Institut pour la **M**aîtrise **d**es **R**isques
Sûreté de Fonctionnement - Management - Cindyniques

GUIDE

DEMARCHE ET METHODES DE SURETE DE FONCTIONNEMENT DES LOGICIELS

Edition 2 du 3 avril 2013

IMdR – GTR 63

« Démarche et méthodes de Sûreté de Fonctionnement des logiciels »

Ce document est la mise à jour du précédent guide, datant de mai 2000.

Il est le résultat d'un travail collectif au sein du Groupe de Travail et de Réflexion n°63 de l'Institut pour la Maîtrise des Risques.

Il prend en compte le retour d'expérience depuis la précédente version. Il résulte d'échanges entre personnes confrontées, dans leurs domaines respectifs, à la définition, au développement ou à l'expertise de logiciels devant répondre à des exigences de Sûreté de Fonctionnement.

Il privilégie les aspects pratiques en tirant partie des expériences industrielles des membres du Groupe.

Les membres ayant participé de manière active à la mise à jour de ce guide sont :

- Mihaela BARREAU (ISTIA)
- Jean-Louis BOULANGER (CERTIFER)
- Samuel COLIN (SAFE RIVER)
- Sébastien CROIZÉ (SNCF)
- Benjamin JOGUET (DCNS)
- Patrice KAHN (KSdF Conseil)
- André LEBLOND (THALES)
- Philippe LECLERCQ (RIS)
- Maximilien LAFORGE (SCHNEIDER ELECTRIC)
- Pascal SOUCHET (ALL4TEC)

SOMMAIRE

AVERTISSEMENT

Le présent guide est conçu pour présenter une démarche globale, générique, apte à être mise en œuvre lors du développement d'un logiciel, en présence de contraintes de Sûreté de Fonctionnement quel que soit le niveau de ces contraintes.

Il décrit une démarche globale s'appuyant sur des méthodes référencées et pour certaines présentées en annexe, sans toutefois décrire de manière détaillée et exhaustive toutes les méthodes que l'on peut trouver dans l'ensemble des documents abordant ce thème, que ce soit dans le domaine de la recherche ou dans un secteur industriel particulier.

Il fixe un cadre global dans lequel peuvent être utilisées des techniques et des méthodes complémentaires.

Il propose une terminologie basée sur des travaux et des normes existantes, validée par l'usage dans les principaux secteurs ayant à développer des logiciels à contraintes de Sûreté de Fonctionnement.

Il est indépendant des aspects organisationnels de la Sûreté de Fonctionnement (équipe indépendante, prise en compte de la Sûreté de Fonctionnement par l'Assurance Qualité, expertise externe, ...). Il appartient à chaque lecteur cherchant à mettre en œuvre la démarche proposée de la décliner en fonction de sa propre organisation.

1. INTRODUCTION

1.1 OBJET DU GUIDE

Le présent guide a pour but de présenter d'une manière pratique l'essentiel de l'état de l'art en matière de Sûreté de Fonctionnement des logiciels et de donner une démarche générale sur laquelle peut s'appuyer tout projet ayant à traiter avec des exigences de Sûreté de Fonctionnement.

La Sûreté de Fonctionnement est vue dans ce guide comme une (ou plusieurs) caractéristique(s) dont la construction s'appuie sur une démarche qualité (dite classique) dont les instruments de base sont les manuels qualité, les plans qualité et les procédures qualité (au sens ISO 9001), voire des niveaux de maturité reconnus (CMMI / SPICE), c'est-à-dire une démarche de développement maîtrisée, pour laquelle on considère les principes et les dispositions connus et réellement mis en œuvre. Le guide ne reprendra pas ces concepts, considérés acquis, se consacrant essentiellement aux aspects spécifiques à la Sûreté de Fonctionnement des logiciels.

Dans certains domaines, la démarche de Sûreté de Fonctionnement des logiciels fait l'objet de normes ou de recommandations particulières ; le guide n'a pas pour mission de suppléer ces références.

1.2 DOMAINE COUVERT

Le domaine de la Sûreté de Fonctionnement des Logiciels couvert par ce guide, reprend les concepts généraux que sont :

- la fiabilité,
- la disponibilité,
- la maintenabilité,
- la sécurité-innocuité (sens du terme anglais Safety).

Dans la suite du document, le terme « Sûreté de Fonctionnement » couvre ces quatre aspects. Le terme « Safety » est utilisé lorsque certaines des dispositions présentées ne s'appliquent a priori qu'à des logiciels devant répondre à des contraintes de sécurité-innocuité.

Il ne couvre pas les aspects sécurité-confidentialité (sens du terme anglais Security).

1.3 MOTIVATION

La multiplicité des techniques, des expériences et la diversité des informations et des considérations théoriques rendent complexe la généralisation de la Sûreté de Fonctionnement et l'élaboration d'une démarche efficiente. Ces éléments progressent et évoluent régulièrement : c'est ce qui a motivé le Groupe de Travail à œuvrer à la mise à jour du présent guide.

Contrairement au matériel qui tombe en panne, le logiciel ne vieillit pas et n'est sensible qu'à des fautes ou des insuffisances de conception qui, présentes dans le code dès sa mise en œuvre, se manifestent selon l'utilisation qui est faite du logiciel ou selon les défaillances du matériel ayant un impact sur le fonctionnement du logiciel. Il est nécessaire de mettre en place une démarche adaptée à cette technologie pour identifier et analyser les risques et prendre les mesures appropriées pour les réduire.

La démarche à définir pour répondre aux attentes de la communauté logiciel ne peut être spécifique d'un domaine ou d'un type de logiciel particulier. Elle doit être suffisamment générique, tout en restant pragmatique, pour être déclinée dans chaque contexte, en y puisant les éléments de base d'une réflexion, de principes d'analyse et de pratiques adaptées.

1.4 AUDIENCE DU GUIDE

Ce guide s'adresse :

- à tous ceux qui souhaitent avoir un aperçu de l'état de l'art en matière de Sûreté de Fonctionnement des logiciels au travers de la description générale de la démarche,
- à des équipes de développement ou des ingénieurs Sûreté de Fonctionnement de par les informations pratiques sur le phasage à mettre en œuvre ou sur les méthodes ou les techniques qu'il recommande.

Plus généralement, par la vision globale qu'il fournit, il peut intéresser tout acteur d'un projet logiciel : fournisseur, client, exploitant et utilisateur, ayant un rôle à jouer, à un instant ou à un autre, dans la vie du logiciel et la construction du niveau de Sûreté de Fonctionnement visé.

1.5 LIMITES DU GUIDE

Ce guide n'a pas vocation de référencer des solutions tant en matière de méthodes et de modèles que d'outils, dans la mesure où ce domaine est toujours en pleine évolution. Il cite cependant quelques méthodes jugées suffisamment représentatives et largement reconnues dans la communauté Sûreté de Fonctionnement logiciel.

1.6 STRUCTURE DU GUIDE

Le présent guide est structuré de la manière suivante :

- l'introduction positionne le guide par rapport au domaine visé, aux différents objectifs, et à l'audience qu'il peut avoir,
- le chapitre 2 décrit le contexte dans lequel l'utilisation du guide est proposée,
- le chapitre 3 rappelle les concepts et la terminologie nécessaire à la bonne compréhension du guide,
- le chapitre 4 présente les principes de base de la démarche, qui sont repris phase par phase dans les chapitres suivants,
- le chapitre 5 précise les données en entrée de la démarche, au travers de l'expression des besoins de Sûreté de Fonctionnement,
- le chapitre 6 décrit les dispositions permettant de construire la Sûreté de Fonctionnement d'un logiciel,
- le chapitre 7 décrit les dispositions permettant de vérifier et de valider la Sûreté de Fonctionnement d'un logiciel,
- le chapitre 8 décrit les dispositions permettant d'évaluer le niveau de Sûreté de Fonctionnement atteint d'un logiciel,

- le chapitre 9 récapitule la répartition des activités de Sûreté de Fonctionnement (développement et vérification) par rapport aux principales activités des phases d'un cycle de vie,
- le chapitre 10 propose la formalisation de cette démarche appliquée à un projet donné, sous la forme d'un plan de Sûreté de Fonctionnement logiciel,
- le chapitre 11 présente différents aspects complémentaires utiles à la couverture du domaine et pouvant éclairer certains points abordés dans les chapitres précédents,
- le chapitre 12 dresse une conclusion sur le guide,
- des annexes (terminologie, fiches normes, fiches méthodes, exemples) viennent compléter le présent guide.

Chaque norme ou méthode, décrite en annexe sous forme de fiche, est signalée dans le corps du document par des caractères italiques.

2. CONTEXTE

Différents contextes sont confrontés aujourd'hui à des exigences nécessitant la mise en œuvre d'une démarche particulière lors du développement d'un logiciel pour en démontrer le bon fonctionnement et l'absence de dangers.

Outre les domaines classiques que sont l'aéronautique, le nucléaire, le spatial ou le transport ferroviaire, d'autres domaines tels que le médical, l'industrie pétrolière, l'automobile, la marine, la métallurgie, le secteur tertiaire doivent intégrer la Sûreté de Fonctionnement, sous l'une ou plusieurs de ces caractéristiques pour répondre aux attentes des utilisateurs finals ou plus généralement de leurs clients.

Des normes ou recommandations existent aujourd'hui, le plus souvent propres à un domaine, elles ont servi en partie de base à l'élaboration de ce présent guide et sont présentées sous forme de fiches en annexe 2.

Ces normes ou recommandations ont chacune un contexte d'application (projets critiques) et des principes généraux ; le présent guide n'a pas pour but de choisir une de ces normes mais bien de puiser parmi celles-ci les éléments concrets et pragmatiques sur lesquels les rédacteurs de ce guide proposent de s'appuyer, au travers de leur expérience.

3. TERMINOLOGIE DE SÛRETÉ DE FONCTIONNEMENT

Là encore, le but de ce guide n'est pas de recréer, ex nihilo, une terminologie mais de rappeler, ici, la terminologie, aujourd'hui la plus couramment partagée dans le monde industriel.

Pour ce faire, les principaux termes jugés utiles sont définis en annexe 1, avec, le cas échéant la définition retenue et des définitions complémentaires.

4 PRINCIPES DE BASE DE LA DÉMARCHE DE SÛRETÉ DE FONCTIONNEMENT LOGICIEL

Positionnement de la démarche

La Sûreté de Fonctionnement n'est en général pas un objectif pour un logiciel seul ; elle se décline d'un ensemble d'exigences de plus haut niveau concernant le système, le sous-système et le logiciel dans son environnement d'utilisation (environnement matériel, procédures d'exploitation, ensemble de logiciels et de matériels intégrés, ...).

Il convient donc de mettre en œuvre une démarche de Sûreté de Fonctionnement analogue au niveau du système complet. Cette mise en œuvre doit permettre de déterminer pour le ou les logiciels à considérer, les exigences de type Sûreté de Fonctionnement auxquelles il doit satisfaire.

Dès lors, il est préférable d'assurer la cohérence des approches utilisées à ces différents niveaux (y compris pour le matériel) afin de pouvoir réutiliser les données et consolider les résultats des diverses approches.

Le logiciel doit être vu comme l'utilisation d'une technologie, au même titre que les technologies électromécaniques ou pneumatiques. Le logiciel remplit un besoin exprimé et il doit être considéré comme une solution technologique.

Les fonctions ou services que doit remplir le logiciel doivent toujours être maîtrisés par les personnes qui expriment ces besoins (des ensembliers fonctionnels par exemple). Ceci permet d'éviter une frontière trop hermétique entre le « monde » du système et le « monde » du logiciel.

Etapes de la démarche

L'hypothèse de base du présent guide est qu'une telle démarche existe et que les besoins de Sûreté de Fonctionnement du Logiciel exprimés constituent le point d'entrée des activités décrites dans ce guide.

Pour que cette hypothèse soit valide et que la démarche puisse réellement être mise en œuvre, le chapitre suivant précise les formes possibles et préférées que peut prendre cette expression de besoin.

Quoi qu'il en soit, la démarche applicable au logiciel doit s'appuyer sur des activités de Sûreté de Fonctionnement reposant sur une logique :

- construction de la Sûreté de Fonctionnement du logiciel (analyse des choix de conception orientée Sûreté de Fonctionnement et introduction des mécanismes nécessaires au niveau de Sûreté de Fonctionnement visé),

- vérification de la Sûreté de Fonctionnement du logiciel (vérification de la bonne implémentation et de l'efficacité des mécanismes déterminés lors de la construction et vérification du respect des hypothèses et des résultats des analyses de Sûreté de Fonctionnement menées),
- validation (plus éventuellement évaluation et/ou certification) du niveau de Sûreté de Fonctionnement atteint (vérification de l'adéquation vis à vis des objectifs initiaux).

Démarche et cycle de vie

Une telle démarche s'appuie sur un cycle de développement du logiciel adapté au contexte du projet. Pour la compréhension de la démarche et des recommandations du présent guide, on s'appuiera sur un cycle de vie classique reposant sur un découpage en phases :

- spécification du logiciel,
- conception préliminaire (architecture du logiciel),
- conception détaillée (modules),
- réalisation (codage),
- tests unitaires,
- intégration (Logiciel/Logiciel, Logiciel/Matériel),
- validation du logiciel.

Au-delà du cycle de vie du logiciel, un cycle de vie du système doit aussi être suivi. Il s'appuie en amont sur des phases de spécification du système et de conception du système et en aval sur des phases d'intégration-système et de validation du système.

Spécificités du logiciel

Toute association d'idées, par référence au matériel ne peut que troubler les esprits. Ici point de défaillance de composant, au sens du passage de ce composant d'un état de bon fonctionnement à un état de défaillance.

Les défaillances du logiciel ne peuvent pas être traitées comme les défaillances du matériel. Là où les défaillances du matériel sont aléatoires, les défaillances du logiciel sont systématiques. Si leur manifestation dépend de l'utilisation du logiciel, l'introduction de leur cause dépend avant tout d'activités humaines :

- c'est l'humain qui spécifie,
- c'est l'humain qui conçoit et qui implémente en partie ou en totalité,
- c'est l'humain qui teste,
- c'est l'humain qui valide,
- c'est l'humain qui juge l'adéquation du résultat final et la conformité aux référentiels législatifs,

C'est donc l'humain qui introduit involontairement des anomalies et qui n'a pas la capacité (ressources, compétences, imagination, …) pour les éliminer toutes ou pour concevoir les mécanismes permettant de les rendre inoffensives. Les anomalies résiduelles se manifesteront à l'utilisation.

Dans le cas de l'utilisation d'approches formelles ou d'outils de génération automatique de code, l'activité humaine reste malgré tout toujours fortement présente.

Dans le cas présent, nous avons des défauts (fautes) dans le logiciel résultant d'un dysfonctionnement du processus de développement ou de maintenance (spécification, conception, implémentation ou activités V&V). Ces défauts dont l'activation entraîne une erreur ont pour manifestation une défaillance, effet non souhaité sur le fonctionnement du système ou plus précisément de la machine numérique (informatique) incluant ce logiciel.

En d'autres termes, il n'y a pas altération du logiciel en cours de fonctionnement mais uniquement la manifestation, dans certaines conditions d'entrées, d'un défaut de celui ci.

Si les conditions liées à l'environnement du logiciel et/ou à son utilisation sont réunies, l'erreur apparaîtra toujours, indéfiniment. On peut dire ici que l'apparition de la défaillance est déterministe.

La démarche de vérification (et notamment le test) consiste à faire fonctionner le logiciel (ou simuler son fonctionnement) dans un nombre très important de conditions de sollicitations, en recherchant sans pouvoir le garantir que ce nombre puisse être exhaustif. De ce fait, la mise en œuvre de ce logiciel peut rencontrer un certain nombre de conditions entraînant une défaillance, non exercées durant les tests, sans que l'occurrence puisse en être prévue.

Ainsi, aux yeux de l'utilisateur, les erreurs apparaîtront dans des conditions apparemment aléatoires et viendront se joindre aux défaillances du matériel. Dans ces conditions l'utilisateur verra les manifestations des uns et des autres comme un tout et peu lui importe la cause, il fera le constat de la non-disponibilité de sa machine, de son système.

Exemplaire unique, même s'il est dupliqué, chaque version (chaque modification) peut introduire, en même temps que les améliorations qui justifient la nouvelle version, son lot d'anomalies ou simplement modifier un fonctionnement jusqu'alors considéré satisfaisant.

La complexité de sa conception et de son développement, ajouté à la combinatoire de ses modes de fonctionnement en fait un produit difficile à maîtriser dont la Sûreté de Fonctionnement doit se construire et se vérifier méticuleusement, faute de quoi, les défaillances en exploitation peuvent être nombreuses et le plus souvent de conséquences graves.

La taille et la complexité des logiciels actuels rendent utopique de vouloir par construction ou par contrôle en fin de développement, éviter toutes les fautes contenues dans un logiciel. Le logiciel zéro-défaut n'existe pas, il est donc nécessaire de mettre en place tout ce qui permettra de tolérer la présence de fautes, tout en garantissant l'absence d'événement redouté dont les conséquences sont jugées trop graves.

Il est dès lors nécessaire, pour assurer cette maîtrise, de mettre en place une démarche adaptée à cette technologie pour identifier et analyser les risques et prendre les mesures appropriées pour les réduire.

Les standards et normes actuels partent du principe qu'il est difficile de quantifier une probabilité de défaillance d'un logiciel, et favorisent de ce fait l'approche qualitative.

Néanmoins, il existe deux types de modèles visant à quantifier la fiabilité d'un logiciel :

- les modèles de croissance de fiabilité, utilisés en exploitant le retour d'expérience issu des tests et de l'utilisation du logiciel,
- les modèles prévisionnels fournissant des éléments quantifiés permettant d'orienter le processus et les méthodes de développement.

Ces modèles sont globalement peu utilisés car, s'ils permettent d'aider dans l'analyse du comportement prévisible d'un logiciel, la plupart des hypothèses sur lesquelles ils s'appuient est sujette à débat. Les résultats sont par ailleurs peu significatifs au regard des limites quant à leur utilisation.

Pour toutes ces raisons, une démarche de Sûreté de Fonctionnement du logiciel doit reposer sur l'utilisation combinée de différentes techniques complémentaires, puisque aucune technique individuelle ne peut garantir l'absence de fautes résiduelles.

En plus des activités spécifiques de Sûreté de Fonctionnement, la démarche de Sûreté de Fonctionnement logiciel doit être accompagnée d'activités qualité modulées en fonction de la criticité des logiciels. Des dispositions spécifiques sont nécessaires pour la réalisation des composants critiques. Elles consistent en un renforcement des exigences sur les méthodes et les outils à appliquer à ces composants, ainsi que sur les vérifications et les contrôles auxquels ils sont soumis. Le niveau d'exigence qualité d'un logiciel augmente avec le niveau de Sûreté de Fonctionnement exigé. Ainsi, des techniques seront choisies et mises en œuvre, en fonction des niveaux de criticité des composants à développer.

5. EXPRESSION DE BESOINS EN MATIERE DE SÛRETÉ DE FONCTIONNEMENT

L'expression des besoins ne doit pas être vue comme une phase du cycle de vie Sûreté de Fonctionnement, mais plus comme le point de départ de la démarche proposée.

Cette expression de besoin est un préalable nécessaire. L'identification des exigences de Sûreté de Fonctionnement du logiciel est le résultat de la démarche d'allocation qui se fait à partir des exigences de Sûreté de Fonctionnement du système et des choix de conception associés, pour ensuite être déclinées au niveau du logiciel.

Cette première activité de la démarche, essentielle pour le choix et la justification des méthodes et des démonstrations attendues, repose sur des principes variés et pas toujours bien maîtrisés.

On y retrouve, par exemple :

- des exigences pour le logiciel seul,
- des exigences pour le logiciel et le matériel reposant sur des exigences système, avec ou sans allocation sur les constituants, et avec ou sans prise en compte des interactions matériel et logiciel. La détermination de ces exigences repose sur l'architecture du système et suit les différents niveaux de décomposition de celle-ci.

La traçabilité entre besoins système et besoins logiciel est une activité de niveau système mais est aussi une information très utile pour la définition du logiciel.

Types d'exigences

L'expression des besoins en matière de Sûreté de Fonctionnement pour un logiciel résulte aussi d'une conception système incluant des analyses de Sûreté de Fonctionnement système permettant d'allouer les exigences de Sûreté de Fonctionnement sur chacun des constituants du système et de recenser les événements redoutés ou les besoins fonctionnels à sécuriser, les fonctions de Sûreté de Fonctionnement nécessaires à la satisfaction des choix d'architecture système.

Les exigences doivent priorisées. Elles doivent être présentées de manière formalisée et être vérifiables.

Elles peuvent être exprimées en termes de :

- événements redoutés du logiciel déclinés de ceux du système et de son architecture et des Analyses de Risques système, avec, le cas échéant, une classification des défaillances,
- exigences de Sûreté de Fonctionnement concernant les fonctionnalités et les performances opérationnelles du logiciel directement déduites des fonctionnalités du système (exemple : fonction Safety intrinsèque),

- définition des modes dégradés et des conditions de passage entre modes, ainsi que les fonctionnements attendus dans chacun des modes dégradés et des comportements aux limites,
- exigences de Sûreté de Fonctionnement inhérentes au choix de conception système pour prise en compte d'exigences opérationnelles de Sûreté de Fonctionnement système (exemple : fonction de surveillance d'état du système),
- exigences opérationnelles qualitatives de type comportemental et relatives à la stratégie de tolérance aux fautes (exemple, respect du critère de défaillance unique, fail-op, fail-safe, fail-op/fail-safe, ...),
- exigences opérationnelles, qualité de service, quantifiées du logiciel héritées ou déduites des exigences opérationnelles (exemple : durée d'exploitation, effort et fréquence de maintenance,),
- exigences opérationnelles qualitatives héritées des exigences opérationnelles système (exemple : facteur qualité : robustesse, fiabilité, crédibilité, ergonomie, avec référence aux normes définissant les termes utilisés),
- exigences sur le processus de développement (utilisation de telles méthodes de construction ou de vérification, comme : AMDE ou AMDEC (AEEL), Arbres de Fautes, preuves, ...)
- exigences sur le processus de démonstration de conformité à caractère réglementaire (exemple : respect d'une norme traitant de Sûreté de Fonctionnement).

Les exigences quantitatives permettent de positionner le niveau de criticité du système et peuvent conduire par allocation de fiabilité à fixer les exigences que le logiciel doit satisfaire. Cependant, pour le logiciel, une telle exigence est difficile à évaluer, mais peut servir de référence en termes de niveau de Sûreté de Fonctionnement et peut être une base d'une analyse de comparaison de solutions.

De plus, une exigence quantitative ne doit pas être formulée seule et globalement pour un logiciel donné. Ainsi, une probabilité d'occurrence d'une défaillance d'un logiciel n'a pas véritablement de signification si cette exigence n'est pas accompagnée de la désignation de la ou des fonctions (ou événements) concernées ainsi que de la définition du profil d'utilisation du logiciel auxquels les exigences se réfèrent. C'est pourquoi l'exigence quantitative telle qu'un MTBF n'est pas recommandée car non significative pour le logiciel. C'est d'autant plus vrai que, pour évaluer ensuite une probabilité, il faut lui associer une durée et que cette dimension n'est pas pertinente dans le cas du logiciel (durée calendaire, durée d'activation,..) car elle ne rend pas compte de la sollicitation du logiciel tel que mis en œuvre par « l'humain ».

La plupart des normes et standards utilisent le niveau d'intégrité de la sécurité (SIL : Safety Integrity Level) couvrant les aspects quantitatifs et qualitatifs au niveau du système. Pour le logiciel, cela conduit à définir les exigences qualitatives et les exigences techniques nécessaires à l'obtention d'un degré de confiance et du niveau de rigueur associé.

Fonctions de sécurité (Safety) et Sûreté de Fonctionnement

En particulier, la déclinaison des exigences du système peut déboucher sur la définition d'une part de fonction dite « de sécurité » et d'autre part d'exigences de Sûreté de Fonctionnement, y compris sur les fonctions Safety.

La définition des fonctions Safety s'appuie sur une démarche d'analyse fonctionnelle qui aboutit à la spécification de fonctions ayant pour objectif d'assurer la Safety du système visé, telle que des fonctions de surveillance, d'auto-test, de reconfiguration du système.

La Sûreté de Fonctionnement d'une fonction est une caractéristique de cette fonction assurant, l'aptitude de cette fonction à effectuer les travaux (spécifications) qui lui sont assignées.

En ce sens, selon les missions du système, on peut considérer que tout ou partie des fonctions ont des contraintes de Sûreté de Fonctionnement (sinon, il n'y a pas lieu de considérer le système ni le logiciel, comme devant être sûr de fonctionnement).

Les fonctions Safety ont elles aussi des exigences de type Sûreté de Fonctionnement à respecter, un auto-test doit se déclencher à toute ses sollicitations, une surveillance doit être active en permanence.

La répartition des fonctions opérationnelles en classes de criticité, le choix d'introduire dans le système certaines fonctions Safety et la définition des exigences associées à ces deux types de fonction doivent reposer sur une démarche d'analyse fonctionnelle permettant une vision d'ensemble du besoin du système et du logiciel.

6. CONSTRUCTION DE LA SÛRETÉ DE FONCTIONNEMENT LOGICIEL

Ce chapitre présente les méthodes et techniques spécifiques à mettre en œuvre lors des activités de développement du logiciel pour construire sa Sûreté de Fonctionnement.

Il présente :

- dans un premier temps les techniques d'analyse des choix de conception orientées Sûreté de Fonctionnement,
- dans un second temps les techniques et mécanismes particuliers à mettre en œuvre dans le logiciel pour atteindre le niveau de Sûreté de Fonctionnement requis.

Ces techniques sont présentées par activités / phases de développement du logiciel.

La construction de la Sûreté de Fonctionnement des logiciels va s'exercer principalement lors des activités de spécification, conception et codage. Lors des activités de tests du logiciel, la démarche de Sûreté de Fonctionnement s'oriente essentiellement vers des activités de vérification et de validation (voire d'évaluation, de qualification ou de certification) du niveau de Sûreté de Fonctionnement atteint, mais aussi sur des activités permettant de préparer le retour d'expérience.

Les activités de construction de la Sûreté de Fonctionnement vont avoir un impact sur :

- les spécifications du logiciel en permettant d'améliorer la couverture des exigences de besoins de Sûreté de Fonctionnement,
- les choix de conception (architecture y compris) afin qu'ils permettent de satisfaire le mieux possible les contraintes de Sûreté de Fonctionnement,
- la conception et la mise en œuvre de mécanismes de détection et de maîtrise de défauts logiciels internes ou de défaillances du matériel avant que cela n'entraîne une défaillance du logiciel ou du système contraire à la Sûreté de Fonctionnement,
- les principes de codage en réduisant autant que possible les constructions permissives ou considérées comme risquées,
- les activités de tests pour identifier tous les tests spécifiques permettant de valider que les exigences de Sûreté de Fonctionnement sont satisfaites,
- les activités de vérification et de validation (V&V) à mettre en place tout au long du cycle de vie,
- l'exploitation opérationnelle du logiciel en guidant, par exemple, la mise en place de moyens et procédure de diagnostic et reconfiguration.

Au final, toutes les techniques, les activités et les processus mis en œuvre permettent :

- l'évitement des fautes, par la mise en place d'un processus méthodologique et qualité renforcé,
- l'élimination des fautes, par la mise en place de vérifications et de tests,
- la tolérance aux fautes, par la conception d'une architecture spécifique et la sélection de techniques de programmation spécifiques et la mise en œuvre de mécanismes permettant d'apporter une réponse satisfaisante en cas d'occurrence de fautes ou de défaillances du matériel supportant le logiciel.

6.1. ANALYSE DE SURETE DE FONCTIONNEMENT

La démarche repose essentiellement sur une analyse prévisionnelle des risques dont le but est d'identifier les parties critiques du logiciel et les actions en réduction de risques associées.

6.1.1. Analyse de risques

Cette analyse peut être réalisée au moyen d'une analyse qualitative déductive par Arbre de Fautes - aussi appelé Arbre de Cause ou Arbre de défaillance - ou d'une analyse inductive de type Analyse des Effets des Erreurs du Logiciel (AEEL). Les deux techniques mentionnées précédemment sont des techniques d'analyse statique. Selon le type de logiciel, des techniques d'analyse comportementale peuvent également être utilisées en complément. A ces analyses qualitatives peuvent s'ajouter une analyse quantitative permettant de comparer diverses solutions de développement.

Ces analyses doivent se dérouler en parallèle et en liaison étroite avec les activités de spécification et de conception, de manière continue et itérative. Elles doivent être réalisées en liaison étroite avec les équipes en charge du logiciel.

Elles nécessitent de disposer en entrée :

- des événements redoutés pour le logiciel,
- d'une description du logiciel (au niveau spécification, puis au niveau architecture).

Ces analyses consistent à identifier toutes les causes et les scénarios susceptibles de conduire à la réalisation d'un événement redouté.

A partir de ces scénarios, des actions en réduction de risque sont à identifier :

- lors des activités de spécification :
    - spécification de modes de fonctionnement dégradés pour supprimer ou diminuer la gravité des conséquences du dysfonctionnement considéré. La spécification de ces modes de fonctionnement dégradés doit être cohérente avec les exigences de tolérance aux fautes (fail-stop, fail-op, fail-safe, ...),
    - définition d'invariants Safety,
    - études complémentaires spécifiques visant à démontrer l'improbabilité du risque (modélisation, simulation, ...),
    - identification d'actions (tests, analyse, démonstration, …) de validation spécifiques visant à démontrer que le scénario ne va pas se produire,
    - contraintes sur l'architecture matérielle et logicielle (nécessité d'un code correcteur d'erreur, de checksum, diversification des composants logiciels, ...),
    - contraintes sur la testabilité et l'observabilité en opération.

- lors de la conception :
  - implantation de mécanismes de détection et de traitement d'erreur,
  - raffinement d'invariants Safety,
  - choix de conception minimisant les risques (choix d'architecture, de structures de données, ...),
  - identification de contraintes sur l'exploitation et la maintenance du système et du logiciel minimisant les risques (précautions d'utilisation, limites de fonctionnement, sauvegarde/restauration, surveillances système, matériels, réseaux, disques, base de données...),
  - définition de la stratégie et des moyens associés pour les procédures opérationnelles de diagnostic, de reconfiguration et de reprise,
  - identification de tests unitaires ou d'intégration spécifiques permettant de démontrer l'efficacité des mécanismes implémentés.

Différentes techniques contribuent à cette activité :

Arbre de Fautes

L'analyse par Arbre de Fautes est bien adaptée pendant les activités de spécification ou de conception. Elle n'est pas forcément exhaustive mais offre l'avantage de focaliser les efforts sur des événements précis. De plus, elle permet d'assurer facilement la continuité de l'analyse entre les activités de spécification et de conception, l'arbre étant développé au fur et à mesure que la définition du logiciel est détaillée. Les différentes causes d'un même événement étant regroupées dans un arbre unique, il est plus facile d'appréhender à quel niveau il sera le plus efficace d'intervenir pour diminuer le risque.

AMDE

L'*AMDE* peut paraître plus exhaustive que les Arbres de Fautes, mais le problème qui se pose, lorsqu'on considère le logiciel, est la définition de la liste des modes de défaillance à envisager. Or l'efficacité de l'AMDE repose sur la représentativité de ces modes de défaillances et la complétude de cette liste. En outre, l'AMDEC de façon classique ne permet pas de tracer un scénario mais la propagation d'une erreur au travers du logiciel. De plus l'AMDE est plus coûteuse dans la mesure où elle impose de considérer tous les modes de défaillances y compris ceux qui se révèlent n'avoir pas d'impact vis-à-vis des exigences de Sûreté de Fonctionnement. Lorsque la définition du logiciel est affinée (passage de la spécification à la conception) elle demande également un effort supplémentaire pour la remontée des informations.

L'*AMDE* peut également être utilisée dans une version simplifiée (c'est à dire basée sur une liste réduite de modes de défaillance) menée sur la conception détaillée, pour définir les points d'observation du logiciel en opération nécessaires à une intervention externe de diagnostic et pour définir les procédures associées de diagnostic, de reconfiguration et de reprise externes au logiciel. L'établissement de ces procédures nécessite de faire une synthèse des résultats de l'*AMDE* du logiciel et de l'*AMDEC* du matériel associé au logiciel. Cette utilisation de l'*AMDE* concerne les systèmes informatiques dont la Sûreté de Fonctionnement (souvent il s'agit de disponibilité) est assurée par un système de contrôle indépendant. C'est le cas par exemple de certains systèmes embarqués qui sont surveillés par un centre de contrôle au sol ou de systèmes informatiques où les opérations de diagnostic et de reprise sont confiées à un opérateur.

Analyses comportementales

Les analyses comportementales ou de performances comme la modélisation par automates, files d'attente ou réseaux de Petri permettent de mettre en évidence des problèmes de :

- définition incorrecte ou incomplète des modes de fonctionnement ou des transitions entre modes et donc des spécifications,
- dimensionnement, partage de ressources,
- synchronisation des traitements avec les entrées/sorties,
- ordonnancement des tâches,
- protocole de communication.

Ce type de modélisation est surtout utile pour aider ou valider le choix entre plusieurs implémentations possibles.

Analyses prévisionnelles

Les analyses prévisionnelles ont pour but d'évaluer la fiabilité d'un logiciel par la description de sa structure, de son processus de mission et de son mode développement, avec pour objectifs principaux :

- prévoir les niveaux de probabilité ou le nombre de manifestations de défaillances en développement et en opérationnel,
- permettre de comparer différentes solutions de développement et choisir celle qui présente le moins de risque que ce soit en terme de choix technologique ou d'effort de développement ou des test,
- pouvoir comparer ses résultats aux objectifs qui peuvent être fixés par contrat dans les normes qui y sont attachées.

La vérification exhaustive du comportement du logiciel est souvent impossible car elle se heurte aux limites des outils existants. Par contre, l'exploitation par simulation du modèle fournit un bon support pour l'analyse de Sûreté de Fonctionnement.

6.1.2. Classification des logiciels

A partir des résultats des analyses précédentes, il est conseillé d'effectuer une classification des fonctions logicielles, puis des composants issus de la conception, en catégories de criticité.

Le but de cette classification est d'adapter à la criticité du logiciel, les contraintes en termes de processus de développement ou de mécanismes à implanter dans le produit logiciel.

Ces contraintes peuvent être, par exemple, sélectionnées parmi celles-ci :

- développement formel : "spécification" et preuves formelles, génération automatique de code,
- règles de codage spécifiques,
- programmation défensive,
- inspection (opérations formalisées de contrôle et de vérification) du code et de la documentation,
- équipe de test indépendante,
- objectifs de couverture de tests plus sévères.

Cette classification fournit également des éléments pour la détermination de solutions à mettre en place pour garantir l'isolement de logiciels de criticité différente.

L'arbre des fautes peut fournir un bon support pour la répartition des logiciels ou des parties du logiciel dans les différentes catégories de criticité ; il permet aussi de prendre en compte la notion de fautes multiples.

Cela suppose bien évidemment qu'une classe de gravité ait été affectée aux événements redoutés pour le logiciel par l'analyse système.

6.2. MISES EN ŒUVRE DES CHOIX DE CONCEPTION

6.2.1. Conception

Lors des activités de conception, certains choix doivent être faits afin de garantir l'obtention du niveau requis de Sûreté de Fonctionnement. Ces choix portent sur la stratégie de tolérance aux fautes ainsi que sur les solutions techniques permettant de se prémunir contre la « pollution » d'un logiciel critique par un logiciel de criticité moindre.

Technique de tolérance aux fautes

Les techniques de tolérances aux fautes visent à :

- éviter la propagation des défaillances d'un composant du logiciel à tout le système,
- assurer la continuité du service malgré la défaillance d'un composant.

Dans le premier cas, on aura recours à des techniques de programmation défensive et à des traitements d'exception. La programmation défensive peut concerner aussi bien la vérification des données que le contrôle du flot d'exécution. Ces deux aspects sont traités au paragraphe suivant (codage). La définition de ces mécanismes et de la politique de gestion des exceptions est guidée par les analyses de Sûreté de Fonctionnement logiciel.

Dans le second cas, il faut employer des techniques de diversification fonctionnelles qui sont :

- la programmation en N-versions qui procède par masquage des fautes,
- les blocs de recouvrement qui procèdent par détection d'erreur puis recouvrement par reprise,
- la programmation N-autotestable qui procède par recouvrement d'erreur par compensation.

La pénétration dans l'industrie de ces techniques reste encore limitée. On peut citer l'utilisation de la *programmation N-autotestable* dans les systèmes avioniques civils (Airbus A320, ...), de la *programmation en N-versions* dans certaines applications ferroviaires (poste d'aiguillage en Suède, système de contrôle des gares en Italie, ...) et des blocs de recouvrement à EDF (Plan de défense du réseau).

Concernant la *diversification fonctionnelle*, notamment dans le cas de la *programmation en N-versions*, le problème qui se pose (outre le coût) lors de sa mise en œuvre est de savoir sur quels facteurs jouer pour obtenir une diversification réduisant la présence de fautes corrélées dans les différentes variantes. On a cependant constaté que cette technique permettait de mettre en évidence des fautes de spécification (omission, ambiguïté, ..) du fait de l'exploitation de la spécification par des équipes de développeurs indépendantes. Malgré tout, l'erreur de spécification peut être un mode commun de ces techniques de redondance ou de diversification et donc en réduire l'efficacité voire l'intérêt. C'est pourquoi il est absolument nécessaire d'apporter des garanties suffisantes sur les spécifications, pour qu'elles soient non ambigües et non interprétables. On peut citer ici les techniques de simulation ou de validation formelle.

Isolation de logiciels de criticités différentes

Au sein d'une application logicielle, tous les composants logiciels n'ont pas toujours la même criticité. Pour des raisons économiques, il n'est pas, dans la majorité des cas, possible de développer toute l'application avec des contraintes correspondant au niveau de criticité maximale. Les logiciels moins critiques sont plus sujets à contenir des fautes résiduelles. Il faut donc employer des techniques particulières pour garantir qu'un logiciel non critique ne va pas venir polluer un logiciel critique. Le choix de ces techniques est à faire lors des activités de conception.

Les risques à considérer sont :

- pollution des données, du code exécutable ou de l'environnement d'exécution du logiciel critique par le logiciel non critique,
- défaillance dans l'activation du logiciel critique suite à une monopolisation des ressources (UC, mémoire, périphériques, ...) par le logiciel non-critique ou à un branchement inopiné dans le logiciel critique de la part du logiciel non-critique.

Une première solution est la ségrégation matérielle, mais elle n'est pas toujours possible.

Une deuxième solution est d'avoir recours à des solutions techniques permettant la cohabitation de logiciels de criticité différentes dans un même calculateur tout en garantissant l'intégrité des logiciels critiques vis-à-vis des défaillances des logiciels non-critiques. Ces solutions peuvent être :

- l'utilisation de processeurs proposant des mécanismes matériels de protection d'accès (segmentation de la mémoire, association de privilèges à chaque processus) pour éviter les problèmes de pollution de la mémoire,
- l'utilisation d'une gestion préemptive des processus ou tâches garantissant l'allocation de l'UC au logiciel critique,
- l'interdiction d'allocation dynamique de la mémoire et interdiction de récursivité directe ou indirecte,
- l'accès aux périphériques par l'intermédiaire d'un gérant de ressources chargé de libérer les ressources pour les logiciels critiques,

- encapsulation des données.

6.2.2. Conception détaillée et Codage

Lors des activités de conception détaillée et de codage, un certain nombre de techniques peuvent être employées pour la réalisation des logiciels critiques. Le choix de ces techniques est guidé par le type d'application et par les résultats des analyses de Sûreté de Fonctionnement.

Programmation défensive

La programmation défensive vise à détecter des situations susceptibles d'entraîner la défaillance d'un composant logiciel ou du système.

Elle peut détecter des erreurs soit sur les données, soit sur les flots de contrôle. Cette technique peut être utilisée pour la protection d'un composant critique vis-à-vis des défaillances d'un composant moins critique.

En ce qui concerne les erreurs sur les données, la programmation défensive consiste à insérer dans le code des assertions sur les données contrôlant leur correction avant leur utilisation pour les données d'entrée ou leur diffusion pour des données de sorties. Ces assertions peuvent revêtir différentes formes (appartenance à une plage de valeur, contrôle de vraisemblance, invariant liant plusieurs données, ...). Il est à noter que l'utilisation d'un langage fortement typé contribue à la programmation défensive.

La détection des erreurs sur le flot de contrôle d'un programme, peut se faire en insérant dans le code un automate qui vérifie les enchaînements des activités et leurs durées.

Cette technique peut amener un surcoût important en ce qui concerne les temps d'exécution (le surcoût en termes de taille du code est souvent négligeable), il est important de bien cibler les vérifications à mettre en place en fonction des résultats des analyses de Sûreté de Fonctionnement qui mettent en évidence les données et les enchaînements sensibles.

Langage et règles de programmation

Les langages de haut niveau sont recommandés car ils fournissent certains mécanismes de contrôle pouvant être utilisés dans le cadre de la programmation défensive.

Pour les logiciels critiques, les règles de codage doivent interdire les instructions ou les constructions du langage connues pour être non déterministe (en terme d'exécution ou d'exigence mémoire) ou difficiles à utiliser. On peut citer notamment les règles de codage MISRA-C, reconnues et appliquées dans tous les secteurs pour les logiciels critiques développés en langage C.

Logiciels et temps réel

Les choix d'architecture et de principes de conception, comme l'utilisation de processeurs distribués ou non, le traitement temps réel par interruption ou par cycle, la mise en œuvre de protocoles de communication synchrones ou asynchrones, la modélisation plus ou moins parfaite de phénomènes physiques (eux-mêmes plus ou moins contraignants) constituent autant d'éléments qui vont influer sur la Sûreté de Fonctionnement d'un logiciel et du système dans lequel il s'intègre, tant sur le plan de la construction de la Sûreté de Fonctionnement que de sa vérification. En effet, la validation d'un logiciel temps réel, fonctionnant sous interruption est très difficile, à tel point que pour les logiciels critiques, certaines normes proscrivent l'utilisation d'interruptions, de mécanismes non déterministes (asynchrones). Dans certains cas, les règles de bonne pratique utilisées vont jusqu'à interdire aussi l'emploi de sous-programme ou de branchements conditionnels.

C'est pourquoi on privilégie la réutilisation de composants (noyaux temps réels, …) éprouvés ou évalués conformes aux normes en vigueur.

Gestion de l'allocation du CPU aux composants logiciels critiques

Cette gestion peut être faite par un séquenceur. Dans ce cas, une plage de temps est réservée à l'exécution de chacun des processus et le séquenceur traite les débordements (désactivation du processus en débordement et activation du processus suivant). Un timer armé lors de l'activation du processus génère une interruption traitée par le séquenceur lorsque le temps maximum alloué au processus est écoulé.

Le séquenceur offre l'avantage que le comportement de l'application est déterministe. Cependant cette solution requiert d'avoir un nombre de processus limité et une application majoritairement cyclique.

Dans le cas d'un système multitâche, l'ordonnanceur de tâches doit avoir la plus haute priorité et être capable de réveiller les tâches contenant les logiciels critiques avec une précision suffisante et de préempter les tâches contenant les logiciels non critiques. Il ne faut pas que les priorités soient calculées dynamiquement au cours de l'exécution car cela risque de rendre un logiciel non critique prioritaire sur un logiciel critique. Les tâches correspondant aux logiciels non critiques doivent avoir une priorité plus faible que la priorité la plus faible des tâches contenant un logiciel critique. Si les logiciels non critiques utilisent de l'assembleur, il faut s'assurer qu'ils ne contiennent pas de masquage d'interruption, ni de passage en section non préemptive.

Cette solution permet certainement une plus grande évolutivité du logiciel que le séquenceur, mais nécessite de définir précisément le comportement temporel des logiciels critiques et des logiciels non critiques. Pour cela, une modélisation par *Réseaux de Petri* ou automates communicants temporels est bien adaptée. Elle permet de s'assurer que les contraintes de durée et les priorités affectées sont correctes. Il est conseillé d'éviter de créer des tâches en dehors de l'initialisation du logiciel afin que les allocations de temps CPU puissent être maîtrisées dès la conception du logiciel.

Quelle que soit la solution choisie, il est évident qu'une même tâche ou processus ne doit pas contenir de logiciels de criticités différentes.

Génération automatique de code

La génération automatique de code est un moyen de minimiser l'introduction d'erreur lors du codage et contribue ainsi à l'amélioration de la Sûreté de Fonctionnement du logiciel.

Les outils utilisés aident notamment à renforcer la conformité avec des règles de codage en automatisant la transcription de la conception du logiciel en code source ou permettent d'effectuer automatiquement certaines vérifications.

Cependant la génération automatique de code suppose l'utilisation d'une description formelle du logiciel lors des activités de spécification et de conception qui peut, elle aussi, être génératrice de difficultés, ainsi que d'un environnement de développement formel.

Pour obtenir un gain de Sûreté de Fonctionnement au niveau du codage, il faut avoir des générateurs de code dans lesquels on puisse avoir confiance, c'est à dire soigneusement validés et éprouvés, activités qu'il faut rééditer à chaque version de ces outils.

Certains secteurs imposent, par l'application de normes ou standards, que ces outils de génération automatique de code soient évalués par un organisme indépendant, d'un point de vue Safety.

Technique du processeur codé, code correcteur d'erreur

Ces techniques permettent de se prémunir contre la corruption des données ou du code souvent d'origine matérielle en zone mémoire.

6.2.3. Tests

Lors des activités de tests, en plus des activités de vérification et de validation de la Sûreté de Fonctionnement qui sont traitées dans les chapitres suivants, la démarche de Sûreté de Fonctionnement va s'attacher à définir et mettre en place les moyens nécessaires au maintien du niveau de Sûreté de Fonctionnement lors des corrections ou évolutions du logiciel.

L'aspect maintien (non régression) du niveau de Sûreté de Fonctionnement atteint est à prendre en compte lors de la définition de la stratégie de test et des moyens associés.

De plus, lors de modifications du logiciel, l'impact de ces modifications vis-à-vis de la Sûreté de Fonctionnement doit être étudié afin de s'assurer que l'on ne dégrade pas le niveau de Sûreté de Fonctionnement. Si elles ont un impact sur les analyses de Sûreté de Fonctionnement effectuées lors des activités de spécification et de conception, ces analyses doivent être mises à jour car il est important de maintenir la traçabilité entre les exigences de Sûreté de Fonctionnement et leur implémentation dans le logiciel.

Ces analyses vont également permettre d'identifier les composants du logiciel impactés par la modification, d'optimiser la stratégie à retenir pour les tests unitaires, les tests d'intégration et les tests de validation (tout ou partie des scénarios de tests à rejouer) dès lors que des précautions architecturales sont prises (modularité et étanchéité).

7. VERIFICATION DE LA SÛRETÉ DE FONCTIONNEMENT LOGICIEL

Ce chapitre présente les méthodes et techniques spécifiques à mettre en œuvre lors des activités de développement et de test du logiciel pour vérifier sa Sûreté de Fonctionnement.

Il présente dans un premier temps les principes généraux liés à la vérification de la Sûreté de Fonctionnement d'un logiciel, lors de son développement, puis, les principes de vérification associés aux activités de construction.

Il présente ensuite les activités et techniques de vérification du respect des exigences de Sûreté de Fonctionnement réalisées en phases de tests, ainsi que les principes liés à la vérification des procédures d'exploitation. Enfin, il aborde les aspects planification des actions de vérification.

7.1 PRINCIPES DE VERIFICATION DE LA SÛRETÉ DE FONCTIONNEMENT DU LOGICIEL

De même que la construction de la Sûreté de Fonctionnement doit être une activité continue, image d'une préoccupation de tous les instants, la vérification de la Sûreté de Fonctionnement est un processus continu mis en place dès le début du projet, afin de détecter au plus tôt les risques de non Sûreté de Fonctionnement et s'assurer que les dispositions prises et analyses effectuées répondent aux exigences définies.

Un manque de Sûreté de Fonctionnement constaté en phase finale du développement peut être irrémédiable pour le système ; le changement de l'architecture ou des principes de tolérance aux fautes devenant incompatibles des délais de mise à disposition du système, sans parler des aspects financiers.

La vérification de la Sûreté de Fonctionnement des logiciels va s'exercer dès les activités de spécification, conception et codage, afin de s'assurer de la compatibilité des choix effectués, en construisant la Sûreté de Fonctionnement, avec les objectifs visés, puis se poursuivre et s'intensifier lors des activités de tests du logiciel, l'ensemble de la démarche de Sûreté de Fonctionnement s'orientant alors essentiellement vers des activités de vérification, avant de se finaliser par des activités d'évaluation du niveau de Sûreté de Fonctionnement atteint.

La vérification de la Sûreté de Fonctionnement est à considérer comme une activité à part entière ayant pour objet de démontrer l'adéquation des choix effectués et à tout le moins donner confiance en ce que la Sûreté de Fonctionnement ainsi construite permet de répondre aux objectifs du logiciel et du système dans lequel il s'insère.

Elle peut s'appuyer sur des techniques de type contrôle qualité appliquées aux résultats des activités de construction de Sûreté de Fonctionnement (complétude, cohérence, lisibilité, testabilité, ...).

Les activités de vérification de la Sûreté de Fonctionnement vont permettre :

- de conforter, et si possible démontrer, les choix effectués lors des activités de construction dans les étapes spécification, conception, réalisation du cycle de vie,
- de vérifier l'efficacité des dispositions prises, par des actions spécifiques, notamment lors des étapes de test du logiciel, puis du système,
- de finaliser les principes liés à l'exploitation opérationnelle du logiciel en s'assurant de l'efficacité des moyens et procédures mis en place pour le diagnostic et/ou les reconfigurations.

Les activités de vérification en phase de tests vont également permettre de s'assurer de la validité du classement des composants logiciels selon leur criticité et en cas de divergence (défaillance d'un composant classé non critique entraînant la perte complète du système, par exemple) de prendre, bien que tardivement, des dispositions correctives pouvant déboucher sur une reprise de la classification au regard des premiers résultats d'essais ou a minima pour la maintenance future du logiciel.

Dans tous les cas, les actions induites d'un mauvais classement des composants logiciel en classes de criticité, ne doivent pas se limiter à un changement de classe, mais entraînent la reprise des activités nécessaires à la nouvelle classe (ajout de mécanismes de protection, tests supplémentaires pour améliorer la couverture de tests, ...).

7.2 VERIFICATION DES ACTIVITES DE CONSTRUCTION

Les techniques de vérification classiques à toute démarche d'assurance qualité (revue, inspection, lecture croisée, ...) sont, bien évidemment, des outils de vérification de l'ensemble du projet donc de la composante Sûreté de Fonctionnement. A ce niveau l'existence de listes de contrôle, ensemble de points à vérifier selon les caractéristiques d'un projet et selon les produits et les phases considérées, doit permettre une approche plus systématique dans la prise en compte des exigences du projet, et en particulier de la Sûreté de Fonctionnement, lors des actions classiques de vérification.

Si ces techniques de vérification ne sont pas systématisées pour le développement des logiciels, dans le cas du développement d'un logiciel à exigences de Sûreté de Fonctionnement, il est nécessaire, qu'a minima, soient menées :

- une revue de spécification ayant pour but principal de s'assurer de la validité des exigences de Sûreté de Fonctionnement du logiciel définies. Cette revue doit permettre, entre autres, de vérifier que la traçabilité entre les exigences de Sûreté de Fonctionnement du système, les choix de conception système et l'architecture du système et les exigences de Sûreté de Fonctionnement du logiciel sont assurées et que les résultats des AMDE et/ou des Arbres de Fautes sont satisfaisants,

  Cette étape est jugée très importante. En effet, le logiciel reste une technologie. Les fonctions qu'il réalise doivent toujours être maîtrisées par les gens du métier (du système). Le retour d'expérience montre que les erreurs de spécification représentent la majorité des anomalies résiduelles des logiciels.

- une revue de conception permettant de s'assurer que tous les choix liés au développement d'un logiciel répondant à ses caractéristiques de Sûreté de Fonctionnement sont adéquats et ont fait l'objet d'actions spécifiques de vérification (AMDE, Arbre de Fautes, ...) au cours de la conception. Lors de cette revue, il convient aussi de s'assurer que les contraintes liées aux phases suivantes (réalisation, codage, tests) sont correctement définies,

- une revue de codage pour vérifier que les contraintes de réalisation ont été respectées et que les contrôles définis dans le plan de Sûreté de Fonctionnement ont été effectués. Ces contrôles peuvent être de différentes natures : lecture de code, inspection, analyse statique, manuellement ou par l'utilisation d'outils de vérification de standard de programmation ou d'analyse statique. L'automatisation de la vérification de code est à privilégier, d'autant plus que les logiciels sont complexes et volumineux.

Les différentes revues doivent être réalisées par des pairs, dans la mesure du possible.

Outre les techniques classiques de vérification statiques qui naturellement vont permettre de vérifier certains aspects liés à la Sûreté de Fonctionnement, les principales techniques utilisées dans un contexte de projet industriel sont :

- AMDE, Arbres de Fautes (ces techniques de construction servent aussi à vérifier certains aspects de la construction de la Sûreté de Fonctionnement),
- Spécification et vérification comportementale du logiciel (graphe état-transition, diagramme d'état, diagramme de séquences, etc).

Pour les parties critiques du logiciel, il est recommandé, voire obligatoire selon les normes, de mettre en œuvre des méthodes formelles pour spécifier et/ou vérifier le logiciel. Ces méthodes se répartissent selon les catégories suivantes :

- Vérification de programme par analyse statique : cette approche s'utilise sur un logiciel déjà développé. Elle fonctionne par une exécution symbolique du code adaptée à ce que l'on veut vérifier sur celui-ci : par exemple, l'absence de fuite de mémoire, l'absence de débordement de tampon, etc. Certaines méthodes proposent même l'introduction d'assertions dans le code, mais les propriétés exprimables restent simples et souvent dans ce cas le code doit respecter des règles de codage strictes.
- Model-checking (vérification de modèles) : cette approche requiert l'expression du comportement du logiciel dans un formalisme donné. A partir du modèle résultant, tous les états du système sont vérifiés par rapport à des propriétés attendues. Ces propriétés sont le plus souvent des états indésirables du système : le model-checking permet alors de vérifier la non-atteignabilité de ces états par le système en fonction de ses données d'entrée. L'expression du comportement du système peut soit précéder soit suivre l'écriture du logiciel : dans les deux cas une relecture est nécessaire pour s'assurer de l'équivalence du logiciel et de sa modélisation dans le formalisme de model-checking. Les techniques de model-checking sont particulièrement adaptées à la vérification de systèmes à états finis, parallèles ou concurrents.
- Méthode basée sur la preuve : cette approche demande à être mise en place le plus tôt possible dans le cycle de développement logiciel, souvent juste à la suite de la phase de la spécification. En effet, le logiciel est exprimé dans un langage formel à partir duquel sera obtenu plus tard le logiciel final par génération de code. Les méthodes basées sur la preuve sont à la fois les plus difficiles mais aussi les plus puissantes en termes de possibilités de vérification de propriétés et de maîtrise du développement logiciel.

Ces techniques, déployées maintenant dans de nombreux secteurs, portent leurs fruits et montrent que les logiciels développés via ces approches formelles sont moins sujets à anomalies résiduelles.

Cependant, elles nécessitent des investissements tant en compétences qu'en outils. Elles s'appliquent à des domaines ou à des types de projet pour lesquels la description des besoins et le comportement peuvent être considérés comme déterministes ou proches de l'être. C'est pourquoi il est absolument nécessaire de trouver le bon compromis entre les exigences de Sûreté de Fonctionnement à atteindre, les coûts de développement, les coûts de démonstration et les coûts des évolutions successives des versions logicielles.

7.3 VERIFICATION DU RESPECT DES EXIGENCES DE SURETE DE FONCTIONNEMENT

Les activités de vérification de la Sûreté de Fonctionnement pendant les phases de tests du cycle de vie consistent essentiellement à renforcer les actions classiques de tests principalement sous deux angles :

- mise en œuvre de techniques de tests spécifiques au regard des exigences de Sûreté de Fonctionnement. Il s'agit de compléter les techniques de tests mises en œuvre lors de ces étapes par des techniques dédiées aux objectifs de Sûreté de Fonctionnement,
- systématisation et accroissement des exigences en matière de niveau de couverture de test, c'est à dire augmentation des cas de tests afin d'augmenter la proportion de logiciel activée en tests afin de réduire le risque d'occurrence de défaillance lors de l'utilisation du système.

7.3.1 Techniques couramment utilisées

Ces techniques peuvent être :

- tests de robustesse,
- tests aux limites, y compris les modes dégradés du système,
- tests hors limites permettant de piéger le comportement du logiciel dans des conditions de fonctionnement inhabituelles ou improbables,
- tests des mécanismes de Sûreté de Fonctionnement introduits dans le code.

Les essais spécifiques de Sûreté de Fonctionnement et de simulation pour démontrer la validité des actions en diminution des risques concernant tout particulièrement les barrières Safety, les tests, les alarmes et les procédures de reconfiguration doivent être prévus.

Ils peuvent s'appuyer sur des approches déterministes telles que :

- tests couvrant la vérification de l'ensemble des propriétés temporelles,
- tests réalisés en tenant compte du domaine de validité des entrées (+ couverture de branche),
- tests de vérification des zones de confinement,
- tests de non régression liés aux aspects Sûreté de Fonctionnement.

Une fois le logiciel validé, y compris sur le plan de la Sûreté de Fonctionnement, celui-ci est utilisé pour être intégré dans le système avant la validation de ce dernier, selon les phases du cycle de vie système défini. Les tests du système vont aussi compléter les tests du logiciel, celui-ci étant alors mis en œuvre dans un contexte plus proche du contexte opérationnel d'utilisation et non plus simulé, ce qui est le plus souvent le cas pour les tests logiciel.

Dans certains cas, des essais définis pour le test du logiciel peuvent être rejoués dans un contexte opérationnel vaste afin de s'assurer que les moyens mis en œuvre pour permettre la validation du logiciel étaient corrects et représentatifs. C'est en particulier le cas des tests de robustesse et aux limites dont l'efficacité est basée sur l'exécution du logiciel dans un contexte d'exploitation différent du contexte nominal (modes dégradés, environnement perturbé, ...).

7.3.2 Autres techniques

D'autres techniques peuvent être utilisées pour compléter les actions de vérification ou pour en évaluer l'efficacité.

C'est, par exemple, le cas de :

- l'injection de fautes qui vise à s'assurer :
    - de l'efficacité des jeux de tests : vérification de l'exhaustivité et de l'efficacité des tests par analyse du ratio nombre de fautes injectées détectées par les tests / nombre total de fautes injectées,
    - de la tolérance aux fautes : injection de fautes physiques ou des fautes logiciel (on peut s'aider par exemple des AMDE ou de l'Arbre de Fautes pour déterminer les fautes à injecter),

  Il est à noter que l'injection de fautes demande des moyens de tests spécifiques avec des possibilités d'observation fine du comportement du logiciel.
- l'utilisation de tests statistiques qui visent à couvrir le maximum de cas possibles du logiciel par l'utilisation de générateurs de scénarios de tests et l'automatisation forte des tests. Cette technique, si sur le plan théorique ne pose pas de difficulté est contrainte par la mise à disposition de générateurs de scénarios de tests éprouvés et applicables à tout type de logiciel, ce qui réduit fortement sa généralisation dans l'industrie.

De plus, les actions de vérification peuvent, elles aussi, faire l'objet d'une vérification, le plus souvent au travers d'une revue de fin de phase (pour chacune des phases de test) permettant de faire le bilan des anomalies détectées et des couvertures de tests obtenues et le cas échéant de définir les actions complémentaires à mener.

7.4 VERIFICATION DES PROCEDURES D'EXPLOITATION

La Sûreté de Fonctionnement d'un système et des logiciels qui le constituent, repose aussi sur des procédures d'utilisation et d'exploitation efficaces et correctement mises en œuvre.

Ces procédures doivent être définies en même temps que le système et les logiciels qu'elles concernent sont conçus. A ce titre, elles sont à prendre en compte, dans leur principe (la rédaction intervient plus tard et n'est a priori pas faite par les concepteurs du système mais plutôt par les exploitants) lors de la revue de conception.

Une fois le logiciel, puis le système, opérationnels il est nécessaire que ces procédures soient vérifiées et complètement déroulées avant la mise en service opérationnel du système.

7.5 PLANIFICATION DES ACTIONS DE VERIFICATION

La planification des actions de vérification de la Sûreté de Fonctionnement est une activité essentielle dans la démarche de Sûreté de Fonctionnement. Les activités de vérification en matière de Sûreté de Fonctionnement comme en général, peuvent être menées à l'infini. Il convient donc de bien choisir les actions à mener dans un souci constant de compromis entre Sûreté de Fonctionnement (et donc risque) et coût.

Les critères de choix et les actions retenues sont consignés dans le plan de Sûreté de Fonctionnement du logiciel.

8. ÉVALUATION DE LA SURETÉ DE FONCTIONNEMENT

L'évaluation de la Sûreté de Fonctionnement du logiciel repose à la fois sur :

- le niveau de Sûreté de Fonctionnement atteint,
- la conformité du processus de développement.

Cette évaluation n'est le plus souvent requise que face à des exigences Safety.

L'évaluation doit s'assurer que :

- l'ensemble du cycle de développement logiciel a été respecté,
- les activités V&V ont été correctement menées,
- les exigences de Sûreté de Fonctionnement du logiciel sont respectées,
- les résultats de test démontrent qu'il n'y a pas d'anomalies résiduelles contraires à la Safety,
- les éventuelles anomalies résiduelles connues sont maîtrisées.

Pour cela, l'entité en charge de l'évaluation du logiciel analyse l'organisation, les méthodes, la documentation, les analyses de Sûreté de Fonctionnement, les résultats des activités de vérification et de validation.

Les résultats de l'évaluation sont consignés dans un rapport d'évaluation.

L'entité en charge de l'évaluation doit être indépendante vis-à-vis des principaux acteurs du projet pour qu'elle puisse avoir autorité sur la décision de déploiement du logiciel.

Selon la norme utilisée et le niveau de Sûreté de Fonctionnement visé, ces principes d'indépendance sont plus ou moins renforcés.

Les entités en charge de l'évaluation peuvent être :

- un expert indépendant compétent en matière de sécurité fonctionnelle pour le domaine concerné, comme par exemple un ISA (Independent Safety Assessor),
- un organisme accrédité.

L'entité en charge de l'évaluation trace les éventuels écarts par rapport à l'application d'une norme ou d'un standard et conclue sur leur acceptabilité, il porte un jugement sur l'absence d'impact Safety des anomalies ou non-conformités du produit, non encore corrigées, ainsi que sur le niveau de Safety atteint et démontré pour le logiciel.

Selon les domaines d'activité, l'évaluation peut porter des appellations différentes, telles qu'appréciation ou certification.

9. ACTIVITES DE SURETE DE FONCTIONNEMENT ET ACTIVITES DE DEVELOPPEMENT

Les tableaux des pages suivantes présentent, en correspondance, pour chaque phase de développement, les activités principales et les activités de construction et de vérification de la Sûreté de Fonctionnement.

Phase du cycle de vie	Activités de construction de la Sûreté de Fonctionnement	Activités de vérification et de validation de la Sûreté de Fonctionnement
Spécification du système Définir l'ensemble des fonctionnalités, interfaces, contraintes, et exigences du système. Préparer le plan qualité et le plan de validation du système.	Recenser toutes les exigences liées à la Sûreté de Fonctionnement du système : A/ qualitatives : • listes d'événements redoutés, classification des défaillances • définition des modes dégradés et des conditions de passage entre modes, comportement dans chacun des modes, • type de fautes à considérer et stratégie de tolérances aux fautes (respect du critère de tolérance de défaillance unique, fail-op, fail-safe, fail-op/fail-safe, ...), • méthodes à employer et normes à respecter. B/ quantitatives : • durée d'exploitation, • performances.	Vérifier que l'ensemble des contraintes de type Sûreté de Fonctionnement issues des analyses de risques système étayées le cas échéant par des *AMDEC* ou des arbres de fautes ont toutes été prises en compte dans les spécifications. Vérifier la couverture des exigences Sûreté de Fonctionnement du système à l'occasion d'une revue de spécification système. Définir les principes de validation du système sous l'angle conformité aux exigences de Sûreté de Fonctionnement. Préparer la validation du niveau de Sûreté de Fonctionnement du système en exploitation. S'assurer de la mise en œuvre des tâches prévues au plan qualité. Cette activité s'exerce dans l'ensemble des phases du cycle de vie.
Conception du système Bâtir l'architecture du système (décomposition en sous-systèmes matériels et logiciels) Définition des besoins de chaque sous-système Préparation du plan d'intégration du système	Effectuer une analyse préliminaire de risque du système Identifier les fonctions et les composants à risques (*AMDE*, *Arbre de Fautes*, ...) Prendre en compte les nécessités d'isolation de parties à risques dans le choix de l'architecture.	Inclure dans le plan d'intégration les essais spécifiques permettant de s'assurer que les choix d'architecture ont été correctement mis en œuvre Vérifier que l'ensemble des actions en réduction de risques issues de l'analyse de risques sont toutes closes, pour ce qui concerne le niveau système.

Phase du cycle de vie	Activités de construction de la Sûreté de Fonctionnement	Activités de vérification et de validation de la Sûreté de Fonctionnement
Spécification du logiciel Décrire l'ensemble des fonctionnalités, interfaces, contraintes et exigences du logiciel, en correspondance avec les résultats de la conception du système Préparer le plan qualité et le plan de validation du logiciel	Définir les fonctions à risques les plus probables, issues soit des fonctions du système (fonctions opérationnelles), soit des choix de conception (fonctions de Sûreté de Fonctionnement) par l'utilisation d'analyses de type Analyse de Risques (*AMDE*, *AdF*) pour identifier les scénarios au niveau du logiciel qui peuvent amener aux événements redoutés systèmes identifiés dans les phases précédentes, et établir les actions en réduction de risques à mener (définition d'invariants Safety, modélisation du fonctionnement). Recenser l'ensemble des exigences de Sûreté de Fonctionnement applicables au logiciel, exprimées de manière cohérente avec celles du système.	Vérifier la couverture des exigences Sûreté de Fonctionnement du logiciel à l'occasion d'une revue de spécification du logiciel, en présence des gens du métier qui maîtrisent le fonctionnel attendu. Vérifier que les analyses *AMDE, AdF ou* évaluations prévisionnelles ont été réalisées correctement lors de la revue de spécification du logiciel Définir les principes de validation du logiciel sous l'angle conformité aux exigences de Sûreté de Fonctionnement (y compris les tests de robustesse et les exigences de couverture de test)
Conception du logiciel Bâtir l'architecture logiciel (découpage en tâches et en composants) Définir les interfaces entre tous les composants logiciels Définir les exigences de réalisation du logiciel Préparer le plan d'intégration du logiciel	Enrichir les résultats des analyses réalisées en phase de spécification du logiciel (analyse de risque, *AMDEC*, *Arbre de Fautes*, évaluations prévisionnelles, ...). Définir les principes d'architecture et de réalisation permettant de satisfaire les exigences du logiciel (*programmation N versions*, programmation défensive, définition des mécanismes de protection, ...). Identifier les contraintes d'exploitation, de diagnostic ou de reconfiguration héritées des choix de conception du logiciel.	Procéder à une revue de SdF de la conception du logiciel. Une telle revue peut s'appuyer sur les résultats d'analyse comme des *AMDEC*, *Arbre de Fautes*, *modélisation comportementale*, ... Définir les essais d'intégration du logiciel permettant de s'assurer que les choix de conception ont été respectés et qu'ils permettent bien de satisfaire aux exigences spécifiées
Conception détaillée et codage Concevoir et coder chacun des composants en respect des exigences assignées Définir les tests unitaires associés à chaque composant	Prendre en compte les règles de conception et de programmation retenues pour satisfaire aux exigences de Sûreté de Fonctionnement Concevoir et réaliser les mécanismes de Sûreté de Fonctionnement définis dans l'architecture	Vérifier le respect des règles de conception et de programmation (séparation, programmation défensive, règles de codage, ...) Vérifier que les mécanismes prévus ont été implantés Définir les tests permettant de s'assurer de l'efficacité des mécanismes et du respect des exigences associées à chaque composant (tests aux limites, tests de robustesse, ...).

Phase du cycle de vie	Activités de construction de la Sûreté de Fonctionnement	Activités de vérification et de validation de la Sûreté de Fonctionnement
Tests unitaires Exécuter l'ensemble des tests unitaires définis et vérifier que les résultats obtenus sont conformes aux résultats attendus.	Définir les procédures et cas de tests en correspondance avec les objectifs de test.	Exécuter les tests des mécanismes liés à la Sûreté de Fonctionnement introduits pour satisfaire aux exigences du logiciel Evaluer la robustesse des composants logiciels Tester l'efficacité de la tolérance aux fautes des composants. Vérifier la suffisance du niveau de couverture structurel obtenu pour les composants à risque. Initier le recueil des défaillances logicielles.
Tests d'intégration Exécuter l'ensemble des tests d'intégration définis et vérifier que les résultats obtenus sont conformes aux résultats attendus.	Définir les procédures et cas de tests en correspondance avec les objectifs de test.	Exécuter les tests des mécanismes inter-composants liés à la Sûreté de Fonctionnement introduits pour satisfaire aux exigences du logiciel et du système. Evaluer la robustesse des interfaces entre composants logiciels. Vérifier la suffisance du niveau de couverture de test structurel obtenu pour les parties à risques du logiciel. Poursuivre le recueil des défaillances logicielles et comparer aux Arbres de fautes, AMDE, évaluations prévisionnelles.

Phase du cycle de vie	Activités de construction de la Sûreté de Fonctionnement	Activités de vérification et de validation de la Sûreté de Fonctionnement
Intégration matériel logiciel Exécuter l'ensemble des tests d'intégration définis et vérifier que les résultats obtenus sont conformes aux résultats attendus.	Définir les procédures et cas de tests en correspondance avec les objectifs de test.	Exécuter les tests des mécanismes inter-composants liés à la Sûreté de Fonctionnement introduits pour satisfaire aux exigences du système. Evaluer la robustesse des interfaces entre composants matériels et logiciels. Vérifier la suffisance du niveau de couverture de test obtenu pour les parties à risques du système. Poursuivre le recueil des défaillances logicielles et comparer aux Arbres de fautes, AMDE, évaluations prévisionnelles.
Validation du logiciel Exécuter l'ensemble des tests de validation définis et vérifier que les résultats obtenus sont conformes aux résultats attendus. Mesurer le niveau de couverture fonctionnel obtenu	Définir les procédures et cas de tests en correspondance avec les objectifs de test, en mode nominal et en mode dégradé.	Exécuter les tests de robustesse du logiciel (incluant les cas de fonctionnement dégradés du logiciel). Tester l'efficacité des fonctions de Sûreté de Fonctionnement introduites et le respect des exigences de Sûreté de Fonctionnement du logiciel. Tester la robustesse et les performances du logiciel et le niveau de Sûreté de Fonctionnement atteint.
Validation du système Exécuter l'ensemble des tests de validation définis et vérifier que les résultats obtenus sont conformes aux résultats attendus. Mesurer le niveau de couverture fonctionnel obtenu.	Définir les procédures et cas de tests en correspondance avec les objectifs de test, en mode nominal et en mode dégradé.	Exécuter les tests de robustesse du système (incluant les cas de fonctionnement dégradés du système). Tester l'efficacité des fonctions de Sûreté de Fonctionnement introduites et le respect des exigences de Sûreté de Fonctionnement du système. Tester la robustesse du système et le niveau de Sûreté de Fonctionnement atteint. Poursuivre le recueil des défaillances logicielles et comparer aux Arbres de fautes, AMDE, évaluations prévisionnelles.

10. PLAN DE SÛRETE DE FONCTIONNEMENT LOGICIEL

10.1 POURQUOI UN PLAN

Les normes et standards utilisés dans tous les secteurs imposent que les activités de Sûreté de Fonctionnement menées pour le développement d'un logiciel soient décrites dans un plan.

Ce plan est nécessaire pour que l'ensemble des acteurs impliqués dans la Sûreté de Fonctionnement du logiciel ait connaissance de l'organisation, des activités à mener, des responsabilités, du phasage des activités et des résultats attendus, et pour que les responsables de la Sûreté de Fonctionnement au niveau du système, les futurs évaluateurs ou certificateurs, et globalement toutes les personnes ayant à en avoir la connaissance puissent avoir une vision du processus mis en œuvre pour répondre aux objectifs de Sûreté de Fonctionnement concernant le logiciel et puissent se prononcer suffisamment tôt en cas de difficultés identifiées.

10.2 EXEMPLE DE PLAN

Le plan de Sûreté de Fonctionnement d'un logiciel a pour objectif de présenter l'ensemble des dispositions prises par l'entité en charge de la Sûreté de Fonctionnement du projet pour atteindre le niveau de Sûreté de Fonctionnement demandé.

Analogue dans ses principes et dans sa structure à un Plan Qualité Logiciel, il peut être établi indépendamment de tout autre document, fusionné avec le Plan Qualité Logiciel du projet ou inséré dans un Plan de Sûreté de Fonctionnement général système.

La notion de plan type de plan de Sûreté de Fonctionnement décrite ci-après doit être comprise comme une liste de rubriques à traiter plus que comme une exigence forte d'un document séparé.

Certaines des rubriques sont similaires à celles que l'on peut trouver dans un plan de vérification, un plan qualité ou un plan de gestion de configuration. Le choix de répéter les informations ou de faire des renvois entre documents est de la responsabilité du rédacteur, conformément aux pratiques en vigueur dans l'entité dont il dépend.

Le plan de Sûreté de Fonctionnement doit présenter l'ensemble des dispositions organisationnelles et techniques à mettre en œuvre lors du projet pour atteindre le niveau de Sûreté de Fonctionnement visé et respecter les exigences identifiées.

Le plan type proposé est le suivant :

1. Introduction - Contexte

2. Documents de référence et documents applicables

3. Terminologie

4. Organisation

   L'organisation mise en œuvre pour les aspects Sûreté de Fonctionnement doit être présentée en faisant apparaître les relations fonctionnelles et hiérarchiques et les interfaces avec les autres acteurs ou organisations du projet (maîtrise d'œuvre, équipe projet, qualité, ...).

5. Rappel des exigences de Sûreté de Fonctionnement

   Si les objectifs de Sûreté de Fonctionnement du logiciel ne font pas l'objet d'un document précis, ils sont rappelés ici de manière détaillée.

   S'ils sont formalisés de manière satisfaisante dans un document, on rappellera simplement les objectifs principaux avec un renvoi vers ce ou ces documents pour les détails.

   De plus, une classification des logiciels selon leur criticité ou à défaut, une grille de classification et les principes associés, pourra compléter ce chapitre, dans le cas d'un plan de Sûreté de Fonctionnement s'appliquant au développement d'un ensemble de logiciels. Dans ce cas, chaque fois que nécessaire, les dispositions des chapitres suivants préciseront la classe de logiciels auxquels elles s'appliquent.

6. Principes et démarche de Sûreté de Fonctionnement mis en œuvre

   Il s'agit de présenter la démarche de Sûreté de Fonctionnement retenue pour le projet, avec les différentes articulations entre construction, vérification et évaluation.

7. Détail des activités de Sûreté de Fonctionnement Logiciel

   Classées par phases du cycle de développement, et en fonction du type d'activités (construction, vérification, évaluation), les activités de Sûreté de Fonctionnement sont décrites en précisant pour chacune :

   - l'objectif de l'activité et son positionnement par rapport au processus de développement choisi et les conditions de prise en compte des résultats ou documents,
   - les responsables de la réalisation et de la vérification de l'activité,
   - les éléments en entrée de l'activité,
   - les conditions éventuelles de déclenchement de l'activité,
   - les actions à réaliser,
   - les résultats attendus de l'activité,

- les critères de vérification des résultats de l'activité,
- les liens avec d'autres activités,
- les destinataires des résultats ou principes d'exploitation de ces résultats et de répercussion sur les documents du projet (conception, ...).

La gestion des risques et le suivi des priorités doivent guider la détermination des activités et être pris en compte dans le suivi des activités de Sûreté de Fonctionnement Logiciel.

8. Gestion de la documentation

Les fournitures attendues de l'ensemble des activités de Sûreté de Fonctionnement sont définies, avec précision de leur statut (livrable, interne, ...).

9. Méthodes et Outils

Les méthodes et outils à utiliser pour la mise en œuvre de ce plan sont présentés, en cohérence avec les activités de Sûreté de Fonctionnement citées dans le §7 du plan de Sûreté de Fonctionnement. Des informations complémentaires quant à la disponibilité de ces moyens ou aux formations nécessaires à ces moyens sont précisées.

10. Interface avec les autres composantes du système

Les liens entre Sûreté de Fonctionnement logiciel et Sûreté de Fonctionnement système seront précisés, au travers de l'impact des résultats sur les autres composantes du système et si ces activités font l'objet de plans et de dispositions différentes définis et mis en œuvre par des entités distinctes.

11. Planification des activités

Le planning ou les contraintes d'enchaînement des activités entre elles et avec les activités de développement et, plus généralement, avec le planning directeur du projet sont précisés.

12. Gestion des résultats des activités Sûreté de Fonctionnement

Les liens entre la gestion de configuration du projet et la gestion des résultats des activités de Sûreté de Fonctionnement sont précisés. Doivent notamment être précisés les principes de gestion de la validité de ces résultats par rapport aux versions du logiciel.

13. Suivi d'application du plan de Sûreté de Fonctionnement

Les principes, modalités et fréquences de suivi de mise en œuvre du plan de Sûreté de Fonctionnement sont précisés.

Les dispositions à prendre en cas de non application du plan ou de demande de dérogation sont aussi à définir tant sur le plan technique qu'organisationnel.

14. Bilan de Sûreté de Fonctionnement

Le rebouclage des résultats obtenus par rapport aux exigences spécifiées doit être prévu. La réalisation d'un bilan de la Sûreté de Fonctionnement permet de faire ce rebouclage.

Il doit permettre aussi la capitalisation de toutes les actions et décisions prises en cours de projet dans la démarche de construction et de vérification de la Sûreté de Fonctionnement du logiciel.

11. ELEMENTS COMPLEMENTAIRES

11.1. DETERMINATION DE CLASSES DE CRITICITE POUR LE LOGICIEL

11.1.1. Présentation

Un logiciel intégré dans un système (ou dans les sous-systèmes le composant) peut contribuer, de manière semblable au matériel, à des défaillances de ce système. Afin de limiter les effets de celles-ci, voire, dans certains cas, détecter ou tolérer certains défauts, des exigences liées à la Safety spécifient l'immunité désirée et les réponses du système aux conditions de panne. Ces exigences fixées au système (ou à ses sous-systèmes), induisent des exigences plus ou moins fortes sur le comportement du logiciel ou des logiciels correspondants.

Pour les systèmes complexes, comportant plusieurs composantes matérielles et logicielles (plusieurs machines ou calculateurs au sein du système ou plusieurs microprocesseurs au sein du même calculateur), il peut être intéressant de mettre en évidence des sous-ensembles de criticités différentes du point de vue de la Sûreté de Fonctionnement.

Les critères permettant de « garantir » un taux de défaillance d'un logiciel n'étant pas connus dans l'état de l'art actuel, les exigences de comportement du logiciel se traduiront le plus souvent par des classes de criticité du logiciel fixant la rigueur plus ou moins importante imposée aux travaux de réalisation du logiciel (tests, revues, analyses, documentation,...).

11.1.2. Intérêts d'une classification des logiciels

Hormis le cas des secteurs industriels (aéronautique, nucléaire, ...) où la classification du logiciel est imposée par le référentiel utilisé, les intérêts d'une telle classification, pour tout autre système, sont de plusieurs ordres :

- accroître la Sûreté de Fonctionnement du système (ou d'un sous-système) en intensifiant certaines activités (vérification notamment) sur certains composants logiciels,
- réduire globalement le coût d'un développement logiciel ou à effort égal, mieux répartir l'effort de développement, en modulant celui-ci en fonction de la criticité des composants logiciels,
- appliquer à bon escient certaines techniques de Sûreté de Fonctionnement (AMDE, preuve de programme, ...) sur certains logiciels.

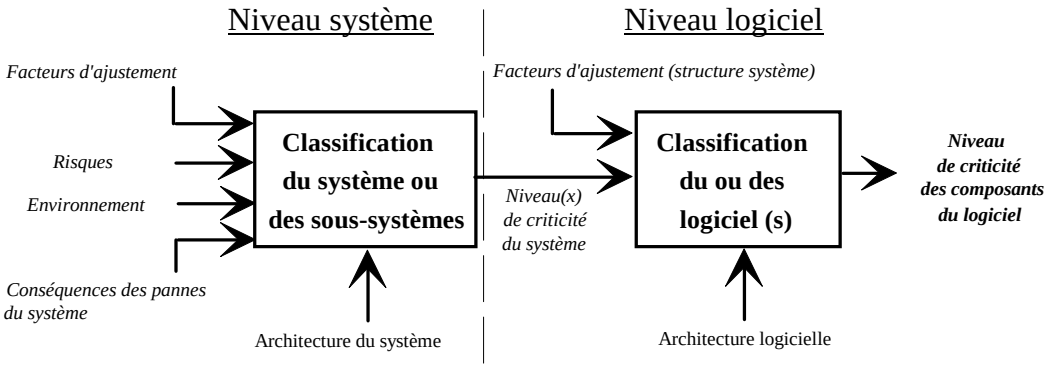
En contre partie, il faut bien savoir que cette classification n'a d'avantage que si l'on sait garantir la ségrégation des logiciels de criticités différentes. En effet, les logiciels les moins critiques ne doivent pas pouvoir polluer les parties de logiciels les plus critiques (par l'intermédiaire de données communes par exemple), sous peine d'anéantir les efforts de développement réalisés sur les logiciels les plus critiques.

En pratique, la mise en œuvre de ségrégation au sein d'un même microprocesseur s'avère en général très délicate et nécessite l'élaboration de dispositifs spécifiques (protections mémoires, ...). Elle est plus aisée lorsque la structure du système est répartie sur des machines différentes ou sur plusieurs microprocesseurs au sein du même calculateur.

Enfin, le compromis entre coût et Sûreté de Fonctionnement doit être également pris en considération, pour éviter de faire de la sur-spécification d'une part ou pour éviter la mise en péril économique d'un projet d'autre part.

11.1.3. Démarche de classification d'un logiciel

Le processus de classification pour fixer la criticité du logiciel comprend des aspects système et logiciel. Il est généralement composé d'étapes itératives à ces deux niveaux :



Classification du système : la structure du système étant définie ainsi que les conditions opérationnelles et environnementales, il s'agit d'identifier les risques de ce système (dans tous ses modes de fonctionnement), ainsi que les cas de pannes ou d'utilisations erronées du système et leurs conséquences.

Cette classification doit prendre en compte des facteurs d'ajustement tels que l'architecture du système, les redondances matérielles ou des restrictions d'utilisation éventuelles. Un niveau de criticité est attribué au système (ou aux différents sous-systèmes) en fonction de la catégorie de risque.

A noter que la classification des systèmes est abordée de manière différente selon les secteurs industriels et les systèmes considérés (civils, militaires, ...). Les critères de classification sont généralement basés sur la description du comportement du système en présence de défauts aboutissant à un classement par catégories d'événements (catastrophique, dangereux, majeur, mineur, sans effet par exemple dans le domaine aéronautique). Des probabilités d'occurrence d'une panne dangereuse du système peuvent être associées à ces catégories.

Classification du logiciel : les fonctionnalités du système assurées par du logiciel étant définies, il s'agit de classer le logiciel du système, ou de chacun des sous-systèmes, en fonction de la classification de niveau système.

De même que pour la classification du système, des facteurs d'ajustement sont à prendre en compte : l'architecture du système et des logiciels peut influer sur le niveau de criticité attribué au logiciel.

Le procédé de détermination en deux étapes est itératif, les décisions de conception de chaque niveau (système et logiciel) influant mutuellement sur les classifications retenues.

Par exemple, les logiciels à versions multiples dissimilaires (ou programmation N-versions), technique de conception qui consiste à réaliser deux ou plusieurs composants logiciels assurant la même fonction de manière à éviter certaines sources d'erreurs communes (introduction d'une hétérogénéité par programmation par des personnes différentes, utilisation de langages différents, ...), permettent de limiter l'impact des erreurs ou de détecter les défauts. Aucune règle précise n'existe cependant pour en déduire une réduction du niveau de criticité du logiciel et une analyse, souvent délicate, au cas par cas est nécessaire.

A noter que le développement d'un logiciel à un niveau de criticité donné n'implique généralement pas qu'on lui attribue un taux de défaillance. L'analyse de Sûreté de Fonctionnement ne peut alors donc pas utiliser des taux de fiabilité fondés sur la classe du logiciel comme on peut le faire pour les taux de panne du matériel.

11.2. ENVIRONNEMENT DE DEVELOPPEMENT ET SURETE DE FONCTIONNEMENT

11.2.1. Introduction

Dans certains domaines, l'utilisation d'outils fiables et de haut niveau constituant l'environnement de développement du logiciel est considérée comme un paramètre important pour la réalisation d'un logiciel à fortes contraintes de Sûreté de Fonctionnement. En effet, un compilateur peut par exemple introduire des erreurs en délivrant un code objet altéré ou un éditeur de lien peut masquer une erreur d'allocation mémoire.

Il faut alors notamment avoir des générateurs de code dans lesquels on puisse avoir confiance, c'est à dire soigneusement validés. Outre la validation du compilateur lui-même, encore faut-il que le processus de vérification tienne compte des caractéristiques particulières de ce compilateur et du langage de programmation. En particulier, si l'on introduit un nouveau compilateur, un nouvel éditeur de liens ou si l'on modifie les options du compilateur au cours du cycle de vie du logiciel, les tests ou analyses de couverture antérieurs risquent de ne plus être valables. Un compilateur sera considéré comme acceptable, pour un produit logiciel donné, lorsque la vérification du produit s'achèvera de manière satisfaisante.

Les contraintes de Sûreté de Fonctionnement se trouvent reportées, lorsque ces outils sont utilisés dans des conditions où ils éliminent, réduisent ou automatisent certaines activités du cycle de vie, sur le logiciel de l'outil au lieu du logiciel exécutable sur cible.

A noter que les démonstrations ou vérifications faites sur l'outil sont étroitement liées à l'environnement de développement et il convient de réexaminer la validité des démonstrations en cas de changement d'environnement (portage, changement de compilateur, ...)..

Il est à noter que si dans certains domaines comme l'aéronautique, le développement à partir d'outils nécessite au préalable que l'outil soit certifié, dans une optique de réduire ensuite certains tests sur le code généré, dans le domaine du nucléaire, si la manière dont l'outil a été lui-même validé est prise en compte, son utilisation n'a pas pour effet de réduire l'effort de test porté sur le code généré ; la certification de l'outil n'est pas un préalable car elle ne dispense pas des actions de tests prescrites.

11.2.2 Classification des contraintes Sûreté de Fonctionnement sur les outils de développement

Il faut mentionner que les considérations qui suivent ne s'appliquent qu'à des outils déterministes, c'est à dire des outils qui, fonctionnant dans le même environnement, génèrent les mêmes sorties pour les mêmes sollicitations en entrée.

Les contraintes de Sûreté de Fonctionnement étant reportées sur le logiciel de l'outil lui-même, un certain nombre d'activités se trouvent reportées sur le cycle de développement de l'outil (vérification des spécifications de l'outil, analyse de couverture des spécifications de l'outil par les tests, ...).

Le processus de développement du logiciel de l'outil est alors assez proche de celui du logiciel développé. Pour les logiciels les plus critiques d'un point de vue Sûreté de Fonctionnement, les critères exigés pour le développement des outils de développement qui peuvent introduire des erreurs, lorsque ceux-ci sont utilisés dans des conditions où ils éliminent, réduisent ou automatisent certaines activités du cycle de vie, sont de niveau équivalent à ceux exigés pour le développement du logiciel cible lui-même.

En fonction des secteurs, les outils sont classés selon plusieurs catégories.

11.2.2.1 Classification selon les principes aéronautiques

On peut distinguer deux catégories d'outils logiciels pour moduler les exigences concernant le développement de ces outils [cf. DO-178B] :

- **les outils qui peuvent introduire des erreurs dans le logiciel cible** c'est à dire les outils dont la sortie est directement utilisée dans la chaîne de production du code exécutable sur la cible tel un outil qui génère un code source à partir d'une description formelle de la conception détaillée. Ces outils sont principalement constitués par des outils de développement du logiciel.

  Ces outils, de par leurs fonctionnalités, peuvent introduire des erreurs dans le logiciel si la sortie générée par l'outil n'est pas vérifiée de la même manière que si l'on n'utilise pas l'outil.

- **les outils qui peuvent masquer ou ne pas détecter les erreurs du logiciel** c'est à dire les outils qui ne peuvent pas introduire d'erreurs dans le logiciel mais qui peuvent s'avérer incapables de détecter les erreurs existantes dans le logiciel tel un outil de test qui génère automatiquement des résultats de tests. Un résultat de test erroné peut éventuellement être présenté comme correct à l'utilisateur de l'outil et masquer une erreur du logiciel.

  Ces outils sont principalement constitués par des outils de vérification du logiciel.

11.2.2.1 Classification selon les principes industriels ou ferroviaires

Les normes IEC 61508 ou EN 50128 permettent de classer les outils selon trois catégories :

- **classe d'outils T1**, qui ne produisent pas de données de sortie susceptibles de contribuer directement ou indirectement au logiciel. T1 peut être par exemple un éditeur de texte ou un outil d'aide à la conception ou aux exigences démuni de toute capacité de génération automatique de code, ou un outil de contrôle de configuration.
- **classe d'outils T2**, qui prend en charge l'essai ou la vérification de la conception du logiciel, lorsque des erreurs internes à l'outil peuvent ne pas arriver à révéler des défauts mais sans créer directement des erreurs dans le logiciel. T2 peut être par exemple un générateur de logiciel de validation, un outil de mesure de la couverture des essais, un outil d'analyse statique.
- **classe d'outils T3**, qui produisent des données de sortie qui peuvent directement ou indirectement contribuer au logiciel. T3 peut être par exemple un compilateur qui incorpore un ensemble d'instructions exécutables dans le code exécutable.

L'utilisation des outils de classe T2 et T3 doit être justifiée au regard des exigences de Sûreté de Fonctionnement.

11.2.3 Impact de la Sûreté de Fonctionnement sur le choix de l'environnement de développement

Il convient de se préoccuper des contraintes de Sûreté de Fonctionnement portant sur l'environnement de développement lors de la préparation des activités de développement et du choix des méthodes et outils de développement du logiciel :

- choisir l'environnement de développement de manière à minimiser sa contribution aux risques potentiels pour le logiciel final,
- choisir l'utilisation d'outils ou de combinaisons d'outils de manière à avoir la certitude qu'une erreur introduite par un outil pourra être détectée par un autre,
- définir les activités de vérification du logiciel ou les règles de développement en tenant compte du niveau de Sûreté de Fonctionnement à atteindre de manière à minimiser les erreurs potentielles du logiciel liées à l'environnement de développement,
- examiner et préciser les effets des options utilisées si des caractéristiques optionnelles existent pour certains outils surtout lorsque l'outil génère directement une partie du produit logiciel (exemple options d'un compilateur qui permettent d'optimiser les performances du code objet).

11.2.4 Cas particulier des outils de génération automatique de code

La génération automatique de code est un moyen de minimiser l'introduction de fautes dans un logiciel lors de l'activité de codage contribuant ainsi à améliorer la Sûreté de Fonctionnement du logiciel.

L'utilisation de tels outils logiciels ne se limite cependant pas à l'automatisation de certaines activités du cycle de vie logiciel, mais sont aussi utilisés dans le but d'éliminer ou de réduire certaines activités du cycle de vie logiciel tel par exemple les tests unitaires si le code source a été généré automatiquement à partir de la conception détaillée. A noter dans ce dernier cas que l'utilisation d'un outil de codage ne dispense pas de valider la conception en elle-même.

11.2.5 Cas des outils de tests

L'environnement et les moyens de tests sont prépondérants dans la démonstration ou l'aide à la démonstration du niveau de Sûreté de Fonctionnement atteint par un logiciel ou par un système. En effet, plus encore que pour les aspects fonctionnels, la validation du comportement opérationnel en présence de défaillance interne ou de perturbations externes nécessite le plus souvent la mise en œuvre de moyens de simulation, de stimulation du système ou du logiciel dans son environnement dont l'efficacité est directement liée à la confiance que l'on pourra placer dans le produit final.

L'exécution de tests de cette nature peut par ailleurs nécessiter des scénarios de sollicitation fonctionnelle ou environnementale difficile à concevoir et/ou à mettre en œuvre d'où la nécessité d'outils conçus pour cela.

Comme de plus le test de systèmes ou de logiciels de ce type est le plus souvent répétitif car sujet à des versions successives de logiciel, il est aussi primordial d'inclure dans un tel outillage des moyens le plus automatisés possible permettant de rejouer les scénarios à chaque version, pour s'assurer la non régression.

11.3 METHODES FORMELLES

Les activités de vérification de construction d'un logiciel utilisé dans un contexte de Sûreté de Fonctionnement incluent a minima une revue de spécification, une revue de conception et une revue de codage. Il est recommandé, pour les parties critiques du logiciel, au moins d'étayer ces revues par l'utilisation de méthodes et d'outils aidant à consolider ces revues, au mieux de réaliser les phases de spécification/conception/codage via ces méthodes et outils, et ainsi faciliter d'autant leur revue.

Tout comme la relecture indépendante de code par un expert du langage, les outils formels ont pour but de proposer une analyse statique du code, c'est-à-dire permettre de prédire le comportement du code (selon des exigences établies à l'avance) sans avoir à l'exécuter. Toujours comme pour la relecture de code, cela impose que le langage utilisé pour décrire le logiciel laisse aussi peu de place que possible à l'ambiguïté. C'est là le premier intérêt des outils formels que nous pouvons citer : là où un relecteur pourra laisser passer une imprécision, les outils formels indiqueront un problème. Le premier intérêt de l'usage de méthodes formelles est une meilleure compréhension, et donc une meilleure maîtrise, du logiciel développé.

Les méthodes formelles se distinguent les unes des autres par le niveau de conception logicielle auquel elles peuvent intervenir et par les techniques de vérification sous-jacentes qu'elles utilisent. Si nous classons les classes de méthodes de la moins englobante à la plus englobante, nous avons :

- l'analyse statique par interprétation abstraite,
- la vérification de modèles (plus couramment appelée model-checking),
- les méthodes formelles.

Ce classement doit se comprendre en termes d'impacts sur le code du logiciel et ne doit pas être vu comme un classement compartimenté : en effet il existe des méthodes basées sur la preuve qui pourraient en théorie s'appliquer à des logiciels déjà écrits. Cependant ce ne serait pas réaliste en pratique au vu des modifications à appliquer au code pour pouvoir les utiliser.

Le classement proposé ci-avant est indicatif de l'impact sur le développement : plus la méthode est englobante, plus la phase de développement logiciel où il faudrait l'utiliser se situe en amont du cycle de développement. Par exemple, les méthodes d'analyse statique par interprétation abstraite peuvent être utilisées sur du code déjà écrit, alors que les méthodes basées sur la preuve demandent à être utilisées idéalement à partir de la phase de spécification.

Les techniques de vérification liées à ces méthodes suivent un classement similaire :

- l'interprétation abstraite,
- le model-checking,
- la preuve.

Ce classement est indicatif des possibilités d'expression des exigences logicielles : les techniques d'interprétation abstraite permettent l'expression d'exigences simples (pas de dépassement d'entiers, etc) tandis que la preuve permet de se rapprocher d'exigences de haut niveau (pas de collision entre deux trains, etc).

Notons que l'on parle de méthodes formelles lorsqu'elles incluent un processus couvrant plusieurs phases de développement (souvent de la spécification au codage) et de langages formels lorsqu'il ne s'agit que du codage.

11.4 SURETE DE FONCTIONNEMENT ET METRIQUES

Les métriques du logiciel sont basés sur un ensemble de mesures élémentaires le plus souvent relatives à la complexité du logiciel ou au processus de développement, à l'architecture et au profil d'utilisation.

Ces mesures, statiques ou dynamiques, s'appuient sur des outils dits de qualimétrie qui fournissent des résultats bruts qui doivent ensuite être comparés aux valeurs jugées acceptables par l'état de l'art par le Service Qualité de l'entreprise ou selon une norme applicable au contrat de développement.

Toutefois, actuellement, les seules mesures permettant d'évaluer le niveau de Sûreté de Fonctionnement atteint lors de la construction d'un logiciel et qui permettraient de prédire le comportement opérationnel du logiciel, sont les évaluations prévisionnelles quantitatives de fiabilité et les modèles de croissance traités au chapitre 8 du présent guide. Néanmoins, comme vu plus haut, il est aujourd'hui impossible de fixer une taux de défaillance au logiciel, sans que celui n'ait été démontré sans anomalies pendant une durée de fonctionnement incompatible avec les besoins d'exploitation. C'est pourquoi les notions de degré de confiance sont privilégiées dans les différents secteurs d'activité, par rapport aux notions de fiabilité intrinsèque.

Conscientes cependant que les métriques élémentaires telles que le nombre cyclomatique ou la complexité hiérarchique pour la complexité ou les taux de couverture de test ou le taux de commentaires pour les métriques représentatives du processus de développement, sont fortement corrélées au niveau de Sûreté de Fonctionnement d'un logiciel, les recommandations quant à l'utilisation de métriques pour le développement de logiciels critiques fixent des seuils d'acceptabilité des résultats des mesures plus stricts (diminution du nombre cyclomatique, augmentation du taux de couverture de tests à atteindre).

Cette difficulté de définir des seuils adaptés à chaque niveau d'exigences de Sûreté de Fonctionnement des logiciels, milite pour une utilisation de telles métriques par comparaison de différentes solutions de développement plutôt que dans l'absolu. Ainsi, les notions telles que les classes d'équivalence sont utilisées pour éviter d'avoir le phénomène d'explosion combinatoire.

Une autre difficulté réduisant l'utilisation de métriques précises réside pour certaines dans la pertinence et la difficulté pour interpréter les résultats et déterminer les actions que l'on peut faire pour améliorer les résultats.

Pour d'autres, la comparaison des évaluations de différentes solutions de développement permet d'en connaitre les « points durs » ou sensibles et ainsi focaliser l'attention du développeur sur telle ou telle fonction, processus de développement, …

L'annexe 3 propose un modèle d'évaluation prévisionnelle de fiabilité du logiciel.

11.5 EXPLOITATION ET MAINTENANCE

La construction ou la vérification d'un système ou d'un logiciel n'est pas une finalité. L'objectif visé est de pouvoir exploiter ledit logiciel en toute confiance et le cas échéant de pouvoir le faire évoluer tout en lui maintenant le niveau de confiance initial.

Il est dès lors indispensable que le logiciel qui va être exploité ait été conçu de manière à pouvoir être correctement exploité et corrélativement que toutes les informations nécessaires à une exploitation correcte soient déterminées et transmises à l'utilisateur.

Plus encore, la construction de la Sûreté de Fonctionnement, et sa vérification, doivent intégrer explicitement les procédures d'exploitation et de maintenance du logiciel et du système.

Le caractère tolérable ou non d'un risque est souvent assorti des conditions d'utilisation dans lesquelles ce risque peut se manifester. Cela signifie que l'on ne peut totalement dissocier la conception du système ou du logiciel de la définition des procédures associées.

Dans de nombreux cas, on jugera de l'acceptabilité d'un risque au travers des contraintes d'exploitation ou de maintenance qu'il génère pour être toléré. Quand ces contraintes sont jugées trop fortes, on aura recours à l'introduction dans le système ou dans le logiciel de mécanismes de Sûreté de Fonctionnement tels que ceux décrits au chapitre 6.2.

Toute action de vérification de la Sûreté de Fonctionnement (revue de Safety par exemple) doit donc inclure la dimension exploitation et maintenance.

Sur le plan de la maintenance, il convient toutefois de distinguer deux aspects :

- la remise en état de marche suite à une panne d'un composant ou le redémarrage à l'identique d'un logiciel,
- la validité des procédures de maintenance a un impact direct sur la Sûreté de Fonctionnement essentiellement sous deux angles : maintien d'un état sûr de fonctionnement suite à intervention et efficacité des procédures en vue de minimiser les durées d'intervention et donc d'accroître la disponibilité du système,
- l'introduction dans le logiciel de corrections pour éliminer des défauts logiciel.

Les procédures de modifications doivent nécessairement inclure la reprise (a minima vérification que les résultats demeurent valides après corrections) des analyses de Sûreté de Fonctionnement pour que la nouvelle version ainsi générée ne régresse pas quant à son niveau de Sûreté de Fonctionnement.

La gestion en configuration des résultats d'analyses et de vérification de Sûreté de Fonctionnement est alors un élément important dans ces procédures.

11.6 SÛRETÉ DE FONCTIONNEMENT ET RÉUTILISATION

11.6.1 Intérêt général de la réutilisation

Lors de l'élaboration de l'architecture d'un logiciel, il est tentant de s'appuyer sur la réutilisation de briques existantes, dont les fonctionnalités répondent en totalité ou en partie au nouveau besoin, et dont le comportement a déjà fait l'objet de vérifications. Cette approche peut permettre d'envisager des économies substantielles tant durant les activités de réalisation que durant les activités de vérification du nouveau logiciel.

Sous l'angle de la Sûreté de Fonctionnement, il peut paraître rassurant de s'appuyer sur des composants présentant un historique en service conséquent, garant a priori d'un fonctionnement exempt d'erreur. C'est cet a priori qu'il convient de traiter ici.

11.6.2 Maîtrise et limites de la réutilisation

Pour être réutilisé, un composant logiciel doit être complètement caractérisé selon différentes facettes, dont certaines sont relativement aisées à établir :

- fonctionnalités,
- interfaces,
- performances,
- …

Et d'autres paraissent plus difficiles à décrire et à quantifier :

- qualité,
- Sûreté de Fonctionnement,
- …

Rappelons de plus que la Sûreté de Fonctionnement d'un logiciel n'est pas intrinsèque à ce logiciel, mais résulte bien de l'adéquation des caractéristiques de ce logiciel aux exigences de Sûreté de Fonctionnement qui lui sont allouées par le niveau système comme résultat d'une démarche d'analyse. Il en résulte qu'un composant logiciel "sûr de fonctionnement" dans un contexte d'utilisation donné ne le sera pas forcément dans un autre contexte.

Il convient donc de renforcer la caractérisation du composant logiciel à réutiliser en intégrant les considérations propres à la Sûreté de Fonctionnement :

- les menaces prises en compte,
- les profils de sollicitation du composant (plage d'utilisation des paramètres, fréquences d'activation des fonctions, …),
- les comportements tolérés sur erreur (reprises à chaud, …),
- les effets de bord (accès à des ressources partagées, …),
- …

Cette caractérisation doit permettre de s'assurer de l'adéquation du comportement du composant logiciel avec sa nouvelle utilisation, y compris en ce qui concerne les exigences de Sûreté de Fonctionnement.

Si l'on dispose d'un historique en service, celui-ci doit être analysé selon ces caractéristiques afin de déterminer sa représentativité vis-à-vis d'un nouvel emploi du composant. Il doit être accompagné de tous les éléments de gestion de configuration nécessaires pour démontrer son applicabilité à la version du composant réutilisé.

La confiance que l'on peut accorder à un composant logiciel résulte habituellement de la complémentarité entre :

- la maîtrise de ses exigences,
- la maîtrise de son processus de réalisation,
- les vérifications sur le produit final.

Si la caractérisation du composant logiciel telle que décrite précédemment permet de satisfaire au premier point, les tests et dans une certaine mesure l'historique en service permettent de satisfaire au troisième, il reste, pour établir pleinement la confiance dans le composant, à disposer des données issues de son processus de production. Ceci constitue souvent un obstacle majeur à la réutilisation.

11.6.3 Contraintes normatives

La réutilisation de composants existants (logiciels ou composants COTS) peut être contrainte par des considérations propres au domaine d'application et aux normes applicables (exigences sur le niveau de développement, considérations sur le code mort ou désactivé, prise en compte des composants COTS dans les tests de validation, …). La prise en compte de ces contraintes est à évaluer très précisément car elle risque, en particulier pour les logiciels de forte criticité, de réduire à néant les gains escomptés de la réutilisation.

11.6.4 Conclusion sur la réutilisation

La réutilisation d'un composant logiciel nécessite au minimum la connaissance détaillée du comportement de ce composant dans le cadre des contraintes et des profils de sollicitation propres à son nouvel environnement d'utilisation.

Un historique en service peut fournir un certain niveau de confiance s'il est représentatif de la nouvelle utilisation du composant. Il faut le compléter par un ensemble de tests suffisant pour assurer la couverture de l'ensemble des profils de sollicitations, en particulier en ce qui concerne la robustesse et les cas limites qui ont généralement été peu mis en œuvre durant l'usage opérationnel du composant.

Pour un logiciel de criticité élevée, la maîtrise de son processus de développement doit être démontrée avec le niveau requis.

12. CONCLUSION

Le présent guide décrit une démarche et des méthodes permettant de construire la Sûreté de Fonctionnement des logiciels, mais ne constitue pas un jeu de recettes toutes faites. Il nécessite d'y consacrer un effort de mise en œuvre pour d'une part prendre en compte l'existant de l'organisation, les méthodes et outils de développement de l'entité en charge des logiciels, et d'autre part le contexte du projet (type de projet, exigences normatives et réglementaires, contraintes technico-économiques, …).

La Sûreté de Fonctionnement des logiciels s'inscrit dans une démarche plus large de Sûreté de Fonctionnement système globale : cohérence des exigences sur le logiciel et des événements redoutés du logiciel par rapport aux événements redoutés et aux exigences de Sûreté de Fonctionnement du système.

La Sûreté de Fonctionnement des logiciels, comme les composants microprogrammés, est un domaine de l'ingénierie système en évolution permanente qui n'a pas encore atteint un niveau de maturité suffisant et comparable à la Sûreté de Fonctionnement des matériels.

La Sûreté de Fonctionnement des logiciels se construit dès les phases amont de son cycle de vie (spécification), avec une répartition des efforts, adaptée au contexte du projet, sur les différentes phases : on n'a souvent pas le temps de faire mais on a toujours le temps de refaire pour corriger une situation non satisfaisante. L'objectif d'une démarche de Sûreté de Fonctionnement des logiciels est de faire bien du premier coup et de pouvoir le montrer (traçabilité) en fournissant les éléments de jugement propres à donner le niveau de confiance approprié. Il n'y a pas de durcissement Sûreté de Fonctionnement des logiciels. La Sûreté de Fonctionnement d'un logiciel ne suit pas de loi physique en matière de défaillance : il ne tombe pas en panne mais a des comportements inattendus, non prévus, c'est ce qui en fait la complexité.

La mise en œuvre de tout un ensemble de méthodes et de techniques pour éviter d'introduire des fautes et pour détecter les fautes introduites peut conduire à une multiplication d'actions génératrices d'efforts et donc de coûts. Il peut être nécessaire d'essayer d'optimiser les charges consacrées au développement de logiciel sûr de fonctionnement, en analysant plus finement, en fonction du contexte des projets, l'efficacité des différentes techniques et méthodes, à partir d'informations issues du retour d'expérience de projets antérieurs ou à partir de méthodes prévisionnelles.

ANNEXE 1. TERMINOLOGIE

Avertissement : Les définitions données dans cette annexe proviennent soit de documents divers (normes, rapport de thèse, articles, projets, …) ou ont été constituées pour la compréhension du corps du présent guide. Elles ne se veulent en aucun cas normatives.

AEEL (Analyse de l'Effet des Erreurs de Logiciels) : Analyse visant à déterminer, suivant une démarche inductive analogue à celle de l'*AMDE*(C), quelles sont la nature et la criticité des conséquences des défaillances du logiciel. Ce type d'analyse ne traite que d'une erreur à la fois dont on trace l'effet.

AMDE(C) (Analyse des Modes de Défaillances, de leur effet et de leur criticité) : Analyse visant à déterminer, en suivant une démarche inductive, quelle est la nature des conséquences des défaillances (et de leur criticité). Ce type d'analyse ne traite que d'une défaillance à la fois.

Analyse de risques : analyse permettant d'identifier les points critiques et les critères de Sûreté de Fonctionnement, dans les divers éléments d'un système. Cette analyse peut se faire aux différentes étapes de la construction d'un système.

Analyse dynamique : processus consistant à évaluer un système ou un composant sur la base de son comportement pendant l'exécution.

Analyse fonctionnelle : Décomposition, par affinement successif, des fonctions en fonctions plus simples mettant en évidence la circulation des données échangées, sans prescrire la manière dont les fonctions seront réalisées.

Analyse statique : Processus d'évaluation d'un système ou d'un composant basé sur sa forme, sa structure, son contenu, sa documentation.

Analyse statique par interprétation abstraite (à ne pas confondre avec l'analyse statique) : Méthode formelle basée sur la transformation de programmes. Le logiciel est transformé, via l'application de règles mathématiques, sous une forme permettant de vérifier certaines propriétés du programme sans avoir à l'exécuter (par exemple, dépassement des indices de tableau, pointeurs non initialisés, etc).

Arbre de Fautes : technique déductive d'analyse et de représentation permettant de rechercher les combinaisons d'événements qui entraînent la réalisation d'un événement unique, considéré comme indésirable.

Bloc de recouvrement : bloc fonctionnel permettant en cas d'erreur constatée lors d'un traitement de revenir à un état précédent exempt d'erreur puis de lui substituer un bloc de traitement, fonctionnellement identique mais potentiellement ne conduisant pas aux mêmes erreurs.

Compensation d'erreur : action permettant de transformer un état erroné en un état exempt d'erreur, à partir d'éléments redondants eux-mêmes exempts d'erreur.

Complexité hiérarchique : Mesure de la complexité de l'architecture d'un logiciel fondée sur son graphe d'appel.

Confinement d'erreur : action permettant une reprise sur un état sain du logiciel, à partir d'une restauration de contexte exempt d'erreur.

COTS : Commercial Off The Shelf. Composant ou logiciel « sur étagère » réutilisable.

Critère de défaillance unique : critère appliqué à un système tel que celui-ci soit capable de remplir sa propre tâche Safety lorsqu'il est soumis à un seul défaut.

Criticité : Classification des défaillances selon la sévérité (à définir) des modes de défaillances d'un système, pondérée par leurs probabilités d'apparition.

Défaillance : Événement survenant lorsque le service délivré n'est plus conforme au service attendu.

Défaillance de cause commune : Défaillances dépendantes de plusieurs dispositifs ou composants qui sont dans l'incapacité de remplir leurs fonctions et qui ont pour origine une même cause.

Défaut, faute : Cause adjugée de l'erreur (pour un logiciel, la faute est liée aux activités de développement).

Déverminage : mise au point d'un dispositif ou d'un programme à l'aide d'un autre dispositif ou d'un autre programme (dévermineur) qui permet la détection et la localisation des défaillances et des erreurs. L'arrêté ministériel du 30 décembre 1983 préconise les termes suivants : « débogage » plutôt que « déverminage » et « débogueur » plutôt que « dévermineur ».

Disponibilité : Aptitude d'un système à se trouver dans un état de bon fonctionnement.

Diversification fonctionnelle : Approche pour le développement d'un système destinée à fournir des services identiques via des fonctions (spécification, conception et réalisation) différentes.

Diversité : existence de différents moyens de réaliser une fonction requise (par exemple autres principes physiques, autres manières de résoudre la même tâche.

Élimination des fautes : Méthodes et techniques destinées à réduire la présence (en nombre et en sévérité) des fautes. Les tests en font partie.

Erreur : L'erreur est un état intermédiaire provenant de l'activation d'un défaut et pouvant mener à une défaillance.

Événement redouté (indésirable) : événement (défaillance) d'un système ou d'un logiciel (état, donnée, comportement) qui ne doit pas être atteint pendant son utilisation.

Fail op/fail safe : caractéristique d'un système exprimant les exigences quant à son état en présence de défaillance ; fail-op : le système reste opérationnel après première défaillance, fail-safe : le système est mis dans un état sûr après occurrence de la première défaillance, fail-op/fail-safe : le système reste opérationnel après la première occurrence de défaillance et est mis dans un état sûr après la seconde si aucune action de maintenance n'a permis de remettre le système dans son état initial. Cet état fail-op/fail-safe traduit généralement le fait qu'un système est tolérant aux fautes.

Fiabilité : aptitude d'un programme à accomplir sans défaillance l'ensemble des fonctions spécifiées dans un document de référence, dans un environnement opérationnel de référence pour une durée d'utilisation donnée.

Fonction de sécurité : fonction spécifique du système de Sûreté de Fonctionnement ou des autres systèmes importants pour la Safety (voir sécurité-innocuité)

Graphe d'appels : Graphe représentant l'architecture d'un logiciel en montrant les relations d'appel entre les procédures le constituant. Un nœud du graphe représente une procédure, un arc représente un appel.

Graphe de contrôle : Graphe représentant la structure d'un composant logiciel et défini en associant un nœud du graphe à chaque instruction, et un arc à chaque transition possible. Le graphe peut être simplifié en associant un seul nœud à un ensemble d'instructions séquentielles ne comportant qu'un seul chemin, un seul point d'entrée et un seul point de sortie.

Injection de fautes : voir tests de mutation.

Invariants Safety : propriété d'un sous-ensemble de données que le logiciel doit impérativement respecter et préserver tout au long de son exécution. Tout non respect d'un invariant Safety conduit à considérer le logiciel comme défaillant.

Maintenabilité : aptitude d'un système à être maintenu ou rétabli dans son état de bon fonctionnement.

Marche dégradée : réduction par étapes des fonctions du système en réponse à une panne détectée alors que les fonctions essentielles sont maintenues.

Matrice de traçabilité : voir traçabilité.

Méthode formelle : cadre méthodologique, basé sur les mathématiques, pour la spécification, le développement et la vérification des logiciels. Les méthodes formelles peuvent être utilisées pour :

- la spécification : le comportement du logiciel peut être décrit a priori avec une notation bien définie et non ambigüe soutenue par une sémantique précise. Selon la méthode formelle utilisée pour la spécification, il est possible de s'assurer que cette description vérifie les propriétés de sûreté requises,
- le développement : le logiciel peut être développé et affiné pas à pas, en démontrant que chaque nouvelle étape de développement garantit les propriétés (souvent de sûreté) assurées par la précédente ainsi que de nouvelles propriétés,
- la vérification : le logiciel peut être vérifié vis-à-vis de propriétés voulues pour celui-ci. Selon le type de méthode formelle utilisé en phase de spécification et/ou de développement, la vérification pourra faire appel à des techniques d'analyse statique par interprétation abstraite, de model-checking ou de preuve.

Mode commun : effet simultané sur plusieurs parties distinctes du système (défaillance de mode commun : défaillance de cause commune se manifestant par le même mode de défaillance).

Mode de défaillance : Effet manifesté d'une faute.

Model checking : Technique formelle de vérification visant à générer tous les états possibles d'un système afin de les comparer vis-à-vis d'une formule logique donnée. Souvent, cette formule logique correspond à un état indésirable du système (il s'agit de la négation d'une propriété de sûreté) : comme l'exhaustivité des états du système est vérifiable, alors la formule logique est garantie vraie pour le système. Dans le cas contraire, l'approche permet d'exhiber un contre-exemple qui servira pour analyses.

Modèles de croissance de fiabilité : modèles mathématiques qui par analyse des données de défaillance et de leur répartition temporelle, essayent de prédire le niveau de fiabilité obtenu et fournissent une indication quant à la croissance ou non de la fiabilité du logiciel considéré et partant sur sa convergence.

Nombre cyclomatique : Dans la théorie des graphes, nombre de chemins indépendants constituants une base pour un graphe fortement connexe. Il vaut 'Nombre d'arcs - Nombre de nœuds + 1'. Appliqué au graphe de contrôle d'un composant logiciel, il caractérise sa complexité.

Preuve formelle : Méthode de raisonnement basé sur une logique mathématique et un système de raisonnement déductif. Exemple : le calcul des prédicats pour la logique du premier ordre avec la théorie des ensembles.

Prévention des erreurs : méthodes et techniques destinées à empêcher l'occurrence ou l'introduction de fautes.

Processeur codé : processeur utilisant des techniques de codage permettant la détection de certaines défaillances pouvant survenir dans la chaîne de traitement, depuis la compilation jusqu'à l'exécution. Il permet aussi de détecter des altérations de données dues à des défaillances matérielles.

Programmation défensive : techniques de programmation incluant des mécanismes de détection d'erreurs et de confinement d'erreur.

Programmation en N versions : développement de multiples versions d'un logiciel (ou d'une partie de logiciel) à partir d'une même spécification. L'exécution simultanée des différentes versions permet par synchronisation et comparaison (voteurs) de détecter une version défaillante et d'utiliser dans ce cas les résultats des autres versions.

Reconfiguration : action consistant à modifier la structure du système ou du logiciel, de telle sorte que les composants non défaillants permettent de délivrer un service acceptable bien que dégradé, après l'occurrence d'une défaillance ayant conduit le système a ne plus délivrer le service attendu. La reconfiguration peut impliquer l'abandon de certaines tâches et la ré-allocation d'autres tâches aux composants restant.

Recouvrement d'erreur : Forme du traitement d'erreur où un état exempt d'erreur est substitué à l'état erroné.

Redondance (passive, tiède, chaude) : présence d'éléments ou systèmes (identiques ou différents) de rechange de sorte que l'un d'entre eux puisse remplir la fonction requise quel que soit l'état de fonctionnement ou malgré la défaillance d'un autre :

- redondance chaude (ou active) : tous les éléments sont actifs en même temps (traitent tous les messages d'entrée de manière concomitante afin de garder leurs états internes étroitement synchronisés), en cas de défaillance d'un élément celui ci est ignoré puis remplacé ou reconfiguré,
- redondance froide (ou passive) : en cas de défaillance de l'élément actif, un élément passif est configuré dans un état lui permettant de reprendre les actions en cours et de poursuivre le service,
- redondance tiède : tous les éléments traitent tous les messages d'entrée, mais un seul fournit les messages de sortie. En cas de défaillance de l'élément actif, un élément redondant est activé et ses sorties sont prises en compte.

Réseau de Pétri : Méthode de description de systèmes complexes permettant de vérifier le comportement dynamique et les interactions entre composants.

Revue : action qualité intervenant notamment en fin de phase de développement, ayant pour objectif de faire, par des personnes autres que l'équipe projet, l'analyse et le bilan des résultats et des actions réalisées au cours de la phase considérée et de vérifier la disponibilité des moyens nécessaires à la phase suivante. Une revue est formalisée par un compte rendu et débouche sur une sanction : autorisation de passage à la phase suivante (avec ou sans réserve), nécessité de reprendre les travaux et prévision d'une nouvelle revue. Pour certaines phases (spécifications, ...) la présence du client est fortement recommandée.

Sécurité-confidentialité (Security en anglais) : aptitude à prévenir les accès ou les modifications non autorisés des données et/ou des programmes.

Sécurité-innocuité (Safety en anglais) : aptitude à demeurer sûr en cas de défaillance ou en présence de perturbations externes.

Ségrégation : séparation physique et/ou logique entre composants permettant d'implémenter efficacement des techniques de diversification.

SIL : Safety Integrity Level – Niveau discret d'intégrité de la sécurité d'une fonction, noté de SIL0 (non sécuritaire) à SIL4 (critique). Plus le niveau SIL est élevé, plus les exigences Safety sont élevées. Le niveau SIL ne s'applique qu'aux fonctions. Il ne peut pas s'appliquer aux équipements, organisations ou facteurs humains.

Sûreté de Fonctionnement : Propriété qui permet aux utilisateurs d'un système de placer une confiance justifiée dans le service qu'il leur délivre.

Taux de commentaire : Rapport entre le volume de commentaires et le volume total d'un code source, les lignes de code étant celles contenant des instructions (compilables ou exploitées par le compilateur telles que directives de compilation, inclusion de fichiers, …) ou des données (définition, déclaration, …).

Tests de mutation : technique de tests permettant d'évaluer l'efficacité des jeux d'essais par l'introduction d'erreurs dans le code (mutant) et le pourcentage des erreurs détectées par les jeux d'essais. On considère alors que le ratio entre les mutants détectés et le nombre de mutants introduits est le même que celui entre les autres erreurs détectées par les jeux d'essais et le nombre d'erreurs totales contenues dans le code avant introduction des mutants. Cette technique est aussi appelée « injection de fautes ».

Tests de non régression : tests effectués suite à une modification pour vérifier que les parties et données non modifiées ont conservé un comportement conforme à celui attendu et qui était le leur avant l'introduction de la modification.

Tests de robustesse : technique de tests visant à vérifier la robustesse du produit testé dans des conditions de fonctionnement non nominales (forte sollicitation, conditions aux limites, conditions de défaillances, …) ou vis-à-vis d'agressions externes.

Tests statistiques : technique de tests basée sur la définition aléatoire d'un grand nombre de jeux de tests en vue d'obtenir un taux de couverture suffisant.

Tolérance aux fautes : Méthodes et techniques destinées à fournir un service conforme à la spécification en dépit des fautes.

Traçabilité : Moyen d'expliciter l'association entre des éléments (produits ou processus) exploités, produits ou mis en œuvre lors du cycle de vie d'un système ou d'un logiciel. Elle est habituellement utilisée :

- en "traçabilité verticale" : pour relier les exigences décrivant les besoins et les contraintes initiaux avec la description du système répondant à ce besoin, et avec les éléments constitutifs de l'implémentation,

- en "traçabilité horizontale" : pour relier les éléments de description du système avec les tests destinés à s'assurer de leur implémentation correcte.

Elle se formalise le plus souvent sous forme de matrices faisant apparaître les liens de traçabilité entre des éléments de description du logiciel. Elle est utilisée :

- pour naviguer entre ces éléments,
- pour démontrer la complétude de la prise en compte d'un ensemble d'éléments.
  ex : matrice de traçabilité des exigences : ensemble de tables qui permettent d'assurer que les exigences du système et/ou les exigences du logiciel sont bien reprises dans les documents de spécification du logiciel, de conception, de test, de vérification, …

ANNEXE 2. FICHES NORMES

Fiches normes jointes :

- CEI 60880 (Nucléaire),
- CEI 62138 (Nucléaire),
- CEI 61508 globale et partie logiciel (Tous secteurs),
- CEI 62061 (Machines),
- CEI 61511 (Industrie de transformation),
- DO 178 (Aéronautique civile),
- EN 50128 (Ferroviaire),
- ECSS-Q80 (Spatial),
- ISO 26262 (Automobile),
- CEI 62304 (Médical).

Référence *:* CEI 60880	***Origine*** *:* Commission Electrotechnique Internationale	***Date*** : 2006
Titre *:*	Centrales nucléaires de puissance – Instrumentation et contrôle-commande importants pour la sûreté – Aspects logiciels des systèmes programmés réalisant des fonctions de catégorie A	

BUT

Définir les principes et exigences relatives au logiciel des systèmes de Sûreté des centrales nucléaires pour les rendre cohérentes avec les normes appropriées traitant du matériel et de l'intégration du système.

DOMAINE D'APPLICATION PRIVILEGIE

Nucléaire (et par extension énergie) - Internationale

CONTENU

Cette Norme est applicable aux logiciels de hautes fiabilités des systèmes d'instrumentation et de contrôle-commande (I&C) programmés des centrales nucléaires de puissance, réalisant des fonctions de catégorie A telle que définie par la CEI 61226.

Cette norme fournit un ensemble de prescriptions pour chaque étape de l'élaboration du logiciel : spécification, conception, développement, vérification, qualification et mise en œuvre, ainsi que la documentation accompagnant chaque étape.

Des directives et informations complémentaires sur la manière de satisfaire aux prescriptions sont données en annexe.

INTERETS

- Description des étapes concises et abordant l'ensemble des tâches (construction, vérification).
- Recommandations relativement précises dans les annexes.
- Aspects tests bien détaillés.
- Aborde l'étape d'exploitation, ce qui est assez rare dans les normes.
- Traite les aspects logiciels relatifs à la défense contre les défaillances de cause commune

INCONVENIENTS

- Limitée aux fonctions de catégories A.
- Limitée aux aspects logiciels.
- Pas d'exigences sur les méthodes
- Ne traite pas des aspects organisation.
- Norme interprétable

POINTS FORTS D'UN POINT DE VUE SURETE DE FONCTIONNEMENT

S'adresse aux logiciels à hautes exigences de Safety.
Traite des logiciels en langages généraliste et en langage application
Définit des règles précises sur la pratique en matière de développement de logiciels critiques.
Constitue une référence internationale dans le domaine.

ETAT ACTUEL

2ième édition de la norme qui :

Assure la cohérence avec les nouvelles recommandations de l'AIEA

Tiens compte des nouvelles normes publiées (IEC 61513, IEC 61226, IEC 62138, IEC 60987)

Intègre les recommandations de la CEI 60880-2 de 2000

Norme Bilingue (français/anglais)

Référence *:* CEI 62138	***Origine*** *:* Commission Electrotechnique Internationale	***Date*** : 2004
Titre *:*	Centrales nucléaires de puissance –Instrumentation et contrôle-commande importants pour la sûreté – Aspects logiciels des systèmes informatisés réalisant des fonctions de catégorie B ou C	

BUT

Définir les principes et exigences relatives au logiciel des systèmes de Sûreté des centrales nucléaires pour les rendre cohérentes avec les normes appropriées traitant du matériel et de l'intégration du système.

DOMAINE D'APPLICATION PRIVILEGIE

Nucléaire – Internationale

CONTENU

Cette Norme est applicable aux logiciels des systèmes d'instrumentation et de contrôle-commande (I&C) programmés des centrales nucléaires de puissance, réalisant des fonctions de catégorie B ou C telle que définie par la CEI 61226.

Cette norme fournit un ensemble de prescriptions pour chaque étape de l'élaboration du logiciel : spécification, conception, développement, vérification, et mise en œuvre, ainsi que la documentation accompagnant chaque étape.

INTERETS

- Description des étapes concises et abordant l'ensemble des tâches (construction, vérification)
- Séparation claire B et C
- Aspects tests bien détaillés
- Aborde l'étape d'exploitation, ce qui est assez rare dans les normes

INCONVENIENTS

- Limitée aux aspects logiciels
- Pas d'exigences sur les méthodes
- Ne traite pas des aspects organisation
- Norme interprétable

POINTS FORTS D'UN POINT DE VUE SURETE DE FONCTIONNEMENT

Traite des logiciels en langages généraliste et en langage application

Constitue une référence internationale dans le domaine.

ETAT ACTUEL

Norme Bilingue français/anglais

1ière édition

Référence *:* CEI 61508 ***Origine*** *:* CEI ***Titre*** *:* Sûreté fonctionnelle : systèmes relatifs à la sûreté (7 parties)	***Date*** *:* 2011

BUT

Fournir une approche générale de toutes les activités liées au cycle de vie de Sûreté de Fonctionnement de systèmes destinés à exécuter des fonctions de Sûreté de Fonctionnement lorsqu'ils comportent des dispositifs électriques/électroniques/électroniques programmables (E/E/PES). Il constitue un cadre à l'établissement de normes d'application spécifiques à chaque secteur.

DOMAINE D'APPLICATION PRIVILEGIE

Cadre générique tous secteurs - International

CONTENU

Le document se compose de sept parties :

- Partie 1 : prescriptions générales : objectifs de Sûreté de Fonctionnement, type de systèmes, ...
- Partie 2 : règles organisationnelles et techniques applicables aux systèmes électriques/ électroniques/ électroniques programmables (E/E/PES),
- Partie 3 : règles organisationnelles et techniques applicables au logiciel,
- Partie 4 : définitions et terminologie,
- Partie 5 : lignes directrices pour l'application de la partie 1,
- Partie 6 : lignes directrices pour l'application des parties 2 et 3, comprenant un guide de modulation des techniques recommandées selon l'objectif de Sûreté de Fonctionnement,
- Partie 7 : description succincte des techniques recommandées et bibliographie.

Il spécifie les prescriptions pour obtenir la Sûreté fonctionnelle des systèmes de sûreté et la réduction des risques externes. Il utilise un cycle de vie de Sûreté pour toutes les activités (depuis la conception initiale jusqu'au déclassement, en passant par la création, l'installation, la mise en service et l'entretien) nécessaires à la réalisation et l'exploitation d'un système.

L'approche adoptée, basée sur la prévention des risques, permet la détermination du niveau d'intégrité selon une classification en 4 niveaux. Ces niveaux d'intégrité de Sécurité (SIL) définissent les cibles d'intégrité des fonctions de Sûreté. Des cibles "numériques", pour la mesure des défaillances des E/E/PES, sont fixées en relation avec chaque niveau d'intégrité.

INTERETS	**INCONVENIENTS**
• Modulation des exigences en fonction du niveau d'intégrité de Sûreté des fonctions • Généricité de la norme, chaque secteur industriel devant décliner ses exigences propres • Est basé sur la création d'un document de Spécification des Exigences de Sûreté	• Les exigences sont relativement élevées et d'aspect plutôt théorique qu'industriel • Concerne principalement la Sûreté des personnes mais peut également être applicable pour évaluer la protection des équipements, des produits et de l'environnement

POINTS FORTS D'UN POINT DE VUE SURETE DE FONCTIONNEMENT

Présente un panorama assez complet des techniques Sûreté de Fonctionnement.

ETAT ACTUEL

La première version date de 1997. La version 2011 introduit des problématiques nouvelles telles que l'utilisation des FPGA/ASIC, CPLD ; la nécessité de formaliser la conformité du produit développé par un manuel de sécurité, …

Référence *:* CEI 61508 ***Origine*** *:* CEI ***Titre*** *:* Sûreté fonctionnelle : systèmes relatifs à la sûreté Partie 3 : Prescriptions concernant les logiciels	***Date*** *:* 2011

BUT

Etablir des prescriptions concernant les procédés et activités du cycle de vie s'appliquant à tout logiciel destiné à exécuter des fonctions de Sûreté au sein d'un système comprenant des dispositifs électriques/ électroniques/ électroniques programmables (E/E/PES).

DOMAINE D'APPLICATION PRIVILEGIE

Cadre générique tous secteurs - International

CONTENU

Cette partie de la norme est exclusivement consacrée au logiciel et fournit des prescriptions destinées à éviter et contrôler les défauts et défaillances du logiciel sur toutes les activités. L'application des mesures et des techniques est graduée selon le niveau d'intégrité de sécurité du logiciel (SIL) en 4 niveaux. Les prescriptions concernent tous les logiciels : système d'exploitation, logiciel du système, logiciel des réseaux de communication, les outils d'assistance et le logiciel applicatif.

Ces prescriptions concernent plus spécifiquement :

- le cycle de vie de Sûreté de Fonctionnement du logiciel,
- la spécification des prescriptions concernant la Sûreté de Fonctionnement du logiciel,
- la planification de la validation,
- la conception et le développement du logiciel,
- l'intégration (matériel/logiciel),
- l'utilisation en fonctionnement et la maintenance,
- la validation de la Sûreté de Fonctionnement du logiciel,
- la modification du logiciel,
- la vérification du logiciel.

Il est basé sur l'élaboration d'une spécification des prescriptions concernant la Sûreté de Fonctionnement du logiciel et d'une spécification des prescriptions sur les fonctions de Sûreté du logiciel et sur l'intégrité du logiciel pour chaque système E/E/PES relatif à la Sûreté de Fonctionnement.

INTERETS	**INCONVENIENTS**
• Modulation des exigences en fonction du niveau d'intégrité de Sûreté des fonctions • Généricité des exigences	• Les exigences sont relativement élevées et d'aspect plutôt théorique qu'industriel, des guides d'application sectoriels sont nécessaires • L'utilisation directe

POINTS FORTS D'UN POINT DE VUE SURETE DE FONCTIONNEMENT

Présente un panorama assez complet des techniques SdF appliquées au logiciel..

ETAT ACTUEL

La première version date de 1997. La version 2011 introduit des problématiques nouvelles telles que la nécessité de formaliser la conformité du logiciel développé par un manuel de sécurité logiciel, la notion de capabilité logiciel ; les logiciels configurables, la qualification des outils, les approches objet…

Référence *:* CEI 62061	***Origine*** *:* Commission Electrotechnique Internationale	***Date*** : Janvier 2005
Titre *:*	Sécurité des machines – Sécurité fonctionnelle des systèmes de commande électriques/ électroniques/ électroniques programmables relatifs à la sécurité	

BUT

Définir les principes et exigences spécifiques au secteur des machines pour les systèmes de commande électriques relatifs à la sécurité.

DOMAINE D'APPLICATION PRIVILEGIE

Industrie

CONTENU

Cette Norme donne une méthodologie et des exigences pour :

- assigner le niveau d'intégrité de la sécurité prescrit pour chaque fonction de commande relative à la sécurité devant être réalisée par les SRECS (Safety-Related Electrical Control System),
- permettre la conception des SRECS appropriés aux fonctions de commande assignées,
- intégrer et valider les SRECS.

Cette norme fournit un ensemble de prescriptions pour chaque étape de l'élaboration du système : spécification, conception, développement, vérification, et mise en œuvre, ainsi que la documentation accompagnant chaque étape.

INTERETS

- Modulation des exigences en fonction du niveau d'intégrité de Sécurité des fonctions
- Simplification des calculs liés aux taux de défaillance par rapport à la norme CEI 61508
- Intégration de langages automates

INCONVENIENTS

- Ne traite pas des aspects organisation.
- Ne couvre pas les systèmes de très haute criticité (équivalent SIL4 de la norme CEI 61508).
- Le secteur des machines est couvert aussi par la norme ISO 13849 ; les liens entre ces deux normes sont complexes.

POINTS FORTS D'UN POINT DE VUE SURETE DE FONCTIONNEMENT

Cette norme étant non limitée aux aspects logiciels, elle permet de couvrir l'ensemble des exigences de sécurité au système développé.

ETAT ACTUEL

Version officielle

Référence *:* CEI 61511 ***Origine*** *:* Commission Electrotechnique Internationale ***Titre*** *:* Sécurité fonctionnelle – Systèmes instrumentés de sécurité pour le secteur des industries de transformation	***Date*** : Janvier 2003

BUT

Définir les exigences relatives aux spécifications, à la conception, à l'installation, à l'exploitation et à l'entretien d'un système instrumenté de sécurité, de telle manière qu'il puisse être mise en œuvre en toute confiance, et ainsi établir et/ou maintenir les processus dans un état de sécurité convenable.

DOMAINE D'APPLICATION PRIVILEGIE

Industrie – Process de transformation

CONTENU

Cette Norme est structurée en 3 parties :

- Partie 1 – Cadre, définitions, exigences pour le système, le matériel et le logiciel,
- Partie 2 – Lignes directrices pour l'application de la partie 1,
- Partie 3 – Conseils pour la détermination des niveaux exigés d'intégrité de sécurité.

Cette norme permet, sur la base d'un niveau de SIL déterminé, de :

- Identifier les exigences fonctionnelles et les exigences concernant l'intégrité de sécurité relatives aux fonctions instrumentées de sécurité.
- Spécifier les exigences relatives à l'architecture du système et à la configuration du matériel, au logiciel d'application et à l'intégration du système.
- Développer le logiciel d'application en conformité avec les exigences de sécurité.

Cette norme prend en compte toutes les phases de vie de sécurité, depuis le concept initial, en passant par la conception, la mise en œuvre, l'exploitation et la maintenance jusqu'au déclassement.

INTERETS

- Modulation des exigences en fonction du niveau d'intégrité de Sécurité des fonctions
- Simplification d'ensemble pour des systèmes instrumentés de sécurité de complexité moyenne
- Intégration de langages automates

INCONVENIENTS

- Ne traite pas des aspects organisation.
- Les renvois vers la norme CEI 61508 sont nombreux et ne facilitent pas la vision d'ensemble.
- Selon le langage automate choisi, l'application de cette norme ou de la CEI 61508-3 est à moduler.

POINTS FORTS D'UN POINT DE VUE SURETE DE FONCTIONNEMENT

Cette norme étant non limitée aux aspects logiciels, elle permet de couvrir l'ensemble des exigences de sécurité au système développé à base d'automates.

ETAT ACTUEL

Mise à jour en cours.

Référence *:* DO-178 C / ED-12C	***Origine*** *:* RTCA/EUROCAE	***Date*** *:* 2011
Titre *:* Considérations sur le logiciel en vue de la certification des systèmes et équipements de bord		

BUT

Fournir des recommandations pour la réalisation de logiciels destinés à des systèmes et équipements embarqués à bord des aéronefs civils, logiciels qui doivent remplir une fonction définie avec, en ce qui concerne la Sûreté de Fonctionnement, un degré de confiance en accord avec les exigences de navigabilité.

DOMAINE D'APPLICATION PRIVILEGIE

Aéronautique civile - International - Aspects logiciels d'une certification de navigabilité

CONTENU

Le document couvre tous les processus du cycle de vie logiciel et notamment : spécification, conception, codage, intégration, vérification, assurance qualité, gestion de configuration. Il présente les objectifs de ces processus, les activités pour satisfaire ces objectifs et les preuves permettant de montrer que les objectifs ont été atteints.

Les recommandations sont formulées en fonction du niveau logiciel qui est fondé sur la contribution du logiciel à d'éventuelles conditions de panne, déterminée par l'analyse de Sûreté de Fonctionnement du système. Le niveau logiciel engendre des exigences qui varient avec la catégorie de panne et donc avec la criticité des fonctions réalisées par le système.

Cinq catégories de logiciel sont définies (A, B, C, D, E en ordre décroissant d'exigences) qui déterminent :

- les exigences spécifiées,
- les niveaux de vérification associés à chacune des étapes de développement,
- les critères d'arrêt des tests,
- la formalisation de la documentation requise.

INTERETS

- Modulation des exigences en fonction de la criticité des fonctions permettant d'optimiser le coût du développement en fonction de la criticité des fonctions
- Document récent
- Traite de manière précise les aspects vérification et critères d'arrêt des tests du logiciel
- Application possible à d'autres domaines que l'aéronautique (pas de référence à d'autres normes spécifiques à l'aéronautique)

INCONVENIENTS

- Ne présente pas de méthodologie pour mener les activités permettant d'atteindre les objectifs fixés, (ce n'est donc pas un guide de développement),
- Ne traite pas du cycle de vie des systèmes (analyse de Sûreté de Fonctionnement des systèmes et validation),
- Les aspects documentaires sont assez peu détaillés (pas de plans types en particulier),
- Ne traite pas de l'organisation du développement et de la répartition des responsabilités.

POINTS FORTS D'UN POINT DE VUE SURETE DE FONCTIONNEMENT

- Présente des recommandations pour des logiciels de criticité très élevée (niveau A et B notamment),
- Aborde des sujets tels que la qualification des outils, la réutilisation de logiciel,
- Constitue une référence internationale dans le domaine.

Cette version est complétée par 4 fascicules normatifs complémentaires concernant :

- DO 330 : Considérations pour la Qualification des outils logiciels
- DO 331 : Développement et vérification à base de modèles
- DO 332 : Technologie Orientée Objet et Technique associées

DO 333 : Méthodes formelles

ETAT ACTUEL

Le document est publié officiellement et existe en versions française et anglaise.

Référence : ECSS-Q80A ***Origine*** : European Cooperation for Space Standardization ***Titre*** : Space Product Assurance : Software Product Assurance	***Date*** : 10 octobre 2003

BUT

Définir un ensemble d'exigences d'Assurance Produit pour le développement et la maintenance des logiciels des systèmes spatiaux. Ce standard sert de base pour l'écriture des exigences d'Assurance Produit Logiciel.

DOMAINE D'APPLICATION PRIVILEGIE

Spatial – Européen

CONTENU

Les exigences contenues dans ce document concernent la gestion et l'organisation de l'assurance qualité du logiciel, les activités et processus du cycle de vie du logiciel et la qualité des produits logiciels (incluant à la fois le code et les produits le concernant comme la documentation et les données de tests).

A chaque exigence est associée la sortie correspondante attendue. Certaines exigences portent un label « activités d'assurance » car elles peuvent être réalisées par une fonction indépendante d'Assurance Produit Logiciel.

Ce standard comporte des exigences sur la Sûreté de Fonctionnement du logiciel qui portent sur les points suivants :

- analyse fonctionnelle de niveau système pour identifier les modules logiciels critiques,
- attribution de niveau de criticité aux composants logiciels,
- choix de conception pour réduire le nombre de composants critiques,
- dispositions spécifiques pour assurer la Sûreté de Fonctionnement des modules critiques,
- isolation des logiciels critiques par rapport aux non critiques.

INTERETS	**INCONVENIENTS**
• permet de faire le lien entre la SdF système et la SdF logiciel • permet de faire le lien entre la SdF logiciel et l'Assurance Qualité logiciel • fournit des lignes directrices sans contraindre les méthodes ou techniques à employer	• essentiellement consacré à l'Assurance Qualité logiciel

POINTS FORTS D'UN POINT DE VUE SURETE DE FONCTIONNEMENT

Les exigences en matière de Sûreté de Fonctionnement logiciel sont pragmatiques.

ETAT ACTUEL

Version officielle depuis 2003.

Référence : ISO 26262 ***Origine*** : ISO ***Titre*** : Véhicules routiers — Sécurité fonctionnelle	***Date*** : Décembre 2011

BUT

Déclinaison de la CEI .61508 au domaine de l'automobile, la norme définit les exigences pour le développement de nouvelles fonctions ou la modification de fonctions existantes ayant à satisfaire à des contraintes de sécurité.

Elle couvre le management de la safety, les phases de conception système, matériel et logiciel mais aussi la production de série et l'exploitation.

DOMAINE D'APPLICATION PRIVILEGIE

Automobile : véhicule routier à 4 roues de moins de 3T5

CONTENU

Cette norme présente :

- le cycle de vie « sécurité » pour toute la vie du produit y compris la production de série, et la répartition des activités dans ce cycle,
- le modèle de défaillance établi, basé sur la prise en compte des situations de conduite pouvant entrainer un accident en présence de défaillance du véhicule et sur l'analyse de la sévérité des conséquences, de l'exposition au danger selon la fréquence d'utilisation de la fonction pour laquelle les défaillances sont examinées et de la capacité du conducteur a maîtriser cette situation pour éviter le danger,
- la méthode de détermination de la classification (Automotive Safety Integrity Levels, ASILs : 4 niveaux du plus faible au plus fort de A à D),
- la définition d'exigences à satisfaire en fonction du niveau ASIL dans le but d'un risque résiduel acceptable,
- les exigences de validation et de « confirmation ». les mesures de confirmation (revues et audits)à réaliser suivant niveau ASIL et le degré d'indépendance des acteurs défini.
- les principes de décomposition possibles du niveau ASIL sur des éléments indépendants (deux systèmes satisfaisant à un niveau ASIL B peuvent fournir une fonction ASIL D à condition de prendre en compte certaines précautions définies dans le document,
- les attendus au regard de chaque étape du cycle de vie du logiciel.

INTERETS	**INCONVENIENTS**
• donne une vision complète technique et management • fournit un cadre assez précise et structuré pour les phases de réalisation matérielle et logicielle	• allègement des exigences de version en version surtout pour la partie logiciel • calculs liés aux défaillances aléatoires assez complexes

POINTS FORTS D'UN POINT DE VUE SURETE DE FONCTIONNEMENT

Vision globale permettant une appréciation de tous les impacts potentiels en fonction du niveau de safety vis ».

ETAT ACTUEL

La version officielle (en anglais) est sortie officiellement.

Référence : CEI 62304 **Origine** : Commission Electrotechnique Internationale ***Titre*** : Logiciels de dispositifs médicaux – Processus du cycle de vie du logiciel	***Date*** : 2006

BUT

Description des méthodes de développement de logiciels du domaine médical en continuité de la norme CEI 601-1-4 : Appareils électro médicaux. Partie 1 Règles Générales pour la sécurité. 4. Norme collatérale : Systèmes électro médicaux programmables.

Elle couvre uniquement le cycle de vie en fonction des niveaux de sécurité identifiés via les analyses de risques définies dans la CEI – 60601-1-4.

DOMAINE D'APPLICATION PRIVILEGIE

Logiciels du domaine médical

CONTENU

Cette norme présente :
La norme est basée sur l'analyse du logiciel en termes de contribution à des situations dangereuses pour aboutir à une classification de sécurité (3 niveaux A à C) et la définition de différents processus à mettre en œuvre pour la maîtrise de ces logiciels, selon leur niveaux de criticité :

- processus de gestion des risques pour définir les niveaux de sécurité à viser
- processus de développement
- processus de maintenance,
- processus de gestion des configurations
- processus de résolution de problèmes

Les principes de modulations des exigences, sont basés sur l'utilisation de termes différents :

- «doit» signifie qu'une exigence donnée est obligatoire pour être conforme à la présente norme
- «il convient de – est recommandé» signifie qu'une exigence donnée est recommandée mais n'est pas obligatoire pour être conforme à la présente norme
- «peut – est admis» est utilisé pour décrire une manière autorisée pour être conforme à une prescription donnée
- «établir» signifie définir, documenter et mettre en oeuvre; et
- Lorsque la présente norme utilise l'expression «si nécessaire» ou «le cas échéant», conjointement à un processus, une activité, une tâche ou un résultat exigé, cela signifie que le fabricant doit utiliser le processus, l'activité, la tâche ou le résultat et dans le cas contraire il doit justifier sa décision par écrit.

INTERETS	**INCONVENIENTS**
• vision assez classique du cycle de vie logiciel pour des logiciels critiques	• n'aborde pas réellement les approches SdF logiciel

POINTS FORTS D'UN POINT DE VUE SURETE DE FONCTIONNEMENT

La démarche est cohérente avec les principes applicables aux autres normes médicales et de par les processus, les activités, les techniques abordées, la norme reste globalement réaliste, même si pour le niveau le plus critique, les exigences sont assez élevées au regard des pratiques habituelles dans le domaine médical

ETAT ACTUEL

Edition publiée. Traduction NF EN 62304 en octobre 2006

ANNEXE 3. METHODES ET TECHNIQUES

Fiches méthodes jointes

- AEEL
- Arbres de Fautes
- Programmation diversifiée (N_versions), Blocs de recouvrement, Diversification du logiciel
- Modèles de croissance de fiabilité, Relevé de défaillances, Relevé de données de suivi
- Modèles d'évaluation prévisionnelle
- Processeur codé
- Techniques de modélisation :
  - Réseaux de Petri
  - State charts
  - Simulation
  - Modélisation fonctionnelle et dysfonctionnelle
  - SysML / UML
- Familles de techniques formelles :
  - Analyse statique par interprétation abstraite
  - Model-checking
  - Preuve formelle
  - Méthode formelle
- Langages formels
  - Langage LUSTRE
  - Langage B

AEEL
(Analyse des Effets des Erreurs du Logiciel)

BUT

L'AEEL a pour but d'étudier (identification des modes vraisemblables de défaillance, causes possibles pour chaque mode, effet sur le système final) et de maîtriser les risques de défaillance d'un produit logiciel (définir et mettre en place des actions de prévention ou de correction).

PHASES PREFERENTIELLES D'UTILISATION

L'AEEL est prioritairement une méthode d'aide à la conception du logiciel pour les systèmes dont la Sûreté de Fonctionnement est une composante principale.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Pour réaliser une AEEL, il est nécessaire de bien connaître le logiciel à analyser. Aucun outil spécifique n'est nécessaire.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

Il est important de définir le niveau de détail auquel on descend dans l'arborescence du logiciel pour effectuer l'analyse. L'analyse est extrêmement efficace lorsqu'elle est centrée sur les composants logiciels pouvant être à l'origine de défaillance de l'ensemble du système. Elle peut éventuellement permettre d'optimiser les vérifications en différenciant les niveaux de tests réalisés en fonction de la criticité des composants logiciels.

La mise en œuvre de l'AEEL peut être délicate pour des systèmes complexes à nombre de composants élevé. Dans ce cas, il sera préférable de procéder par des analyses à plusieurs niveaux (identification des parties du système à fiabiliser, restriction de l'analyse à certaines fonctions et à certaines défaillances du logiciel comme le blocage des tâches par exemple).

Cette analyse a ses limites : elle est mal adaptée pour représenter des événements dynamiques et conduit à des redondances si les événements redoutés ne sont pas suffisamment indépendants.

METHODES ET TECHNIQUES DE REALISATION

L'AEEL est réalisée à l'aide de tableaux habituellement présentés sous forme de colonnes contenant les informations, pour chaque composant logiciel élémentaire étudié : mode de défaillance, causes de défaillance (possible ou non), effets de la défaillance, moyens de détection des défaillances.

ELEMENTS DE COÛTS

Directement dépendant des objectifs de l'analyse et du niveau de détail (nombre de composants logiciel analysé).

REFERENCES BIBLIOGRAPHIQUES

- Techniques d'analyse de la fiabilité des systèmes / Procédures d'analyse des modes de défaillance et de leurs effets - CEI 812 - 1985
- Fiche de synthèse ISdF – 1994
- Sûreté de Fonctionnement des systèmes industriels - A. Villemeur -1988 - Eyrolles Paris

METHODE DE L'ARBRE DE FAUTES
(aussi appelée "arbre de défaillances" ou "arbre de causes")

BUT

L'analyse de la Sûreté de Fonctionnement d'un système par une technique du type "arbre de fautes" a pour objet de rechercher, sur la base d'un événement redouté, les combinaisons d'événements qui conduisent à l'apparition de cet événement. De nature qualitative, cette analyse permet d'identifier assez rapidement tous les facteurs de risques sur l'environnement d'exécution d'un logiciel (matériel, logiciel, voire humain).

PHASES PREFERENTIELLES D'UTILISATION

Cette analyse est à conduire en phases de spécification et/ou de conception du logiciel.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Les principes de base de l'analyse sont simples et sa mise en œuvre ne nécessite pas d'outil spécifique. La réussite de ce type d'analyse repose sur une bonne connaissance du système.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

Contrairement à d'autres techniques ayant pour but d'identifier les défaillances du logiciel, cette technique repose sur l'analyse des défaillances potentielles du logiciel (quelle qu'en soit l'origine effective) et d'en analyser les conséquences. Ceci permet d'identifier rapidement les parties du logiciel contribuant aux événements redoutés pour le système.

Il convient de définir le niveau de la conception jusqu'où l'on souhaite mener l'analyse sous peine de rendre l'ampleur de l'analyse assez lourde. Seules les branches de l'arbre correspondant à des défaillances logicielles seront détaillées jusqu'au point où elles pourront être affectées à un composant logiciel (fonction, module : à définir en fonction de la profondeur de l'analyse).

Cette analyse a ses limites : elle est mal adaptée pour représenter des événements dynamiques et conduit à des redondances si les événements redoutés ne sont pas suffisamment indépendants.

METHODES ET TECHNIQUES DE REALISATION

L'analyse s'insère en général dans une analyse de Sûreté de Fonctionnement du système plus vaste et notamment par une phase d'analyse préliminaire des risques permettant d'identifier les événements redoutés pour le système. Cette analyse peut ensuite être complétée, pour les composants logiciels critiques, par une analyse du type AMDE.

Sur la base d'une liste d'événements redoutés (les plus indépendants entre eux), la méthode consiste à décomposer chaque événement redouté (racine de l'arbre) en niveaux successifs d'événements reliés par des opérateurs logiques (ET, OU, ...). Un symbolisme précis sera choisi permettant de représenter ces opérateurs. Certaines techniques seront ensuite utilisées sur la base de cet arbre afin d'identifier les points faibles du logiciel (réduction de l'arbre en coupes minimales, ...).

ELEMENTS DE COÛTS

Proportionnels au nombre d'événements redoutés analysé et à la profondeur de l'analyse.

REFERENCES BIBLIOGRAPHIQUES

- Sûreté de Fonctionnement des systèmes industriels - A. Villemeur -1988 - Eyrolles Paris

DIVERSIFICATION DU LOGICIEL

Programmation en N-versions, Blocs de recouvrement, logiciels dissimilaires

BUT

La diversification du logiciel a pour but de tolérer les fautes du logiciel. Un logiciel diversifié est constitué d'au moins deux variantes (ou versions) d'un logiciel conçues de façon indépendante et d'un décideur destiné à fournir un résultat exempt d'erreur à partir de l'exécution de ces variantes.

PHASES PREFERENTIELLES D'UTILISATION

L'utilisation de cette technique doit être prévue dès le début du développement du logiciel.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Autant d'équipes de développement que de variantes.

Une architecture matérielle supportant l'exécution en parallèle des variantes pour la programmation en N-versions et les logiciels dissimilaires.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

Il a été mis en évidence que, outre la possibilité de tolérer les fautes du logiciel, cette technique permet de détecter certaines fautes de spécification. En phase de validation, elle permet de faire des tests dos-à-dos des variantes.

Par contre, il est difficile de déterminer sur quels facteurs jouer pour obtenir une diversification réduisant la présence de fautes corrélées dans les différentes variantes.

Cette technique est coûteuse à mettre en œuvre, il est essentiel de la réserver à des applications critiques.

METHODES ET TECHNIQUES DE REALISATION

A partir d'une spécification, réalisation de variantes par des équipes différentes.

Ces variantes :

a/ programmation N-versions : s'exécutent simultanément avec vote majoritaire, à intervalle de temps régulier sur les données fournies par chacune des variantes. Le voteur peut-être unique ou lui aussi réalisé en N versions (une par variante),

b/ blocs de recouvrement : sont exécutées successivement, après recouvrement du contexte (« remise à zéro »), en cas de non satisfaction de la variante précédente à des critères prédéfinis (appelés assertions).

Exemple d'application

Logiciels dissimilaires : AIRBUS A 320 et A 330.

Blocs de recouvrement : SPIN (Schneider Electric).

ELEMENTS DE COÛTS

Inférieur au développement de N logiciels (N étant le nombre de variantes).

REFERENCES BIBLIOGRAPHIQUES

- Informatique Tolérante aux fautes - Série Arago - n°15 - OFTA - 1994 - MASSON Paris
- Sécurisation des architectures informatiques – exemples concrets - Sous la direction de Jean-Louis Boulanger - Lavoisier 2009.

MODELE DE CROISSANCE DE FIABILITE

(fiabilité du logiciel expérimental)

BUT

La mise en œuvre des modèles de croissance de fiabilité a pour but d'évaluer la fiabilité d'un programme par observation de son comportement, avec pour objectifs principaux :

- prévoir le processus d'apparition des défaillances, dès que le logiciel est opérationnel,
- assurer un niveau de confiance dans le produit réalisé,
- maîtriser l'effort de test de maintenance.

PHASES PREFERENTIELLES D'UTILISATION

Cette démarche étant essentiellement expérimentale avec pour objectif d'obtenir des informations sur le comportement du logiciel dans sa vie opérationnelle, sa mise en œuvre n'est possible qu'à partir du moment où le logiciel est dans une phase relativement stable, en général en phase de tests de validation.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

L'évaluation de la fiabilité d'un programme nécessite le recueil de données relatives à la détection de défaillances survenues lors d'exécutions de ce programme.
Pour obtenir une évaluation représentative de la satisfaction de la continuité du service, il est nécessaire que les données recueillies correspondent à une phase où le programme et son environnement sont suffisamment stables et complet par rapport au besoin.
De plus, cette phase de collecte de données doit avoir été planifiée le plus tôt possible dans le cycle de vie du logiciel pour être la plus efficace.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

Cette démarche est d'autant plus efficace qu'elle a été planifiée tôt dans la vie du projet. Plus tard la collecte a lieu, plus difficile est l'exploitation des données et moins fiable les résultats obtenus.
Il est à noter que cette démarche donnera des résultats significatifs pour des logiciels dont les objectifs de taux de défaillance se situent autour de 10-3 à 10-4/h.
En revanche, cette démarche ne permettra pas à elle seule d'évaluer la fiabilité d'un logiciel de type sécuritaire pour lequel les objectifs de taux de défaillance se situent plutôt aux alentours de 10-6 à 10-9/h.

METHODES ET TECHNIQUES DE REALISATION

La méthode d'évaluation à l'aide de modèle de croissance de fiabilité comporte deux étapes composées de plusieurs tâches :

- observation du déroulement de programme et enregistrement des défaillances et défauts avec :
  définition des objectifs,
  définition et organisation de la collecte des données,
  collecte des données,
- interprétation des données afin d'en déduire une évaluation de la fiabilité du programme avec :
  analyse des données,
  analyse des données temporelles et étude de tendance,
  évaluation de la fiabilité du logiciel à l'aide de modèles de croissance.

ELEMENTS DE COÛTS

Le coût principal provient de la mise en place et de la réalisation de la collecte de données. La phase d'analyse elle-même peut être relativement courte. Néanmoins cette durée dépend directement de la qualité des données disponibles et donc du travail résultant de la mise en forme des données.

REFERENCES BIBLIOGRAPHIQUES

- Software reliability Measurement, prediction application - Musa/Ianino/Okumoto - Mc Graw-Hill Company 1987
- Guides méthodologiques ISDF 1995 :
  « Guide d'évaluation de la fiabilité du logiciel »
  « Fiabilité du logiciel : guide pratique pour le suivi »

MODELE D'EVALUATION PREVISIONNELLE FIABILITE

(fiabilité prévisionnelle du logiciel)

BUT

La mise en œuvre des modèles prévisionnels de fiabilité a pour but d'évaluer la fiabilité d'un logiciel par la description de sa structure, de son processus de mission et de son mode développement, avec pour objectifs principaux :

- prévoir les niveaux de probabilité ou le nombre de manifestations de défaillances en développement et en opérationnel,
- permettre de comparer différentes solutions de développement et choisir celle qui présente le moins de risque que ce soit en terme de choix technologique ou d'effort de développement ou des test,
- pouvoir comparer ses résultats aux objectifs qui peuvent être fixés par contrat dans les normes qui y sont attachées.

PHASES PREFERENTIELLES D'UTILISATION

Cette démarche étant prévisionnelle, elle peut être mise en œuvre très tôt dans un développement et fournir de premiers résultats qui s'affinent lorsque les premiers choix techniques sont précisés au fur et à mesure des phases.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

L'évaluation prévisionnelle de fiabilité nécessite une connaissance des moyens mis en œuvre pour le développement et des efforts de test prévus, du profil de mission requis par la spécification de besoin, des solutions retenues pour le projet logiciel tant du point de vue architecture que de caractéristiques des éléments qui le compose.

Pour conforter l'évaluation prévisionnelle de défaillance vis-à-vis des défaillances effectivement constatées en test ces dernières sont intégrées au prévisionnel afin de prévoir avec plus d'exactitude les évaluations prévisionnelles pour les phases ultérieures.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

Cette démarche constitue un outil d'aide au développement, plus elle est mise en œuvre plus son impact peut être grand et limite certains coûts dus à des corrections tardives.

La démarche est applicable à des projets pour lesquels des résultats de probabilités de défaillances très faibles sont recherchés notamment en présence d'activation multiples de défauts.

Cependant ces évaluations peuvent souffrir d'une mauvais appréhension des entrées conduisant à une modélisation erronée ou d'une prise en compte trop tardive des défaillances constatées en test.

METHODES ET TECHNIQUES DE REALISATION

La méthode d'évaluation prévisionnelle de fiabilité comporte plusieurs étapes de mise en œuvre :

- description des moyens prévus pour le développement que ce soit de façon préventive ou corrective,
- analyse de la spécification, de l'architecture du projet, description des caractéristiques des sous-ensembles,
- évaluation préliminaire prévisionnelle, présentation des résultats et des conditions d'obtention, recommandation pour le développement,
- itération par mise à jour des entrées de la modélisation, nouvelles évaluations, nouveaux enseignements,
- avec les premiers tests intégration des défaillances constatées au modèle et confortement des résultats pour les phases ultérieures.

ELEMENTS DE COÛTS

Le coût résulte de l'analyse de la spécification et des premiers choix fait au niveau projet pour constituer une première évaluation préliminaire et des résultats à en tirer en début de projet. Ce coût est et doit être faible. Il est multiplié par le nombre d'itérations en cours de projet.

Le coût devient plus important lorsque l'on souhaite conforter les résultats par les résultats des tests. Cependant une bonne coordination du projet permet de les maitriser.

REFERENCES BIBLIOGRAPHIQUES

- Combined Hardware/Software Reliability Prediction Methodology, Proceeding Annual Reliability and Maintenability Symposium - Eugene Fiorentino & Edward C. Soistman - 1985
- Impact of Hardware/Software Faults on System Reliability,StudyResults, RADC TR 85 228, Vol 1 - Edward C. Soistman and Katherine B. Ragsdale - December 1985
- A Software Reliability Assessment Model, Proc. Annual Reliability & Maintainability Symp, Las Vegas - P.R. Leclercq - Janvier 1992
- Validation d'un modèle d'évaluation prévisionnelle de la sureté de systèmes programmés à l'aide du retour d'expérience, LM18, Poitier - P.R. Leclercq - Octobre 2012
- Guides méthodologiques ISDF 1995 :

  « Guide d'évaluation de la fiabilité du logiciel »

  « Fiabilité du logiciel : guide pratique pour le suivi »

TECHNIQUE DU MONOPROCESSEUR CODE

BUT

La technique du monoprocesseur codé est une technique essentiellement logicielle qui permet la détection des pannes matérielles. L'objectif de détection des pannes est modulable, cette flexibilité impacte directement la puissance de calcul nécessaire.

PHASES PREFERENTIELLES D'UTILISATION

En phase de codage, la technique consiste à élaborer lors de chaque étape du cycle d'automatisme, une signature arithmétique représentative du bon déroulement du cycle. Cette signature provient de codes arithmétiques attachés aux variables représentant à la fois la valeur de la variable et la suite des opérations qu'elle a subies.

L'hypothèse fondamentale est que le codage est tel que les effets des pannes matérielles auront en termes de codage un comportement numériquement aléatoire, ce qui permet d'évaluer le taux de détection (k étant le nombre de bits utilisés pour coder une variable, la probabilité de non détection est 2^{-k}, probabilité de tirer le bon code au hasard).

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

L'impact de cette technique concerne à la fois le logiciel et le matériel :

- logiciel : il faut utiliser une chaîne de développement spécifique, le programme d'application devant subir des transformations,
- matériel : les entrées doivent être fournies codées au processeur et un chien de garde doit contrôler la signature.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

Les principaux intérêts de cette technique sont l'évaluation possible, sous certaines hypothèses, de l'efficacité de la détection et la compatibilité avec toutes les technologies de processeur.

Par contre, il est nécessaire d'utiliser des cartes d'entrées/sorties et un chien de garde spécifique, la chaîne de production est également spécifique et la durée des codages/décodages d'informations consomme de l'ordre de 95% de la puissance CPU disponible. La technique s'adresse donc plutôt à des applications assez peu complexes mais possédant de fortes contraintes Safety.

METHODES ET TECHNIQUES DE REALISATION

La plupart des outils permettent une saisie (et/ou une visualisation) graphique des automates, complétée par une description fine des actions via un langage dédié.

ELEMENTS DE COÛTS

La technique a déjà été largement utilisée dans le domaine ferroviaire dans des applications homologuées par le Ministère des Transports.

Les éléments dimensionnants pour le coût concernent la mise en place d'une chaîne de développement logiciel et d'outils de mise au point spécifiques, la formation à cette technique et de manière plus marginale sauf pour des productions en grandes séries, le coût du matériel spécifique ajouté (chien de garde et codeurs/décodeurs des entrées/sorties.

REFERENCES BIBLIOGRAPHIQUES

- Le processeur codé, un nouveau concept appliqué à la sécurité de transport - Revue générale des Chemins de Fer - C. GALIVEL - Juin 1990
- The coded processor certification - SAFECOMP 92 - P. OZELLO - Zurich Octobre 92
- Sécurisation des architectures informatiques – exemples concrets - sous la direction de Jean-Louis Boulanger - Lavoisier 2009

MODELISATION COMPORTEMENTALE PAR RESEAUX DE PETRI

BUT

En complétant une spécification statique par une description dynamique du comportement d'un logiciel, la modélisation comportementale permet de vérifier des propriétés supplémentaires de cohérence, de complétude, ... Elle permet aussi de tester les modes dégradés du logiciel, et l'efficacité des techniques de tolérance aux fautes.

PHASES PREFERENTIELLES D'UTILISATION

La modélisation comportementale constitue une aide précieuse pour spécifier et concevoir un produit, indépendamment des vérifications qu'elle permet de réaliser. Elle est donc à employer en fin de spécification statique et permet d'affiner celle-ci.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Si la spécification dynamique peut se faire sur papier (sous forme d'automates états/transitions), son intérêt réside dans sa simulation, qui peut être interactive (l'opérateur choisissant pas à pas le chemin d'exécution), ou mieux exhaustive (tous les comportements du modèle sont simulés et mémorisés dans un graphe global appelé graphe d'états du modèle). Cette simulation nécessite l'emploi de stations de travail puissantes et dotées de beaucoup de mémoire vive.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

Analyse de dysfonctionnements

Ce type de vérification porte sur la forme du modèle et permet de mettre en évidence des comportements incohérents ou bloquants : non-réinitiabilité, non-déterminisme, code mort (transitions non tirables), états de blocage, oscillations de variables (avec des automates étendus).

Vérification du modèle par rapport au besoin

Il s'agit de vérifier des propriétés spécifiques au modèle du type : invariants logiques, cohérence du comportement fourni par rapport au comportement attendu, calcul de propriétés globales.

Evaluations temporelles

Certains outils fournissent un mode de simulation automatique dit « stochastique » dans lequel les transitions sont tirées à des dates aléatoires suivant la loi de distribution qui leur a été associée. Cela permet d'évaluer certaines caractéristiques quantitatives du modèle : ses performances (temps de réponse moyen, ...) ainsi que sa fiabilité (fréquence moyenne d'occurrence des pannes, ...).

Génération des tests

Le modèle devient une référence par rapport à laquelle le système devra être validé. Un outil permet de générer automatiquement des scénarios de tests de validation, représentatifs, à partir des spécifications.

ELEMENTS DE COÛTS

Investissement important en termes d'outils (et stations compatibles). Proportionnel à la complexité dynamique du système.

REFERENCES BIBLIOGRAPHIQUES

- Du Grafcet aux réseaux de Petri - René David et Hassane Alla - Paris, Hermès - 1992

STATE CHARTS
(ou méthode de HAREL)

BUT

La méthode par State Charts a pour but de décrire et de valider le comportement de systèmes réactifs.

PHASES PREFERENTIELLES D'UTILISATION

Cette méthode est utilisée pour la spécification et la conception des systèmes réactifs.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Il est nécessaire de disposer des exigences du client qui sont analysées pour définir le système, son environnement et ses fonctions.

Il est souhaitable d'être familiarisé avec le formalisme des machines à états finis.

L'utilisation d'un outil est nécessaire pour traiter efficacement les State Charts (p. ex : Statemate).

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

Cette méthode est d'autant plus efficace pour exprimer le comportement d'un système.
Les state charts étant simulables, la spécification obtenue est sans ambiguïté, et validable par son exécution.
Cette méthode peut devenir assez lourde d'utilisation pour des systèmes complexes.

METHODES ET TECHNIQUES DE REALISATION

Le modèle complet permet la description du système selon 3 vues :

- la spécification de l'architecture (module-charts) qui décrit les modules logiques et physiques, ainsi que l'environnement,
- la spécification des activités (activity-charts) comprenant une partie flots de données et une partie contrôle,
- la spécification du contrôle (state charts).

Le state chart est une extension du diagramme à états finis. Il permet de représenter une description hiérarchique et structurée, des actions ou activités simultanées, des conditions de transitions complexes. Il permet principalement :

- le raffinement en détaillant un état par un ou plusieurs automates. Dans le cas de plusieurs automates, il y a évolution parallèle,
- la spécification explicite ou par défaut d'un état de départ pour un ensemble d'automates, et de synchronisations multiples,
- l'association de contraintes de temps à des états.

ELEMENTS DE COÛTS

- L'utilisation de la méthode nécessite une durée d'apprentissage assez importante.
- Proportionnel à la complexité du système à décrire.

REFERENCES BIBLIOGRAPHIQUES

- Spécification et conception des systèmes - une méthodologie : J.P. CALVEZ - Masson, 1990

ANALYSE STATIQUE PAR INTERPRETATION ABSTRAITE

BUT

L'analyse statique par interprétation abstraite vise à calculer une approximation (au sens « vision restreinte ») du comportement du logiciel afin de vérifier cette approximation vis-à-vis de la propriété pour laquelle elle a été calculée

PHASES PREFERENTIELLES D'UTILISATION

Elle s'utilise plutôt en fin de développement, sur du code déjà écrit, souvent (mais pas uniquement) en Ada ou en C/C++.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

L'utilisation de l'analyse statique nécessite un outillage dédié qui est fonction du type d'analyse à réaliser.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

L'analyse statique par interprétation abstraite fonctionne par la fabrication d'une approximation comportement du du logiciel à vérifier. Cette approximation est faite de telle manière qu'elle est suffisamment précise pour l'exigence à vérifier sans être trop précise afin de limiter l'espace de valeurs à vérifier pour des besoins pratiques. Les exigences à vérifier portent usuellement sur les flots de données ou les flots de contrôle, et portent sur des propriétés telles que l'absence de dépassement d'entiers, le non-dépassement des bornes de tableaux, la détection de code mort, etc. Il s'agit donc de propriétés applicables a posteriori sur du code développé sans optique de vérification.

- Avantages : peut être utilisé a posteriori sur un développement
- Inconvénients : les propriétés vérifiables sont peu complexes, plus de complexité peut imposer la réécriture du logiciel selon des règles de codage précises

METHODES ET TECHNIQUES DE REALISATION

La mise en œuvre consiste le plus souvent à l'utilisation d'outils dédiés à appliquer sur le logiciel développé. En revanche, si les propriétés à vérifier ne sont couvertes par aucun outil, l'écriture d'un outil dédié nécessite un très solide bagage mathématique en interprétation abstraite et dans le langage du logiciel.

ELEMENTS DE COÛTS

Les coûts principaux dans l'utilisation d'outils d'analyse statique sont le coût de l'outil lui-même et le coût d'une éventuelle adaptation du code selon des règles de codage propres à l'outil (instrumentation du code).

REFERENCES BIBLIOGRAPHIQUES

- http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis
- Interprétation abstraite **»,** Numéro spécial de Technique et science informatique*,* Informatiques, enjeux, tendances et évolutions - sous la direction de René Jacquart, Vol. 19, Nb 1-2-3, pp. 155-164. Éditions Hermès science Paris - Jan. 2000
- Utilisations industrielles des techniques formelles – interprétation abstraite - sous la direction de Jean-Louis Boulanger - Collection IC2 - HERMES-Lavoisier - 2011

MODEL-CHECKING

BUT

Le model-checking vise à générer tous les états d'un système afin de s'assurer que tous ces états vérifient un propriété donnée.

PHASES PREFERENTIELLES D'UTILISATION

Elle s'utilise plutôt en amont de la phase de développement, voire en phase de spécification. Selon les cas en effet, le système sera modélisé dans un formalisme (celui du model-checking) et développé dans un autre (le langage servant au codage).

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Les méthodes de model-checking requièrent souvent des connaissances en logique temporelle (comme LTL, CTL, CTL*, etc) et en automates (au sens diagrammes état-transition).

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

Le model-checking fonctionne par la génération de tous les états d'un système et la vérification de ces états par rapport à une exigence exprimée dans un formalisme donné.
Les propriétés vérifiables par model-checking sont de plus haut niveau que pour l'analyse statique par interprétation abstraite. Selon les outils, le système peut être exprimé dans un langage de programmation usuel ou il faudra modéliser le système dans un formalisme ad-hoc. Le formalisme utilisé pour exprimer les propriétés à vérifier est le plus souvent une logique temporelle : si l'avantage en est une certaine précision, l'inconvénient serait de trouver une main d'œuvre formée à l'utilisation de cette logique.

- Avantages : technique très automatisée, les vérifications proposent souvent des contre-exemples permettant de découvrir des scénarios d'erreurs non détectés auparavant. La technique fait que la méthode est particulièrement adaptée aux systèmes concurrents
- Inconvénients : nécessitera une adaptation ou une instrumentation dans le cas d'un code existant. Les propriétés ne seront pas toujours vérifiables, et la méthode fonctionnera mieux sur un système à nombre d'états finis.

METHODES ET TECHNIQUES DE REALISATION

La mise en œuvre consiste en la modélisation du système dans le formalisme de model-checking et la modélisation des propriétés dans le formalisme de vérification. Par exemple, le système sera décrit selon un réseau de Petri et les propriétés seront décrites avec CTL*.
La vérification quant à elle est gérée par des outils dédiés.

ELEMENTS DE COÛTS

Le coût de mise en œuvre est principalement lié à la modélisation du système et des propriétés. Le coût est d'ailleurs également lié l'apprentissage des formalismes en question.

REFERENCES BIBLIOGRAPHIQUES

- http://en.wikipedia.org/wiki/List_of_model_checking_tools
- Automatic verification of finite state concurrent systems using temporal logic - ACM Trans. on Programming Languages and Systems, vol. 8, no 2 - E. M. Clarke, E. A. Emerson et A. P. Sistla - 1986
- Principles of Model Checking - Christel Baier & Joost-Pierter Katoen - The MIT Press - 2008

PREUVE FORMELLE

BUT

La preuve formelle est, avec le model-checking, un moyen d'exprimer et vérifier des propriétés sur un logiciel. Elle se base sur l'utilisation de la logique et d'un système déductif. Elle est utilisée lorsque les propriétés à faire vérifier par le logiciel sont complexes et/ou que le nombre d'états possibles du système rend le model-checking inutilisable.

PHASES PREFERENTIELLES D'UTILISATION

La preuve formelle nécessite souvent un langage de modélisation/conception spécifique, ou des règles de codage spécifiques, aussi il est préférable de l'utiliser dès que possible, dès la phase de conception et tout au long de la phase de développement.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

La mise en œuvre fait appel à des outillages dédiés, au minimum un compilateur/générateur d'obligations de preuve ainsi qu'un outil de preuve. Elle requiert également du personnel qualifié et familier de ces outils de preuve.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

La preuve formelle pour vérifier les propriétés pourra être automatique ou interactive, mais dans tous les cas mécanisée. L'échec d'une preuve de propriétés requiert une expertise sur la raison de cet échec : il peut s'agir d'un manque d'hypothèses sur le fonctionnement du système, d'une propriété violée ou d'une trop grande complexité de la partie logicielle à vérifier. Pour certaines méthodes formelles, il est possible d'utiliser pour les étapes de vérification non pas des techniques de preuve, mais des techniques de model-checking. Dans ce cas, la production de contre-exemples en cas de propriété violée est possible.

- Avantages : cadre rigoureux de description du système, grande expressivité des propriétés à garantir par le système (selon la méthode formelle retenue), le code obtenu est de meilleure qualité qu'avec d'autres approches
- Inconvénients : vérification coûteuse en temps et en expertise (besoin d'experts en preuve, ou en modélisation système pour aider à faire passer la preuve)

METHODES ET TECHNIQUES DE REALISATION

La preuve formelle dépend de la logique et du système déductif utilisés par l'outil de preuve retenu. Selon les cas, la preuve sera plus ou moins automatisée en fonction de la complexité des formules à prouver.

ELEMENTS DE COÛTS

Comme indiqué plus haut, l'utilisation de la preuve requiert du personnel qualifié : il s'agit là du point le plus coûteux, suivi du coût en temps de développement.

REFERENCES BIBLIOGRAPHIQUES

- http://en.wikipedia.org/wiki/Automated_theorem_proving
- http://fr.wikipedia.org/wiki/Assistant_de_preuve

METHODE FORMELLE

BUT

L'utilisation d'une méthode formelle est impliqué par le besoin de maîtrise sur le système logiciel à développer. Elle doit permettre de répondre sans ambiguïté aux questions suivantes :

- Que fait le logiciel ?
- Comment le logiciel doit-il le faire ?
- Fait-il ce qu'il doit faire ?

La réponse à ces questions s'obtient par l'utilisation de techniques rigoureuses, basées sur les mathématiques, afin de raisonner sur des systèmes logiciels et de démontrer leur validité par rapport à une certaine spécification.

PHASES PREFERENTIELLES D'UTILISATION

Cela dépend de la méthode formelle retenue. Différents types de vérification formelle sont présentés dans les fiches sur l'analyse statique par interprétation abstraite, sur le model-checking et sur la preuve formelle : la phase peut donc aller de la spécification au développement final.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Quel que soit le type de méthode retenu, les ressources nécessiteront toujours :

- Un outillage dédié
- Des experts pour concevoir et/ou interpréter les résultats de la vérification

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

L'intérêt des méthodes formelles est double : le premier est d'obtenir la maîtrise du logiciel obtenu. Elles permettent de maîtriser la complexité du système en imposant un cadre rigoureux de raisonnement sur le logiciel en développement.

Le deuxième fait suite au retour d'expérience sur leur utilisation : il s'avère que le coût de mise en place contrebalance largement le coût de tests de validation. En effet, certaines propriétés étant désormais vérifiées au niveau du système conçu, le concepteur est assuré que ces propriétés sont intrinsèques au système. Il n'y a donc plus lieu de réaliser les tests correspondant à ces propriétés.

Il ne faut cependant pas négliger le coût de mise en place de l'utilisation de ces méthodes : aussi il appartient au décideur de cadrer précisément la porté de leur utilisation. Usuellement il s'agira du noyau critique du code, le reste étant développé et vérifié par des méthodes classiques.

ELEMENTS DE COÛTS

Comme indiqué ci-avant, le coût principal des méthodes formelles est l'expertise, que ce soit en personnel qualifié ou en formation de personnel aux méthodes formelles. Ensuite, le coût dépendra de la méthode formelle envisagée.

REFERENCES BIBLIOGRAPHIQUES

- http://formalmethods.wikia.com/wiki/Formal_methods
- Utilisations industrielles des techniques formelles – interprétation abstraite - sous la direction de Jean-Louis Boulanger Collection IC2 - HERMES-Lavoisier - 2011
- Techniques industrielles de modélisation formelle pour le transport - sous la direction de Jean-Louis Boulanger - Collection IC2 - HERMES-Lavoisier - 2012
- Outils de mise en œuvre industrielle des techniques formelles - Collection IC2 - sous la direction de Jean-Louis Boulanger - HERMES-Lavoisier - 2012
- Understanding Formal Methods - Springer Verlag, Foreword by G. Huet - Translation editor M. Hinchey - Jean-François Monin - ISBN 1-85233-247-6 - 2002

LE LANGAGE LUSTRE

BUT

LUSTRE est un langage pour le développement des logiciels temps réel critiques dans des domaines tels que le contrôle-commande, les automatismes et le traitement du signal.

PHASES PREFERENTIELLES D'UTILISATION

Le langage LUSTRE couvre les phases de spécification et de conception. De plus, il est possible à partir d'une description LUSTRE, de générer automatiquement du code C (phase de codage).

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

La mise en œuvre de LUSTRE pour un développement industriel nécessite un outillage dédié (éditeur, générateur de code).

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

LUSTRE est un langage flots de données synchrone. Un programme LUSTRE peut être représenté graphiquement par un réseau d'opérateurs agissant en parallèle au rythme de ses entrées. Le modèle d'exécution de LUSTRE garantit le déterminisme fonctionnel et assure un temps d'exécution borné.

Du point de vue de l'utilisateur, LUSTRE facilite la manipulation de phénomènes temporels. L'aspect graphique du langage renforce la lisibilité des descriptions.

A partir d'une description LUSTRE, il est possible de générer du code C directement compilable. De plus, cette description peut être simulée afin de valider son comportement fonctionnel.

Enfin, grâce à la sémantique mathématique du langage, il est possible de vérifier formellement des propriétés (par model checking).

LUSTRE est adapté aux automatismes en général (représentation équationnelle ou blocs fonctionnels). Il est mal adapté aux applications de type asynchrone (par exemple, protocole de communication).

METHODES ET TECHNIQUES DE REALISATION

LUSTRE possède les propriétés des langages de haut niveau : les concepts de modularité et de hiérarchisation font partie intégrante du langage et portent à la fois sur les données et sur les traitements. La complétude et la cohérence sont garanties par la sémantique du langage : vérification du principe de causalité (i.e. la sortie d'un programme ne peut pas dépendre d'entrées futures), détection de définition cyclique, contrôle sur le typage des données.

ELEMENTS DE COÛTS

L'apprentissage de LUSTRE ne présuppose pas de connaissances particulières ; il est souhaitable d'être familiarisé avec la représentation équationnelle ou la représentation blocs fonctionnels.

REFERENCES BIBLIOGRAPHIQUES

- Utilisation d'un langage synchrone à flots de données pour la réalisation de logiciels temps réels embarqués - Real Time Systems Conference (RTS'93) - J. L. Bergerand, P. Ghaleb, et D. Pilaud - Paris, France - janvier 1993
- Programmation et vérification des systèmes réactifs : le langage LUSTRE - N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud - Techniques et Sciences Informatiques vol. 10 n°2 - 1991
- Approche synchrone et logiciels critiques pour l'automatique - C. Dubois - Revue de l'Electricité et de l'Electronique N°1 - juin 1995

SCADE
BUT SCADE est un environnement complet permettant d'aller de la spécification au code avec des phases de simulation et de preuve.
PHASES PREFERENTIELLES D'UTILISATION SCADE couvre les phases de spécification et de conception. De plus, il est possible à partir d'une description SCADE (5 ou 6), de générer automatiquement du code C ou du code ADA (phase de codage).
RESSOURCES REQUISES POUR LA MISE EN OEUVRE Suite d'outils du commerce qui peut être installée sur un ordinateur bureautique.
INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI SCADE est un environnement de développement. Dans sa version 5, il se base sur le langage LUSTRE. Dans sa version 6, le langage LUSTRE a été étendu à la notion de tableau et d'itérateur de tableau et le langage source est un fichier XML.
METHODES ET TECHNIQUES DE REALISATION SCADE dispose des mêmes propriétés que LUSTRE et dans sa version 6, il dispose d'un pouvoir d'expression plus important permettant de combiner le synchrone et les algorithmes.
ELEMENTS DE COÛTS SCADE est un environnement assez cher car constitué d'un ensemble d'outils dont un générateur de code C certifié et d'un ensemble de kits documentaires permettant la délivrance de certificats aux normes en vigueur dans différents secteurs d'utilisation (aéronautique, …).
REFERENCES BIBLIOGRAPHIQUES • Scade 6 a model based solution for safety critical software development. In Embedded Real-Time Systems Conference - François-Xavier Dormoy - 2008

LE LANGAGE B

BUT

B est une méthode de développement formel pour réaliser des composants logiciels et dont le but est de donner les moyens de vérifier la correction par rapport à la spécification.

PHASES PREFERENTIELLES D'UTILISATION

Le langage B couvre les phases de spécification, conception et réalisation du logiciel. Le code (par exemple en ADA ou en C) du composant est engendré automatiquement.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Un ensemble d'outils logiciels d'assistance à la mise en œuvre est nécessaire pour une application industrielle, notamment pour les phases de vérification (preuve) et de génération de code.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

B est un langage homogène utilisé de la spécification à la réalisation, autorisant par ce moyen des vérifications formelles.

Les démonstrations mathématiques (ou preuves) effectuées lors de l'utilisation de B garantissent la cohérence interne des différents modules, la cohérence de leurs interactions, ainsi que la validité des modules de réalisation par rapport aux modules de spécification.

L'échec d'une démonstration mathématique, lorsqu'il ne s'agit pas d'un problème du modèle B, met en évidence une incomplétude ou une incohérence du cahier des charges.

B est particulièrement adapté à la réalisation d'automatismes, de contrôles-commandes, ou à l'implantation et la vérification d'algorithmes critiques (confidentialité). Par contre, bien que des éléments de réponse existent, B est peu adapté à la réalisation de systèmes logiciels parallèles ou temps réel critiques.

METHODES ET TECHNIQUES DE REALISATION

B est un langage de spécification formelle permettant de modéliser un système (contenant des composants logiciels) par des machines abstraites. De plus, c'est un langage modulaire offrant des mécanismes d'encapsulation des données, de hiérarchisation et de décomposition. Les propriétés du système modélisé sont exprimées par des formules mathématiques. La sémantique formelle du langage B garantit le respect de ces propriétés dans l'implantation logicielle.

ELEMENTS DE COÛTS

L'apprentissage de B ne présuppose pas de connaissances particulières. On constate expérimentalement qu'une formation de plusieurs semaines est nécessaire pour être opérationnel (écrire du B dont on maîtrise la preuve).

La rentabilité de l'application industrielle du langage B est acquise pour les constructeurs l'utilisant (le gain en phase de validation contrebalance largement le surcoût de la spécification).

REFERENCES BIBLIOGRAPHIQUES

- B-Book, assigning programs to meanings - J.-R. ABRIAL - Cambridge University Press - 1996
- Les apports de la preuve en B - C. ROQUES, F. BUSTANY, D. SABATIER - Actes du colloque 10 Fiabilité & Maintenabilité - Saint-Malo - 1996

SIMULATION

BUT

La simulation est une technique qui consiste à rendre exécutable un modèle, pour analyser les modes de fonctionnement et/ou de dysfonctionnement.

PHASES PREFERENTIELLES D'UTILISATION

La simulation peut-être utilisée comme moyen de vérification (vérification d'une spécification, d'une architecture et/ou d'une conception) mais elle peut, lorsqu'elle est associée à la réalisation d'un modèle spécifique, être un moyen de sélection de cas de tests.

La sélection de cas de tests par simulation est utile en phase de validation système et/ou logiciel.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Pour réaliser une simulation, il faut disposer d'un modèle du système/logiciel et lui ajouter un modèle d'environnement afin de limiter le nombre de cas à simuler.

Le modèle à simuler peut être vite saturé par le manque de ressources informatiques (mémoire, temps de traitement UC, …), les outils de simulation étant très consommateurs de ressources.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

La qualité de la simulation dépend de la représentativité et de la validité du modèle d'entrée et du modèle d'environnement sur lesquels s'exécute la simulation.

METHODES ET TECHNIQUES DE REALISATION

Les méthodes et techniques de réalisation de la simulation sont directement liées à l'outil choisi pour construire le modèle et à sa capacité à exécuter le modèle produit.

ELEMENTS DE COÛTS

Directement dépendant de la taille et de la complexité du modèle d'entrée et de l'outil de simulation choisi.

REFERENCES BIBLIOGRAPHIQUES

- B-Book, assigning programs to meanings - J.-R. ABRIAL - Cambridge University Press - 1996

UML
Unified Modelling language

BUT

UML est un langage de modélisation graphique à base de pictogrammes. UML est utilisé pour spécifier, visualiser, modifier et construire les documents nécessaires au bon développement d'un logiciel, au travers d'un standard de modélisation permettant de représenter l'architecture logicielle.

UML est une notation (et non un langage) qui est la réunion de plusieurs approches. L'idée étant de profiter des différentes capacités de modélisation au sein d'une même notation.

PHASES PREFERENTIELLES D'UTILISATION

Prévu à la base pour la modélisation de la spécification d'un logiciel, UML peut maintenant, dans sa version normalisée, couvrir les aspects spécification, architecture et conception.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Une difficulté de la notation UML réside dans l'absence d'une sémantique complète et dans la possibilité de se définir sa propre utilisation. Il est donc nécessaire en début de projet de se définir un guide de modélisation permettant de garantir que l'ensemble des membres d'un projet (équipe de développement, équipe V&V, équipe sécurité) interprète le modèle de la même manière.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

UML ne permet pas de réaliser la vérification interne du modèle et la génération de code.

UML doit être considéré comme un moyen de réaliser des diagrammes qui viennent en aide à des éléments textuels. L'utilisation dans le cadre d'application critique doit s'accompagner de la mise en place d'une méthodologie et de guide permettant de respecter les exigences de ce type d'application.

METHODES ET TECHNIQUES DE REALISATION

La notation UML permet de réaliser des modèles permettant d'aider à la compréhension des éléments textuels.

ELEMENTS DE COÛTS

Dans un premier temps, la notation UML semble facile a appréhender mais le nombre de concepts et de symboles rend difficile la réalisation (comment choisir la bonne flèche qui relie 2 composants) d'un modèle et sa compréhension.

REFERENCES BIBLIOGRAPHIQUES

- Le guide de l'utilisateur UML - Grady Booch, James Rumbaugh, Ivar Jacobson - 2000
- UML 2 par la pratique - Études de cas et exercices corrigés - Pascal Roques – Eyrolles - 2006

SysML
System Modelling Language

BUT

SysML est une notation (et non un langage) qui a été construit sur la base de UML dans le but de couvrir l'aspect système.

PHASES PREFERENTIELLES D'UTILISATION

SysML permet d'identifier les exigences, de les modéliser et de faire le lien avec les cas de tests.

RESSOURCES REQUISES POUR LA MISE EN OEUVRE

Comme pour la notation UML, la notation SysML nécessite de définir une méthodologie et des guides permettant à l'ensemble du projet de créer, de lire et de vérifier les modèles.

INTERÊTS, INDICATIONS ET CONTRE-INDICATIONS OU LIMITATIONS D'EMPLOI

SysML introduit de nouveaux diagrammes par rapport à UML. Parmi ces diagrammes, on trouve le diagramme d'exigences, le diagramme de composants et le diagramme des données. Ces trois diagrammes permettent de mieux appréhender la spécification, tant au niveau système que logiciel.

METHODES ET TECHNIQUES DE REALISATION

La notation SysML permet de réaliser des modèles permettant d'aider à la compréhension des éléments textuels.

ELEMENTS DE COÛTS

Dans un premier temps, la notation SysML semble facile a appréhender mais le nombre de concepts et de symboles rend difficile la réalisation (comment choisir la bonne flèche qui relie 2 composants) d'un modèle et sa compréhension.

REFERENCES BIBLIOGRAPHIQUES

- SysML par l'exemple - Un langage de modélisation pour systèmes complexes - Pascal Roques – Eyrolles - 2009

ANNEXE 4. EXEMPLES DE METHODES ET TECHNIQUES

Les pages suivantes illustrent les principales techniques préconisées dans le guide.

Les exemples proposés doivent permettre de mieux comprendre les principes d'utilisation et la manière de présenter les résultats.

Toutefois le formalisme adopté dans ces exemples n'a aucun caractère normatif et il appartient au lecteur de ce guide d'identifier (notamment à partir des fiches méthodes jointes en annexe 3) et de rechercher la forme normalisée éventuelle associée à la méthode.

Dans certains domaines les prescripteurs peuvent aussi imposer la forme de certaines analyses afin d'harmoniser la présentation des résultats des analyses faites par les différents fournisseurs.

Liste des méthodes illustrées dans cette annexe :

- Analyse préliminaire de risques
- Analyse des Modes de Défaillance, de leur Effet (et de leur Criticité), AMDE(C)
- Arbres de Fautes du logiciel
- Programmation diversifiée : Blocs de recouvrement
- Programmation diversifiée : programmation N_versions
- Programmation diversifiée : programmation N autotestables (variantes 1 et 2)

ANALYSE PRÉLIMINAIRE DE RISQUES

1 SOUS-SYSTEME OU FONCTION	2 PHASE	3 ELEMENTS DANGEREUX	4 CAUSE	5 SITUATION DANGEREUSE	6 CONSEQUENCES	7 GRAVITE	8 MOYEN DE REDUCTION DE RISQUES	9 GRAVITE RESULTANTE	10 MOYEN DE VERIFICATION
Panneau de contrôle	Exploitation	Incendie non signalé	Voyant défectueux	Pas de détection d'incendie d'une zone	Risque intervention tardive, propagation	Majeure	Doublement de l'affichage par un klaxon	Mineure (panne double)	Revue architecture Tests
			Erreur logiciel détection	Pas de détection incendie	Incendie non maîtrisé	Majeure	Redondance matériel / logiciel	mineure (panne double)	Revue de conception système

ANALYSE DES MODES DE DÉFAILLANCE, DE LEUR EFFET (ET DE LEUR CRITICITÉ) AMDE(C)

COMPOSANT	FONCTIONS ETATS	MODES DE DEFAILLANCE	CAUSES POSSIBLES	EFFETS	GRAVITE	MOYENS DE DETECTION	PROBABILITE	CRITICITE	ACTIONS CORRECTIVES	REMARQUES
surv_zone	Surveillance zone	Perte de la fonction	Blocage process	Zone non surveillée	Elevée	Affichage témoin état fonction	Faible	Moyenne	fonction monitoring de sécurité	
dec_alarm	Déclenchement alarme	Déclenchement intempestif	Incertitude capteur	Inondation zone inutile	Moyen	Aucun	Moyenne	Moyenne	Double capteur Autre gamme de capteur	

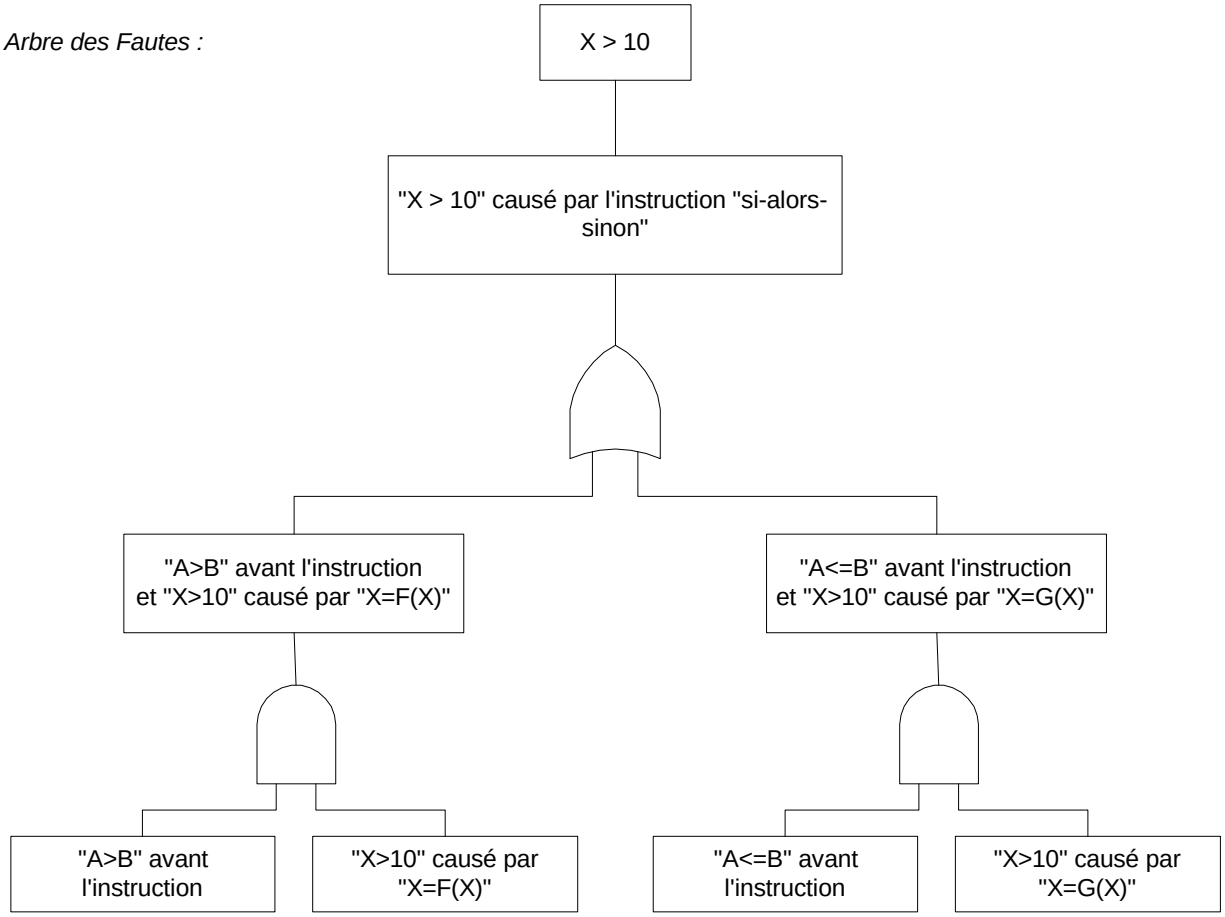
ARBRE DES FAUTES DU LOGICIEL

Evénement-racine : X > 10

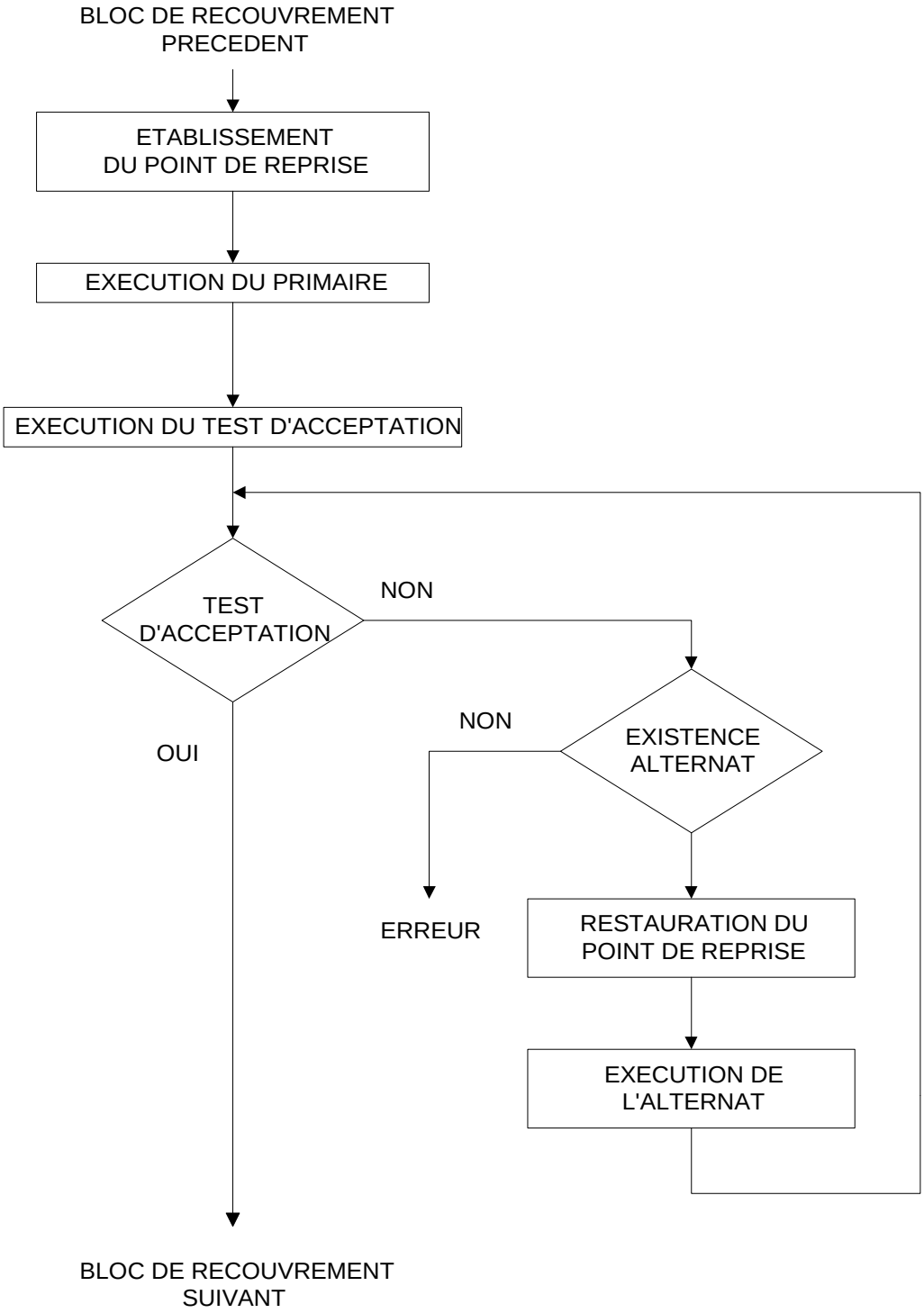
Instruction concernée :

```
si A>B    alors      X=F(X)
          sinon      X=G(X)
```

Arbre des Fautes :

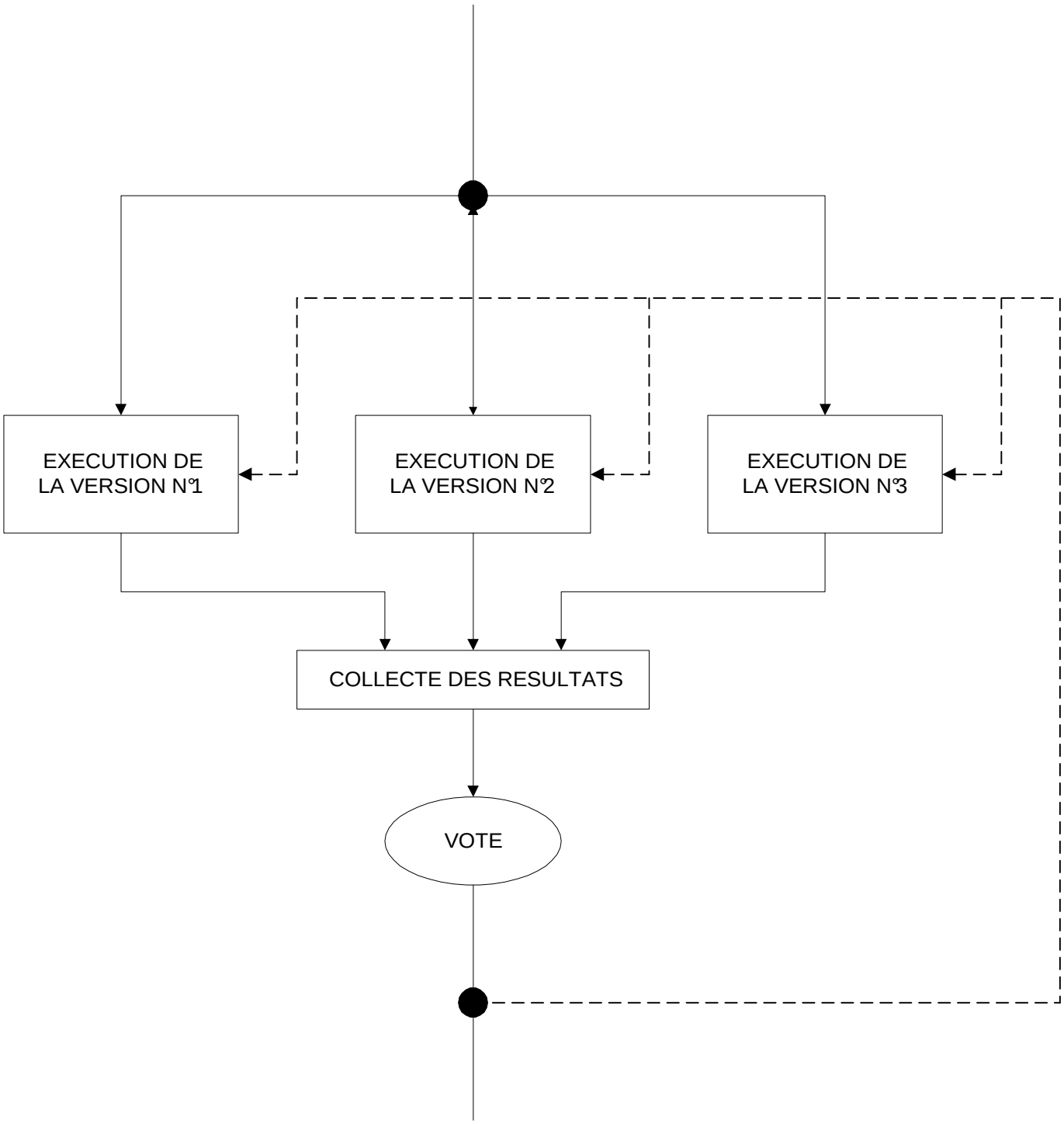


BLOC DE RECOUVREMENT



PROGRAMMATION N VERSION

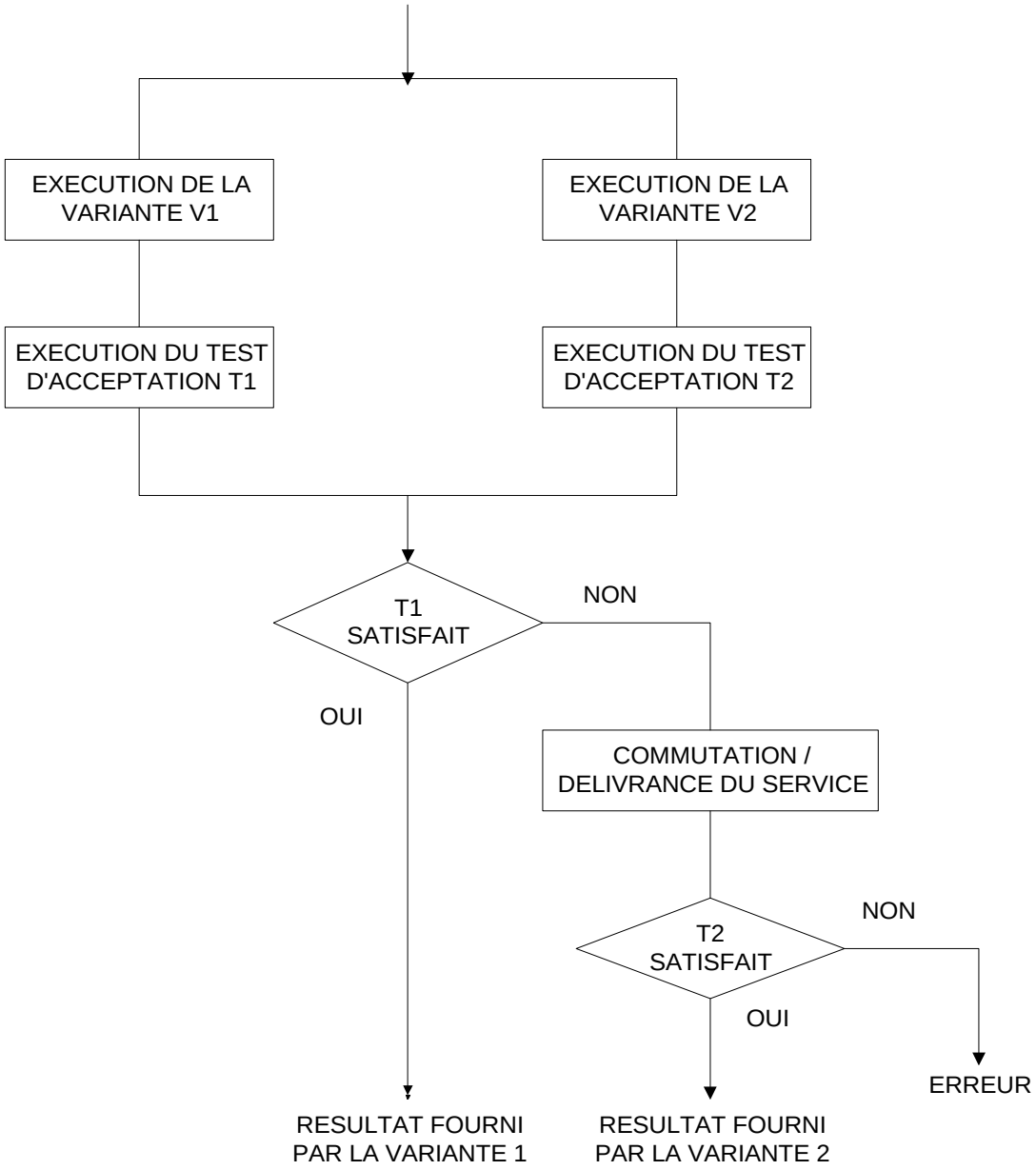
PRINCIPE POUR TROIS VERSIONS



PROGRAMMATION N AUTOTESTABLES

PRINCIPE EN DEUX AUTOTESTABLES

(COMPOSANT AUTOTESTABLE = VARIANTE + TEST D'ACCEPTATION



PROGRAMMATION N AUTOTESTABLES

PRINCIPE EN DEUX AUTOTESTABLES

(COMPOSANT AUTOTESTABLE = 2 VARIANTES + ALGORITHME DE COMPARAISON