(s).....

Q3 Je travaille:
(cochez tout ce qui convient)

dans l'industrie ❑
comme consultant ❑
pour un gouvernement ❑
pour un organisme d'essais/ certification ❑
dans un service public ❑
dans l'enseignement ❑
comme militaire ❑
autre(s).....

Q4 Cette norme sera utilisée pour/comme
(cochez tout ce qui convient)

ouvrage de référence ❑
une recherche de produit ❑
une étude/développement de produit ❑
des spécifications ❑
des soumissions ❑
une évaluation de la qualité ❑
une certification ❑
une documentation technique ❑
une thèse ❑
la fabrication ❑
autre(s).....

Q5 Cette norme répond-elle à vos besoins:
(une seule réponse)

pas du tout ❑
à peu près ❑
assez bien ❑
parfaitement ❑

Q6 Si vous avez répondu PAS DU TOUT à Q5, c'est pour la/les raison(s) suivantes:
(cochez tout ce qui convient)

la norme a besoin d'être révisée ❑
la norme est incomplète ❑
la norme est trop théorique ❑
la norme est trop superficielle ❑
le titre est équivoque ❑
je n'ai pas fait le bon choix ❑
autre(s) .....

Q7 Veuillez évaluer chacun des critères ci-dessous en utilisant les chiffres
(1) inacceptable,
(2) au-dessous de la moyenne,
(3) moyen,
(4) au-dessus de la moyenne,
(5) exceptionnel,
(6) sans objet

publication en temps opportun .....
qualité de la rédaction.....
contenu technique .....
disposition logique du contenu .....
tableaux, diagrammes, graphiques, figures.....
autre(s) .....

Q8 Je lis/utilise: *(une seule réponse)*

uniquement le texte français ❑
uniquement le texte anglais ❑
les textes anglais et français ❑

Q9 Veuillez nous faire part de vos observations éventuelles sur la CEI:

.....
.....
.....
.....
.....
.....

ISBN 2-8318-6585-9

ICS 45.060

Typeset and printed by the IEC Central Office
GENEVA, SWITZERLAND