INRS ND 2140-181-00

Cahiers de notes documentaires - Hygiène et sécurité du travail - *N° 181, 4e trimestre 2000*

Comment construire les tests d'un logiciel

→ *P. Charpentier, Département Ingéniérie des équipements de travail, Centre de l'INRS-Lorraine, Nancy, N. Diette, Sté Surlog, 1 bis rue du Petit-Clamart, 78140 Vélizy-Villacoublay, et F. Escaffre, Sté CEP-Systèmes/RIdatas, 150 rue Nicolas-Vauquelin, 31047 Toulouse cedex*

Comment construire les tests d'un logiciel (*)

→ *P. Charpentier,*
Département Ingéniérie des équipements de travail,
Centre de l'INRS-Lorraine, Nancy,
N. Diette, Sté Surlog,
1 bis rue du Petit-Clamart,
78140 Vélizy-Villacoublay,
et F. Escaffre, Sté CEP-Systèmes/RIdatas, 150 rue Nicolas-Vauquelin,
31047 Toulouse cedex

Le logiciel « zéro défaut » n'existe pas. La présence de fautes logicielles systématiques, introduites à la conception d'un dispositif programmé, doit donc être considérée avec beaucoup d'attention, en particulier lorsque les conséquences de ces fautes peuvent influer sur la sécurité d'un dispositif complexe.
Cet article propose une méthode de mise en œuvre des tests, qui sont la principale composante de la vérification d'un logiciel. En annexe, sont présentées les exigences qui permettent de s'assurer que cette vérification a effectivement été réalisée et qu'elle s'avère efficace. Pour détecter un maximum de fautes, ces tests doivent être conduits en suivant une démarche qui définit, dans un premier temps, les objectifs de tests et la stratégie adoptée pour les atteindre, et qui démontre, dans un deuxième temps, que lesdits objectifs ont été satisfaits.

logiciel • faute • test • méthodologie • informatique

There is no such thing as a « zero fault » software. The presence of systematic software faults introduced at the design stage of a programmable device must therefore be given comprehensive and careful consideration, in particular when the consequences of these faults can influence the safety of a complex system.
This article proposes a method of carrying out tests, which are the main component of software verification. The requirements to ascertain that this verification has been efficiently carried out are given in annex. To detect the maximum number of faults, testing must be carried out according to an approach that firstly defines the objectives of the tests and the strategy adopted, and secondly that demonstrates that the said objectives have been achieved.

• *software* • *fault* • *test* • *methodology*

Dans le domaine des machines, les systèmes programmés peuvent être utilisés pour traiter des informations relatives à la sécurité, à condition de respecter les exigences de sécurité qui leurs sont applicables [1]. Ils doivent, entre autres, satisfaire aux prescriptions contenues dans la norme européenne EN 954-1 [2, 3], un des points les plus délicats à traiter étant de garantir un comportement sûr en présence de fautes.

En effet, il est matériellement impossible de caractériser et de prévoir toutes les fautes susceptibles d'affecter ces systèmes complexes [4]. Seul le recours à des moyens éprouvés en sûreté de fonctionnement (SdF), pour développer le matériel et le logiciel de ces systèmes, permet d'obtenir en final une confiance justifiée dans la sécurité de ces systèmes [5, 6]. Ces moyens peuvent être répartis en quatre catégories [7] :

■■ La prévention des fautes. C'est l'ensemble des dispositions prises par le constructeur pour assurer un développement maîtrisé du produit. La *prévention des fautes* concerne donc le processus général de conception du système.

■■ La tolérance aux fautes. C'est l'utilisation de méthodes et techniques constructives destinées à fournir un service à même de remplir la, ou les fonctions du système en dépit des fautes pouvant l'affecter. La tolérance aux fautes a donc une influence sur l'architecture du système. Dans un cadre de sécurité, la *tolérance aux fautes* se limitera à l'utilisation de techniques de détection des fautes, la réaction étant la mise en position de repli.

■■ L'élimination des fautes. C'est l'application, au cours de chaque phase du cycle de vie du système, de méthodes et techniques destinées à réduire la présence des fautes, en nombre et en sévérité.

(*) Travaux réalisés dans le cadre du Projet européen STSARCES, financé par la DG XII (Commission européenne , Direction générale XII - Science, recherche et développement, Bruxelles, Belgique).

L'élimination des fautes regroupe l'ensemble des techniques de vérification du matériel et du logiciel.

■■ La prévision des fautes. C'est l'évaluation du comportement du système par rapport à l'apparition des fautes. Elle se base sur un ensemble de méthodes et techniques destinées à estimer la présence, la création et les conséquences des fautes. La *prévision des fautes* est destinée à s'assurer que les techniques de tolérance et d'élimination mises en œuvre ont l'efficacité souhaitée.

Dans le cadre du projet européen STSARCES (1) [8], l'INRS a mené des travaux, en coopération avec les sociétés Veridatas (aspect qualité) (2) et Surlog (aspect sûreté de fonctionnement), pour adapter deux techniques complémentaires de SdF aux systèmes programmés utilisés en sécurité des machines. Ces techniques sont destinées à traiter les fautes logicielles systématiques (3) affectant les logiciels systèmes (4), en s'attachant :

■■ À prévenir ces fautes, par la *définition d'exigences en matière de qualité et de sûreté des logiciels* [9, 10]. Des exigences ont été formulées pour proposer aux concepteurs une méthode de travail rigoureuse, dans le but de minimiser le nombre des fautes restant dans le logiciel en phase d'exploitation. Ces exigences sont adaptées aux logiciels systèmes développés pour la sécurité des machines et sont cohérentes avec la norme CEI 61508 (partie 3 « Prescriptions concernant les logiciels » notamment [11]). Elles concernent :
- le produit logiciel,
- le processus de développement du logiciel,
- la vérification du logiciel.

Les moyens de vérifier que ces exigences sont effectivement respectées et efficaces ont été consignés dans le document « Guide to assessing software quality and safety requirements » [12].

■■ À éliminer ces fautes, en proposant une *démarche de construction des tests du logiciel*. Une démarche didactique de construction des tests du logiciel a été étudiée, pour aider les concepteurs à respecter les exigences formulées en matière de certification et réaliser ainsi des tests efficaces et suffisants. Cette seconde voie a conduit à la rédaction du document « Guide for the construction of software tests » [13].

Cet article est dans la continuité de celui publié en 1997 dans cette revue [9], qui présentait les exigences en matière de qualité et de sûreté des logiciels. Il présente tout d'abord la problématique liée à la vérification du logiciel, en focalisant sur son principal composant, c'est-à-dire le test. Les principales étapes de la démarche de construction des tests du logiciel sont ensuite présentées :
- définition préalable des tests à réaliser,
- exécution des tests,
- vérification des résultats de tests.

Les objectifs de chaque étape sont indiqués, ainsi que les activités à y exercer et les traces à fournir pour aider à l'analyse. Des exemples d'application illustrant certains points de la démarche sont donnés pour la construction des tests de validation. Une annexe rappelle les exigences du document [10] relatives à la vérification du logiciel.

1. La vérification du logiciel

1.1. Généralités

La vérification du logiciel a pour but de démontrer que les produits logiciels issus d'une phase du cycle de développement sont conformes aux spécifications (incluant les exigences légales et réglementaires) établies lors des phases précédentes.

Elle a également pour but de détecter et de rendre compte des fautes qui peuvent avoir été introduites au cours des phases précédant la vérification.

La vérification du logiciel est composée :
- des tests, partie prédominante pour les logiciels de faible taille,
- des activités de revues et d'analyse. Ces activités peuvent dans certains cas remplacer certains tests (exemple : un test qui ne pourrait pas être réalisé sans détérioration d'un composant matériel).

1.2. Le test du logiciel

Le test du logiciel [14 à 16] est une approche dynamique de la vérification, destinée à s'assurer que ce logiciel possède effectivement les caractéristiques requises pour son contexte d'utilisation. La première action à entreprendre est donc de décrire avec précision ce contexte, en particulier les fonctionnalités attendues, les contraintes d'environnement, ou encore les situations dangereuses à considérer.

Le test a pour objectifs :

1. De détecter d'éventuels écarts entre le comportement attendu et le comportement observé au cours des tests, ce qui élimine un grand nombre de fautes présentes dans le logiciel.

2. D'obtenir la confiance nécessaire avant l'utilisation opérationnelle. Il faut cependant noter que le nombre de fautes détectées ne peut pas être considéré comme un critère de réussite des tests. Le retour d'expérience montre en effet qu'à complexité technique et industrielle constante, un grand nombre d'erreurs détectées par rapport à d'autres projets « de référence » peut seulement être interprété comme l'indicateur d'un logiciel contenant un très grand nombre de fautes et non comme l'atteinte d'un bon taux de détection des fautes présentes. Il est donc très difficile d'avoir confiance en un logiciel ayant un grand nombre de fautes détectées par le test.

Remarques

■ L'observation d'un logiciel sous test, l'analyse de l'architecture du logiciel et du codage, l'évaluation du processus de développement sont autant d'éléments pour répondre à la question : « Est-ce que le logiciel est bien fait ? ».

Mais, pour traiter complètement la sûreté de fonctionnement au niveau du système, il faut être également en mesure de répondre à la question : « Est-ce que le logiciel réalisé correspond bien à l'application à traiter ? ». Seules des analyses au niveau du système permettent de répondre à cette question, ce qui dépasse l'étude des tests du logiciel.

Il faut donc insister sur le fait qu'il est illusoire de qualifier un système « complexe » à la seule observation de son comportement sur un nombre limité de jeux de tests. Les tests sont essentiels pour détecter des erreurs et améliorer la confiance dans l'aptitude du système à accomplir sa mission, mais insuffisants pour « qualifier » le comportement d'un système.

(1) STSARCES : STandards for SAfety Related Complex Electronic Systems. Contrat CEE n° SMT4-CT97-2191. Financement DG XII.

(2) Veridatas est une entité du pôle Conseil du bureau Véritas.

(3) Faute logicielle systématique : faute introduite à la conception d'un logiciel, qui se révèle lorsque la partie du programme dans laquelle elle est située est activée.

(4) Logiciel partie intégrante du système, fourni par le vendeur et non accessible pour modification à l'utilisateur final. A distinguer du logiciel applicatif, qui est un logiciel spécifique à l'application utilisateur, dans lequel les fonctions du système sont programmées.

■ Il existe un risque pour le concepteur de faire définir par une même entité (individu, équipe), la spécification, la conception, la stratégie de tests et les cas de tests. Deux raisons au moins justifient le recours à un tiers pour tester un logiciel :
• le but des tests est d'exécuter un programme avec l'intention de trouver ses erreurs [14], ce qui constitue un processus mental non naturel, difficile à mener par une même entité ;
• le programme peut contenir des erreurs dues à la non compréhension de l'implémentation ou des spécifications par le développeur [14]. Dans ce cas, il est probable que celui-ci aura ces mêmes « non-compréhensions » quand il testera son propre programme (mode commun de défaillance).

■ La correction des fautes logicielles peut :
• injecter de nouvelles fautes et perturber des parties correctes déjà testées ;
• rendre active une partie du logiciel jusqu'alors inaccessible et donc révéler un grand nombre de nouvelles fautes, qui ne pouvaient être constatées avant cette correction.

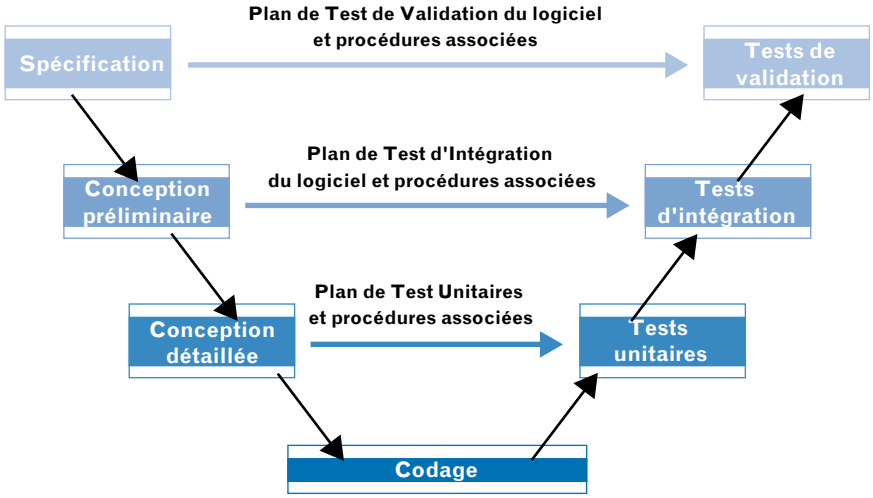

Fig. 1. Phase de définition des tests dans le cycle de « développement en V » - Test definition phase in the « V-development » cycle

1.3. Les différents tests du logiciel

Les tests doivent être menés à différents niveaux du développement d'un logiciel. En suivant le « cycle en V » de développement du logiciel *(fig. 1),* ils se décomposent en :

■■ Tests unitaires, pour démontrer que chaque module effectue toute la fonction prévue et seulement cette fonction. On peut distinguer dans ces tests unitaires :
- les tests de logique (recherche d'erreur, vérification de l'enchaînement correct des branches parcourues) ;
- les tests de calcul (vérification des résultats des calculs, des performances, de l'exactitude des algorithmes).

Typiquement, les tests de calcul comprennent les tests de données dans les limites des spécifications (état normal), aux limites spécifiées et en dehors de ces limites (état anormal). Les tests de comportements anormaux (hors limites, erreurs) sont généralement appelés tests de robustesse.

■■ Tests d'intégration du logiciel, pour démontrer le bon fonctionnement d'unités fonctionnelles constituées d'un assemblage de modules. Ils portent principalement sur la vérification des enchaînements entre modules, la circulation des données, les aspects dynamiques, les séquences d'événements prévus et les reprises en cas d'interruption.

■■ Tests de validation, pour s'assurer que le logiciel implanté dans le matériel répond aux spécifications fonctionnelles, en vérifiant plus particulièrement les fonctions générales, les interfaces matériel / logiciel, le fonctionnement temps réel, les performances, l'utilisation et l'allocation des ressources.

Cette décomposition du test, reliée au cycle de vie, distingue les tests de validation du logiciel de ceux du système. Elle montre ainsi clairement qu'il est nécessaire de vérifier d'une part, la conformité du logiciel à ses exigences propres et d'autre part, la conformité du système à ses exigences (les premières étant déduites des secondes). Cette distinction ne doit cependant pas conduire à valider le logiciel indépendamment du matériel sur lequel il est implanté.

1.4. Exigences pour la vérification du logiciel

Les exigences à satisfaire pour la vérification du logiciel, qui portent sur les activités de revues et d'analyse ainsi que sur le test, sont présentées en *annexe I.* Leur respect limite le nombre des fautes introduites dans un logiciel.

Elles servent à définir, vis-à-vis d'un concepteur, des résultats à obtenir pour toutes les activités techniques liées à la vérification du logiciel. Il est donc recommandé au concepteur du système de prendre en compte ces exigences dès le début du développement du logiciel.

Les exigences sont classées en deux niveaux, suivant la criticité du logiciel. Ces niveaux déterminent le degré de rigueur exigé pour le développement du logiciel, afin d'éviter les fautes dont le logiciel peut être la cause. Le processus de classification pour fixer le niveau des exigences au logiciel nécessite une évaluation spécifique de chaque système *(cf. chap. A.1 de l'annexe I).*

La démonstration du respect de toutes les exigences applicables au logiciel à évaluer doit être faite par le concepteur du système. Chacune de ces exigences élémentaires doit faire l'objet d'une réponse appropriée, soit par l'intermédiaire des documents produits, soit par l'intermédiaire des activités conduites pour développer le logiciel et dont les traces auront été conservées pour évaluation. Il est par exemple important de noter qu'une activité de test qui serait réalisée de façon informelle, par exemple à l'aide d'un analyseur logique sans aucune trace papier ou informatique, ne constitue pas une preuve de test. Ceci n'empêche pas le concepteur de mener cette activité si elle lui est nécessaire dans une phase spécifique de mise au point.

2. La démarche de construction des tests du logiciel

Ce chapitre propose une démarche pour tester efficacement un logiciel. Son suivi facilitera le respect des exigences formulées en annexe I et permettra d'éviter au maximum l'introduction de fautes au sein d'un logiciel. Les activités liées aux trois phases de tests décrites au § 1.3 sont identiques.

Le processus de test, pour qu'il soit de qualité, doit suivre les mêmes principes que tous les autres processus de réalisation ou de création, c'est-à-dire :

- définition au préalable des tests à réaliser,
- exécution selon des procédures prévues,
- vérification selon des modalités fixées.

2.1. Définition préalable des tests à réaliser

Les tests se préparent dès la phase de spécification - conception correspondante. Leur définition fait l'objet des plans de tests (de validation, d'intégration, unitaires). Elle nécessite la rédaction de procédures associées.

La figure 1 présente, pour un « cycle de vie en V », les phases de définition des tests.

2.1.1. Objectifs de tests

Définition des objectifs

Dans l'absolu, l'effort de test (validation, intégration et unitaire) d'un logiciel peut être aussi important que l'on veut. La combinatoire des données d'entrée, des modes d'utilisation, des modes de fonctionnement, des paramètres d'exploitation, la variété des aspects vérifiables (conformités aux spécifications, aux manuels d'utilisation et d'exploitation, aux exigences de robustesse, de performance, d'ergonomie, comportement nominal, en cas de défaillance, etc.) sont telles qu'une infinité de tests peut être menée sans qu'on soit certain de la conformité totale aux exigences.

En pratique, l'effort de test doit donc être adapté aux enjeux. Il nécessite de se fixer des objectifs préalablement à toute définition d'une stratégie, de méthodes et techniques, etc., et finalement des tests eux-mêmes.

Définir ces objectifs permet de réaliser les tests avec un maximum d'efficacité en garantissant la couverture des objectifs définis, donc de focaliser les efforts sur les aspects essentiels.

Les *objectifs de tests* de chaque phase de tests seront identifiés et définis en considérant :

- les exigences fonctionnelles et de performance (en tenant compte aussi des contraintes particulières de l'environnement final d'utilisation du logiciel), exprimées dans les documents de spécification / conception du produit logiciel ;
- les exigences en matière de qualité et de sûreté de fonctionnement (rectitude, robustesse, maintenabilité, disponibilité, sécurité...) ;
- toutes les contraintes identifiées d'implémentation (utilisation d'une base de données, utilisation des nombres réels, langage de programmation, logiciel temps réel…) ;
- les exigences légales et réglementaires ;
- les contraintes de coût et de temps de test.

Il en résulte la définition des types de tests à réaliser (chemins structurels, chemins fonctionnels, données d'entrées - sorties aux limites, hors limites, de robustesse, de performance...) et la justification de ces choix.

Taux de couverture

Pour chaque type de tests défini, l'exigence sur le taux de couverture des tests à atteindre est de 100 %.

Exemple

Si le choix de tester les chemins fonctionnels est retenu, un taux de couverture de 100 % signifie qu'après réalisation des tests, chacun des chemins fonctionnels du logiciel doit avoir été parcouru au moins une fois. Il est bien sûr nécessaire d'avoir les moyens de mesurer le taux de couverture réel obtenu pour le comparer au taux de couverture souhaité (ici, de 100 %).

Mais ce taux peut être réduit par des hypothèses à justifier :

■ Il se peut par exemple, lors des tests de « validation site », que le logiciel soit dans un environnement particulier, empêchant le test de certaines fonctionnalités prévues pour un environnement différent. Le taux de couverture ne pourra donc pas atteindre 100 % des fonctionnalités décrites dans les spécifications. La justification de la diminution du taux de couverture devra être consignée dans le dossier de test.

■ De même, certains tests portant sur les aspects « temps réel » ne sont pas toujours réalisables en tests unitaires. Une justification adaptée permettra de limiter le taux de couverture des aspects temps réel durant la phase de tests unitaires.

Attention ! Fixer a priori un objectif en terme de taux de couverture peut conduire le concepteur à dégrader son code pour atteindre plus facilement les objectifs. Une valeur de 100 % pour un taux de couverture en branches ou en appels de fonctions peut être obtenue en diminuant « artificiellement » le nombre d'appels de procédures par duplication du code ou allongement de la taille moyenne des procédures.

Toute démarche d'objectif de taux de couverture doit donc être accompagnée de règles de programmation évitant ce type d'effet pervers.

2.1.2. Stratégie de test

Quel que soit le niveau d'exigence, les tests doivent être conçus et réalisés sur la base d'une stratégie. Elle constitue un préalable nécessaire à l'élaboration des tests et à la conception des jeux d'essai. La stratégie de test témoigne d'une réflexion sur les tests. *Son absence indique un risque d'improvisation dans l'élaboration des tests.*

La stratégie doit définir comment, globalement, les tests vont être menés pour satisfaire aux objectifs préalablement définis, en relation avec les environnements que le concepteur compte mettre en œuvre. Elle doit organiser les tests en une structure lisible d'étapes, ayant des objectifs clairs ou répondant à des contraintes d'environnement précises (par exemple, relatives à la disponibilité d'un outil).

On définit donc les *méthodes* et les *moyens de tests* pour y parvenir, puis l'enchaînement - ou planification - des tâches de réalisation des tests. L'ensemble des moyens de tests et de la *planification* constitue la *stratégie de test* mise en œuvre.

La définition des tests aboutit à la réalisation des *fiches de tests*.

Définition des procédures de test

Selon l'exigence 11 : « Les directives pour la rédaction des procédures de test doivent comprendre la description des jeux de données d'entrée à appliquer, la description des sorties attendues et les critères d'acceptation des résultats des essais », la définition des procédures de tests est exigée pour le niveau 2 et recommandée pour le niveau 1 *(cf. chap. A.3 de l'annexe).*

Un processus de test correctement mis en œuvre implique que les procédures de test soient réalisées le plus en amont possible dans le cycle de développement.

Cette remarque est particulièrement importante dans le cadre de l'évaluation. En effet, le valideur comme le concepteur auront toujours intérêt à ce que ces procédures soient évaluées avant leur dérou-

lement plutôt qu'après, ce qui évite de remettre en cause la phase correspondante de test et permet à l'évaluateur de ne pas concevoir de tests complémentaires lors de l'évaluation finale.

Il est nécessaire que ces procédures contiennent toutes les informations nécessaires :

- à leur identification et leur traçabilité vis-à-vis des objectifs et/ou de la stratégie ;
- à l'exécution du test pendant la phase (activités de préparation, de déroulement des tests, d'enregistrement des anomalies et critère d'arrêt des tests).

Méthodes et techniques de test : techniques de base

Les techniques de base sont réparties en deux catégories suivant qu'elles autorisent ou non l'observation du comportement interne du composant logiciel sous tests [16] :

- « boîte noire » ou test fonctionnel : pas de visibilité du composant sous test, les entrées et les résultats attendus sont exprimés en terme de comportement du composant logiciel d'un point de vue externe, sans se soucier de la structure interne du composant ;
- « boîte blanche » ou test structurel : visibilité du composant sous test, les jeux de test sont produits en analysant le code source.

Moyens de test

Quel que soit le niveau d'exigence, la définition de la stratégie et des méthodes de test doit s'accompagner de la définition des moyens nécessaires pour les mettre en œuvre. L'absence de cette information diminue fortement la crédibilité de la stratégie et des méthodes envisagées.

A titre d'exemple, la liste ci-dessous fournit des moyens auxquels il faut penser au préalable :

- outils : simulateurs, comparateurs, outils de mesure des signaux, outils de test, etc.,
- environnements de test : configurations différentes du système, en usine, sur site, éventuellement variable en fonction du paramétrage, etc.,
- jeux d'essai préenregistrés, tables prédéfinies de paramètres, etc.

En particulier, pour les tests unitaires, la définition d'une stratégie au plus tôt impose l'utilisation d'outils adaptés à la taille du logiciel. Ces outils automatisent l'exécution (et la réexécution éventuelle) des tests et la mesure des taux de couverture.

TABLEAU I

PRODUITS ISSUS DE LA DÉFINITION DES TESTS –
PRODUCTS STEMMING FROM THE TEST DEFINITION

Livrable « théorique »	Contenu
Planification des tests	Objectifs, stratégie, techniques et méthodes, organisation et responsabilités, moyens nécessaires, identification des tests et démonstration de couverture.
Procédures de tests	Description détaillée de chaque test en termes de mode opératoire, de données d'entrée, de résultats attendus et de critères d'arrêt.

2.1.3. Produits issus de la définition des tests

Les produits d'une phase de test (ou livrables) sont des documents ou des résultats sous forme informatique (fichier texte, objets d'un atelier de génie logiciel) observables comme preuve d'une activité du processus de test.

Outre les variations de cycle de vie entre les différents projets, il peut y avoir des différences plus ou moins importantes sur le nom et la répartition des informations entre ces livrables. Les livrables pourraient même ne pas exister. Les noms des livrables sont donc donnés à titre indicatif, avec une courte description de leur contenu : ils doivent être considérés comme des « livrables théoriques », qui sont disponibles dès que le concepteur peut fournir les informations qui les définissent. On vérifiera donc la présence des informations utiles et leur pertinence, plutôt que leur répartition dans une arborescence documentaire « théorique ».

Le *tableau I* donne le contenu souhaité pour les produits issus de la définition des tests.

2.1.4. Exemples de tests de validation

Objectifs de test

A priori, en validation, on doit se fixer comme objectif de *couvrir chaque élément de spécification,* y compris les mécanismes de sécurité. En ce qui concerne ces mécanismes, l'évaluateur devra s'assurer (EN 954-1 [2]) qu'au minimum les objectifs couvrent :

- les fonctions de sécurité spécifiées ;
- les comportements attendus pour la catégorie spécifiée ;
- les aspects dimensionnement et paramétrage.

De plus, il doit être possible de vérifier le comportement « temps réel » du logiciel dans tous les modes opérationnels.

Définition de la procédure des tests de validation

Pour montrer ce que peut être une stratégie de test de validation (une stratégie spécifique est à élaborer pour chaque contexte spécifique, et pour un contexte donné, il existe *n* stratégies possibles), la liste ci-dessous fournit des exemples d'étapes typiques. Ces étapes sont liées aux types de tests ainsi qu'aux contraintes de mise en œuvre des tests (banc de test, environnement…) :

- une étape de test détaillé pour chaque fonction en nominal ;
- une étape de test détaillé pour chaque fonction aux limites ;
- une étape de test détaillé pour chaque fonction hors limites ;
- une étape de test de performance ;
- une étape vérifiant le comportement en cas de défaillance ;
- une étape consacrée au paramétrage ;
- des étapes avec des entrées simulées et des étapes avec des entrées réelles du milieu opérationnel ;
- des étapes « usine » et des étapes sur un site opérationnel.

La stratégie doit s'intégrer dans la stratégie d'ensemble de vérification du logiciel. En pratique, il arrive que les essais ne couvrent pas totalement les objectifs fixés, par exemples :

- parce que certains défauts ou certains modes de défaillance ne peuvent pas être simulés, ou
- parce que certains états ne sont jamais atteints dans l'environnement du système.

La stratégie doit alors montrer quelles sont les vérifications complémentaires qui seront menées pour que le risque de non-conformité soit amené à un niveau raisonnable, par exemple : renforcement de certains tests en phase de tests unitaires ou d'intégration, contrôles théoriques, analyse de comportement sur la base de schémas ou de modèles.

Méthodes et techniques de test

Quel que soit le niveau d'exigence, des tests fonctionnels (en « boîte noire ») doivent être réalisés dans la phase de validation. En règle générale, il ne doit pas y avoir de tests en boîte blanche pour les raisons suivantes :

- les tests de validation doivent vérifier la conformité aux spécifications, donc à une définition du compotrtement du logiciel d'un point de vue externe ;
- la validation doit être réalisée sur le système intégré, donc en règle générale, avec peu de visibilité sur le comportement interne du logiciel.

Lorsque le concepteur prévoit d'utiliser des méthodes de type « boîte blanche », il faut démontrer qu'il y a un réel avantage à observer le comportement interne du logiciel pendant cette phase et que ces observations ne peuvent pas être réalisées dans les phases précédentes. Le cas échéant, il peut tolérer l'utilisation de cette technique si des tests en boîte noire existent et couvrent les spécifications fonctionnelles (ces tests peuvent éventuellement comporter aussi des « aspects boîte blanche »).

Les principales *techniques de base* utilisables en validation sont [13] :

- les tests aux limites de valeur (vérification du comportement du logiciel aux limites des domaines de définition des données d'entrée) ;
- les tests de domaine (contrôle que les donnés de sorties correspondant aux différentes classes d'équivalence définies sont conformes aux spécifications).

Ces techniques de base peuvent être complétées par d'autres techniques :

- moins formelles, comme les tests par intuition (recherche d'erreurs basée sur l'expérience) ;
- plus formelles, pour les hauts niveaux d'exigence comme les tests basés sur des graphes de cause à effet (identification, à partir d'un graphe représentant les relations logiques, des cas de tests qui ont une grande probabilité de détecter des fautes) ou les tests basés sur des automates à états finis (génération des ensembles de tests pour vérifier que le programme est conforme à ses spécifications fonctionnelles) ;
- nécessitant un outillage spécifique pour des objectifs particuliers, par exemple axés sur la mesure de fiabilité, comme les tests aléatoires (vérification du logiciel par la génération de données aléatoires issues de l'ensemble des tests possibles).

Cependant, ou bien ces techniques ne sont pas simples à mettre en œuvre (tests basés sur les graphes de cause à effet ou sur les automates à états finis), ou bien la démonstration de la couverture obtenue reste difficile à établir (tests par intuition, tests aléatoires).

Produits

Les livrables issus de la définition des tests de validation devront contenir les informations détaillées dans l'*encadré 1.*

2.2. Exécution des tests

L'exécution des tests doit se conformer aux procédures et aux conditions définies préalablement dans les plans de test et les fiches de test (activités de préparation, de déroulement des tests, d'enregistrement des anomalies).

Tout non-respect de celles-ci doit être argumenté, ce qui permettra, lors de la vérification des tests, de justifier ces divergences.

ENCADRE 1

Contenu des produits issus de la définition des tests de validation -
Content of the products stemming from the definition of the validation tests

- **Organisation, responsabilités**
  Personnes impliquées dans la validation. Organisation. Responsabilités.
- **Objectifs génériques des tests**
  Par exemple :
  - Couverture de toutes les fonctions ; de tous les modes de fonctionnement ; des exigences de sûreté en cas de défaut ; des exigences de performances ;
  - Conformité à la documentation d'utilisation, d'exploitation ;
  - Vérification d'endurance, etc.
- **Stratégie de tests**
  Démarche retenue, avec ordonnancement en étapes visant des objectifs spécifiques ou s'appuyant sur des environnements ou des techniques spécifiques, par exemples :
  - Test de tel groupe de fonctions ; test de tel mode de fonctionnement ;
  - Test d'enchaînement de telles fonctions, tests de bout en bout ; tests de paramétrage ;
  - Tests d'endurance ; tests avec simulation dans tel environnement, etc.
- **Méthodes et techniques de tests**
  Méthodes et techniques utilisées, par exemple : techniques manuelles ou automatiques, analytiques ou statistiques, tests de domaines, tests aux limites, etc.
- **Environnements et outils de tests**
  Identification des outils de tests utilisés (simulateurs, outils de mesures, comparateurs, etc.).
  Si l'environnement du logiciel n'est pas le même dans les différentes étapes, description de chacun de ces environnements.

TABLEAU II

TABLE DE TRAÇABILITÉ : OBJECTIFS - TYPES DE TESTS - NUMÉROS DE TESTS -
TRACEABILITY TABLE: OBJECTIVES – TYPES OF TESTS – TEST NUMBERS

Objectif élémentaire à vérifier	Types de tests TN	THL	TR	...
Objectif 1	1, 2		4, 5	
Objectif 2	6	3		
Objectif 3	7, 8	9		
Objectif 4	...			
Objectif ...				
Objectif n	m	m + 1		

TN : Test Nominal, THL : Test Hors Limite, TR : Test de Robustesse. 1, 2,..., m, m+1 : numéros des tests.

TABLEAU III

PRODUITS ISSUS DE L'EXÉCUTION DES TESTS -
PRODUCTS STEMMING FROM EXECUTION OF THE TESTS

Livrable « théorique »	Contenu
Rapport de tests	Conditions de réalisation des tests, éléments testés, tests exécutés, résultats obtenus, interprétation des résultats et bilan.

2.2.1. Table de traçabilité

Il est souhaitable d'établir une table de traçabilité lors de l'exécution des tests, pour démontrer :

- que les tests référencés par la table sont bien documentés par une procédure ;
- qu'il n'existe pas de tests qui ne soient pas référencés par la table.

Les tables de traçabilité fournissent une correspondance entre les tests réalisés et les objectifs définis pour ces tests. Le *tableau II* présente un format possible de table de traçabilité.

2.2.2. Critère d'arrêt des tests

Le critère d'arrêt des tests traduit la politique retenue concernant la poursuite des tests sur détection d'anomalie. Il doit être défini au préalable pour éviter de réaliser des tests inutiles, qui seront de toute façon à rejouer. Tout événement bloquant pour la suite correcte du processus de test, doit être considéré dans la définition de ce critère. Cela peut être la découverte d'un « bug » ou encore, d'une non-conformité majeure. La détection d'un événement constituant le critère d'arrêt des tests va entraîner l'arrêt du groupe de tests en cours avant son terme et le passage au groupe de tests suivant. Le groupe de tests arrêté sera à réaliser à nouveau après correction.

Remarques

- la durée des tests et les efforts engagés ne sont, en aucun cas, des critères d'arrêt des tests : aucune garantie sur le logiciel ne peut être établie sur ces seules observations.
- La diminution du nombre d'erreurs constatées par unité de temps n'est pas non plus un critère d'arrêt. Elle résulte souvent de la difficulté, pour l'équipe de tests, de trouver de nouveaux tests pour mettre en évidence des défaillances. La modification de l'équipe de tests, ou la mise en place de nouveaux outils, augmente souvent le nombre d'erreurs détectées. Seule la non-diminution doit être prise en compte comme un élément bloquant pour l'évaluation.
- Les mesures de taux de couverture pourraient être un critère d'arrêt s'il était possible d'évaluer la probabilité de présence de fautes non détectées. Elles seront cependant prises en considération comme un des éléments de l'évaluation.

En fait, l'arrêt des tests se fait suite à l'analyse des événements redoutés au niveau système et des aspects qui ne sont couverts par aucune des étapes de tests. De cette analyse, on déduit si le risque lié à cette « non-couverture » est acceptable et conforme aux exigences applicables.

2.2.3. Produits issus de l'exécution des tests

Les remarques du § 2.1.3 restent applicables aux livrables issus de l'exécution des tests. Le *tableau III* donne le contenu souhaité pour les rapports de tests.

2.2.4. Exemple des tests de validation

L'*encadré 2* détaille le contenu des livrables (rapports de tests) issus de la phase d'exécution des tests de validation. Les informations sur les objectifs de tests, les environnements et outils de tests utilisés, etc., sont requis pour une bonne compréhension des rapports de validation. On notera qu'il doit y avoir autant de rapports que de versions fournies.

ENCADRE 2

Contenu des produits issus de l'exécution des tests de validation - Content of the products stemming from execution of the validation tests

- **Objectifs des tests**
  Doivent être définis en tenant compte de la référence (identification des exigences, paragraphe d'un document, etc.)et forment une partie des objectifs généraux de la planification des tests.
- **Environnements et outils de tests utilisés**
  Peut être fourni par référence à la documentation de définition des tests. A compléter par les informations concernant l'étalonnage des appareils de mesure (données de calibration), le cas échéant, et par les écarts au plan de test.
- **Données et / ou signaux en entrée**
  A fournir avec leur séquencement et leur valeur. Si les tests sont automatisés, ces informations peuvent prendre la forme d'enregistrements numériques à condition de disposer des règles et moyens pour les interpréter.
- **Données et/ou signaux attendus en sortie**
  Idem ci-dessus.
- **Mode opératoire**
  Suite des actions à réaliser pour mener le test.
- **Critères de réussite**
  Implicitement, l'obtention des données et/ou signaux attendus est un des critères. Ce critère doit être complété lorsque nécessaire par d'autres critères, par exemples : temps de réponse, tolérance sur les valeurs en sortie, absence ou occurrence de tel événement avant tel laps de temps.
- **Essais réalisés**
  Peuvent être fournis par référence aux procédures de tests. A compléter par les écarts au plan de test, le cas échéant (tests non exécutés).
- **Chronologie des essais**
  Dates effectives des différents essais, ordonnancement des essais.
- **Faits marquants**
  Écart à l'ordonnancement prévus, abandons de certains tests, etc.. Peut faire l'objet d'un document séparé dit « journal de test ».
- **Détail des résultats obtenus**
  Pour chaque test, signaux et/ou données en sortie constatés (valeurs et séquencement) . Ces résultats peuvent prendre la forme d'enregistrements numériques, à condition que les règles et moyens de les interpréter soient disponibles.
- **Liens avec la gestion de modification**
  Pour chaque test en échec, lien vers la demande de modification correspondante.
- **Bilan des non-conformités**
  Liste synthétique des non-conformités détectées, avec leur impact sur la sécurité .
- **Preuve de couverture des tests**
  Par exemple, une liste des tests élémentaires et une référence croisée entre les exigences à tester et les tests identifiés.

2.3. Vérification des tests

La vérification des tests doit démontrer que les objectifs de test sont atteints. Elle consiste à analyser les produits de la phase à vérifier (voir exemple des tests de validation) pour :

■■ Etablir la traçabilité des tests réalisés pour chacun des types de tests définis, compte tenu des objectifs.

Il est nécessaire d'établir la traçabilité au fur et à mesure de la réalisation des tests afin d'optimiser la phase de vérification des tests pour laquelle cette traçabilité est nécessaire.

Si la matrice de traçabilité n'existe pas, le contenu de chaque procédure de tests doit être analysé pour établir la matrice. Cette évaluation doit être affinée en examinant sur le fond les procédures pour s'assurer que :

- les tests sont valides, pertinents et reproductibles ;
- les tests couvrant un objectif donné sont réellement probants pour l'objectif ;
- les méthodes de test spécifiées lors de la planification des tests sont bien utilisées.

■■ Evaluer le taux de couverture atteint par analyse des résultats de tests.

■■ Etablir l'acceptation ou le refus des résultats de la phase, en prenant en compte les justifications de divergences établies lors de l'exécution des tests.

Relations avec la gestion de configuration et des modifications

Lors de la vérification, la gestion de configuration et des modifications doit permettre de s'assurer que les tests nécessaires ont effectivement été réalisés :

• conformément à ce qui était prévu (c'est-à-dire par rapport à une version référencée des documents de planification de test et de définition des tests) ;

• sur la version de logiciel sur laquelle porte le processus d'évaluation.

CONCLUSION

Le logiciel « zéro défaut » n'existe pas. La présence de fautes logicielles systématiques introduites à la conception d'un dispositif programmé doit donc être considérée avec beaucoup d'attention, en particulier lorsque les conséquences de ces fautes peuvent influer sur la sécurité d'un dispositif.

La vérification du logiciel est un des moyens qui doit obligatoirement être mis en œuvre pour traiter ce problème et éviter l'introduction de fautes. Les exigences présentées en annexe permettent de s'assurer que cette vérification a effectivement été réalisée et qu'elle est efficace.

Les tests sont la composante principale de la vérification. Pour détecter un maximum de fautes, ces tests doivent être conduits en suivant une démarche qui débute par la définition préalable des objectifs de tests et de la stratégie adoptée pour atteindre ces objectifs, pour finir par la démonstration que ces objectifs sont satisfaits.

La réflexion induite par cette démarche impose au concepteur d'appréhender le processus test dans sa globalité, sans se focaliser sur la seule phase d'exécution. Cette réflexion constitue une condition préalable indispensable pour réussir les tests d'un logiciel (en termes d'efficacité de détection) et minimiser ainsi les risques d'apparition de défaillances dangereuses en exploitation.

BIBLIOGRAPHIE

1. Directive 98/37/CE du 22 juin 1998 relative au rapprochement des législations des États membres relatives aux machines. *Journal Officiel des Communautés Européennes, n° L. 207 du 23 juillet 1998, pp. 1-46.*

2. EN 954-1 – Sécurité des machines. Parties des systèmes de commande relatives à la sécurité. Partie 1 : Principes généraux de conception. *Paris - La Défense, AFNOR, juil. 1996, 46 p.*

3. VAUTRIN J.P., CHARPENTIER P., VIGNERON C. – La sécurité en machinerie. Introduction au concept de catégorie. *Cahiers de Notes Documentaires - Hygiène et sécurité du Travail, 1996, 163, Points de repère, pp. 255-262.*

4. Mode de défaillance des circuits intégrés : constats des problèmes posés. *Nanterre, Institut de Sûreté de Fonctionnement, , Document interne (groupe de travail MDCI), 1994, 15 p.*

5. CICCOTELLI J., BUCHWEILLER J.P. – Analyse et certification des composants de sécurité machines. *In : 10e colloque national de fiabilité et de maintenabilité (actes). Saint-Malo, oct. 1996, pp. 1088-1102.*

6. GEOFFRY J.C., MOTET G. – Sûreté de fonctionnement des systèmes informatiques. *Paris, Inter Editions, 1998, 349 p.*

7. LAPRIE J.C. – Guide de la sûreté de fonctionnement. *Toulouse, Éditions Cépaduès, mai 1995, 324 p.*

8. Projet européen STSARCES – Méthodes de validation du niveau de sécurité des systèmes électroniques. 4e PCRD. *Contrat n° SMT4-CT97-2191, nov. 1997, 28 p.*

9. CHARPENTIER P., MENAGER P. – L'évitement des fautes logicielles par la qualité. *Cahiers de Notes Documentaires - Hygiène et sécurité du Travail, 1997, 167, ND 2049, pp. 249-259.*

10. STSARCES Project (n° SMT4-CT97-2191) – Software quality and safety requirements. WP 1.2/1a Final Report. *Nancy, INRS, déc. 1999, 35 p.*

11. CEI 61508 – Sécurité fonctionnelle des systèmes électriques, électroniques, électroniques programmables relatifs à la sécurité. Parties 1 à 7. *Genève, CEI, 1998.*

12. STSARCES Project (n° SMT4-CT97-2191) – Guide to evaluating software quality and safety requirements. WP 1.2/1b Final Report. *Nancy, INRS, déc. 1999, 80 p.*

13. STSARCES Project (n° SMT4-CT97-2191) – Guide for the construction of software tests. WP 1.2/2 Final Report. *Nancy, INRS, déc. 1999, 50 p.*

14. MYERS G.J. – The art of software testing. *New York, John Wiley & Sons, 1979, 177 p.*

15. BEIZER B. – Software testing techniques, 2e éd. *Boston, International Thomson Computer press, 1992, 549 p.*

16. XANTHAKIS S., MAURICE M., DE AMESCUA A., HOURI O., GRIFFET L. – Test et contrôle des logiciels. *Nanterre, Editions EC2, 1994, 341 p.*

ANNEXE I

EXIGENCES DE VERIFICATION DU LOGICIEL - SOFTWARE VERIFICATION REQUIREMENTS

Cette annexe reprend les exigences contenues dans le projet STSARCES [10].

A1. DÉTERMINATION DU NIVEAU DE CRITICITÉ D'UN LOGICIEL

Les exigences du document « Software quality and safety requirements » sont modulées en deux niveaux (1, 2) selon le niveau de criticité des fonctions assignées au logiciel. Le niveau 2 correspond aux exigences les plus fortes pour les logiciels concernés par ce document.

La détermination du niveau de criticité du logiciel découle d'analyses de sûreté de fonctionnement de l'ensemble du système. Elle suit des principes largement fonction du domaine d'application, du type de système, des dommages qu'il peut causer à son environnement (humain en particulier), des personnes auxquelles il est destiné ou de la constitution même du système (architecture par exemple).

L'état de l'art actuel en matière de logiciel ne fournit pas de règles précises. Quelques idées directrices, schématisées par la *figure A-1,* peuvent néanmoins être fournies, afin de guider la détermination du niveau d'exigences à retenir.

• Classification du système : la structure du système étant définie ainsi que les conditions opérationnelles et environnementales, il s'agit d'identifier les types de dangers 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.

• Classification du logiciel : le produit logiciel destiné à assurer les (ou certaines) fonctionnalités du système étant défini, il s'agit de déterminer le niveau d'exigences à fixer en fonction de la classification du système.

De même que pour les aspects système, certaines décisions d'architecture du système ou de conception des logiciels sont à prendre en considération afin de déterminer si elles affectent le niveau d'exigences logiciel retenu.

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 d'une manière à pouvoir 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.

A2. EXIGENCES DE VERIFICATION

Le niveau ainsi déterminé permet d'établir la liste des exigences élémentaires applicables au logiciel considéré. Trois degrés d'exigences sont fixés pour retenir ou non une exigence en fonction du niveau :

• « E » *(Exigée) :* son application est systématique pour le logiciel concerné,

• « C » *(Conseillée) :* son application est recommandée sans être imposée,

• « / » *(pas d'exigence) :* son application est laissée à l'appréciation du concepteur.

Les textes en italique associés aux exigences sont des commentaires informatifs servant à préciser l'objectif de ces exigences, la manière de les appréhender et les limitations éventuelles d'application.

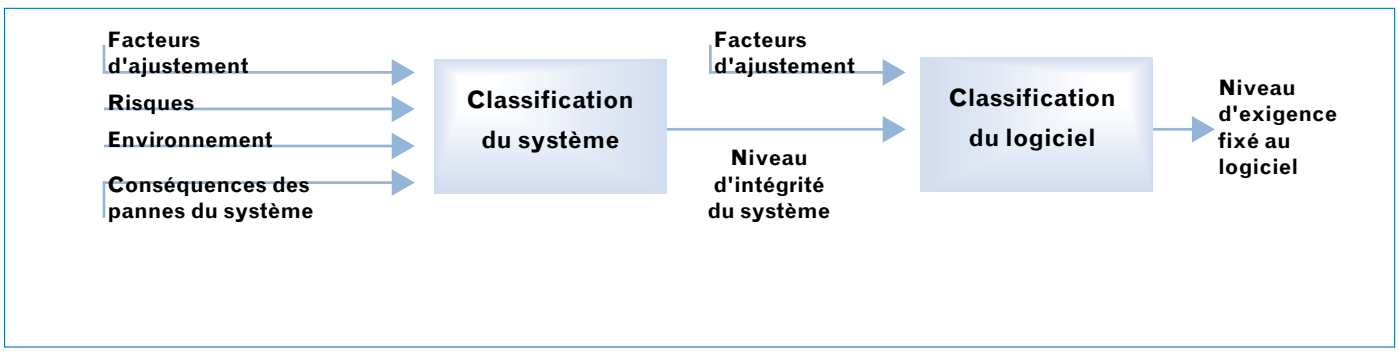

Fig. A-1. Processus de détermination du niveau de criticité d'un logiciel – Process of determining the software criticality level

A2.1. Exigences générales

Cf. tableau A-I : exigences nos 1 à 3.

TABLEAU A-I

EXIGENCES DE VÉRIFICATION - VERIFICATION REQUIREMENTS

Nº	Exigences générales de vérification	Niv. 1	Niv. 2
1	L'évaluation de la conformité du logiciel aux présentes exigences doit pouvoir être réalisée par l'analyste en procédant au cours des phases du développement à tous les audits ou expertises jugés utiles. Tous les aspects techniques des processus du cycle de vie logiciel sont sujets à évaluation par l'analyste. Tous les rapports de vérification (tests, analyses...) ainsi que les documents techniques utilisés durant le développement du logiciel doivent pouvoir être consultés par l'analyste. *L'intervention de l'analyste dès la phase de spécification est préférable à une intervention a posteriori pour limiter l'impact des décisions. Les aspects financiers ou humains du projet ne sont pas sujets à évaluation. L'intérêt du postulant est d'apporter tous les éléments de preuves sur les activités qu'il a mené pendant le développement du logiciel. L'analyste doit disposer de tous les éléments nécessaires pour formuler un avis.*	E	E
2	L'évaluation de la conformité du logiciel aux présentes exigences est faite pour une version spécifique et référencée du logiciel. Toute modification d'un logiciel précédemment évalué et ayant donné lieu à avis final de l'analyste doit être signalée à ce dernier en vue d'une concertation sur les activités d'évaluation complémentaires nécessaires pour réactualiser l'avis. *Toute modification est susceptible de modifier le comportement d'un logiciel, l'avis ne peut donc porter que sur une version précise du logiciel.*	E	E
3	Un rapport de vérification doit être établi pour chaque activité de vérification et doit identifier et documenter toutes distorsions (non-conformités) par rapport : - aux spécifications correspondantes, - aux règles ou normes (conception, codage...), - aux procédures d'assurance qualité, s'il y a lieu. *L'objectif de cette exigence est d'enregistrer les non-conformités identifiées afin de toutes les corriger (soit immédiatement s'il s'agit de non-conformités inacceptables d'un point de vue fonctionnel ou de la sécurité, soit ultérieurement, sur une autre version du logiciel, s'il s'agit d'une non-conformité mineure).*	C	E

A2.2. Exigences sur les revues internes

Les revues internes permettent au concepteur de s'assurer, à des points clés de l'avancement du développement, que le produit atteindra ses objectifs.

Cf. tableau A-I : exigences nos 4 à 7.

Nº	Exigences sur les revues internes	Niv 1	Niv. 2
4	Une revue (avec l'analyste) de spécification doit se tenir en fin de phase de spécification du logiciel. Les activités d'analyse et de vérification des spécifications du logiciel doivent : - vérifier l'exhaustivité et l'adéquation des spécifications du logiciel par rapport aux spécifications du système. - vérifier la traçabilité par rapport aux spécifications système. *La revue de spécification du logiciel doit permettre : de s'assurer que les besoins réels ont été pris en compte dans les spécifications, que les risques techniques ont été identifiés, résolus et réduits par de bons choix ; de vérifier que les spécifications du logiciel satisfont les spécifications du système. Ces activités permettent d'assurer que les spécifications du logiciel sont en cohérence avec les spécifications du système, qu'elles sont complètes et que l'on sait faire la correspondance entre les deux.*	C	E
5	Les activités d'analyse et de vérification de la conception du logiciel doivent vérifier la conformité avec les spécifications du logiciel. *L'objectif est d'assurer la cohérence entre la spécification et la conception (préliminaire et détaillée) du logiciel.*	E	E
6	Une revue externe (avec l'analyste) de validation doit se tenir en fin de phase de validation du logiciel. *Elle permet de savoir si le produit répond à ses spécifications. Si la validation du logiciel est sans réserve, le développement est clos.*	E	E

TABLEAU A-I (SUITE)

EXIGENCES DE VÉRIFICATION -
VERIFICATION REQUIREMENTS

N°	Exigences sur les revues internes (suite)	Niv. 1	Niv. 2
7	Le résultat de chaque revue doit être documenté et archivé. Il doit comprendre la liste des actions décidées en revue et la conclusion de la revue (décision ou non de passer à l'activité suivante). Les activités décidées en revue doivent être suivies et traitées.	E	E
N°	**Exigences sur la vérification du code (source et données)**	**Niv. 1**	**Niv. 2**
8	La vérification du code (analyse statique) doit assurer que le code est conforme ✓ aux documents de conception du logiciel ; ✓ aux règles de codage. *Cette exigence a pour objectif que la conception est mise à jour et cohérente avec le code (source et données).*	C	E

A2.3. Exigences sur la vérification du code

Cette vérification correspond à la première étape de vérification du code après son écriture. Il s'agit d'une vérification « statique », c'est à dire à base d'inspections, lectures croisées, etc. C'est ensuite que seront réalisées les étapes de vérification dynamique (tests unitaires, intégration, validation) qui constitueront le principal moyen de vérification.

Ces vérifications comprennent des vérifications du code et des données.

Cf. tableau A-I : exigence n° 8.

TABLEAU A-II

EXIGENCES SUR LE TEST DU LOGICIEL -
SOFTWARE TEST REQUIREMENTS

N°	Exigences générales sur le test du logiciel	Niv. 1	Niv. 2
9	La stratégie de vérification du logiciel aux différentes étapes de développement, ainsi que les techniques et outils à utiliser pour cette vérification doivent être décrits dans un plan de test avant de les mettre en œuvre. Cette description doit concerner a minima : - l'identification du logiciel ou des constituants logiciels relatifs à la sécurité soumis à validation avant mise en service, - l'organisation des activités de vérification (intégration, validation...) et les interfaces avec les autres activités de développement du logiciel, - l'indépendance de la vérification s'il y a lieu : la stratégie de vérification doit être élaborée, mise en œuvre et les résultats des essais doivent être évalués de manière indépendante (personne, service ou organisme) dans la mesure où la taille de l'équipe de développement le permet, - les méthodes et outils de vérification *(revue, analyse avec liste de contrôle, types d'essais...)* - l'environnement de vérification *(équipements de test, émulateur...)* - la manière de contrôler les résultats des essais. - matrice de traçabilité, fournissant une correspondance entre les tests à réaliser et les objectifs de tests définis. *Un seul document (plan de test par exemple) peut répondre aux exigences de planification de plusieurs activités de vérification (tests unitaires, intégration, validation). Ces documents peuvent, s'il y a lieu, faire référence à des procédures ou instructions générales applicables à tous les projets logiciels, en complément des dispositions spécifiques au projet.* *La formalisation de la stratégie a en outre l'avantage d'assurer la reproductibilité de l'activité (exemple : identifier l'ordre d'intégration en fonction de l'architecture système).* *L'intérêt de cette indépendance est d'introduire, dans l'activité de vérification, des personnes non impliquées dans les phases de développement amont et n'ayant donc pas la vision de la manière dont est réalisé le logiciel. Ceci assure en général une plus grande efficacité des tests.*	C	E

A3. EXIGENCES SUR LE TEST DU LOGICIEL

A3.1. Exigences générales

Il est important, avant d'écrire les premières fiches de tests, de décrire, dans un Plan de tests, une stratégie de tests indiquant la démarche retenue, les objectifs que l'on se donne en termes de couverture de tests, les environnements et techniques spécifiques qui seront utilisés, les critères de succès de test...

Les objectifs de tests doivent être adaptés au niveau d'intégrité de sûreté du logiciel, au type de logiciel, aux enjeux que le logiciel représente... ; en fonction de ces critères, on pourra déduire les types de tests à réaliser : tests fonctionnels, tests aux limites, tests hors limites, tests de performance, tests en charge, tests de pannes des équipements externes, tests de la configuration..., ainsi que l'ensemble des objets devant être couverts par les tests : tests des modes de fonctionnement, test des fonctions de sûreté, test de chaque élément de spécification...

Cf. tableau A-II : exigences nos 9 à 12, page suivante)

TABLEAU A-II (SUITE)

EXIGENCES SUR LE TEST DU LOGICIEL -
SOFTWARE TEST REQUIREMENTS

N°	Exigences générales sur le test du logiciel (suite)	Niv. 1	Niv. 2
10	La vérification d'une nouvelle version doit comporter des tests de non-régression. *Les tests de non-régression permettent d'assurer que les modifications effectuées n'ont pas modifié de manière inattendue le comportement du logiciel par des effets de bord.*	**E**	**E**
11	Les directives pour la rédaction des procédures de test doivent comprendre : - la description des jeux de données d'entrée à appliquer (valeur), - la description des sorties attendues (type, valeur), - les critères d'acceptation des résultats des essais (tolérance...). *Cette exigence implique que les tests réalisés seront documentés (sous un formalisme à définir par le concepteur).* *Cette documentation des tests permet d'optimiser les tests réalisés (ne pas rejouer plusieurs fois le même test) et de pouvoir rejouer ces tests ultérieurement (en tests de non-régression d'une nouvelle version du logiciel ou pour un autre projet similaire, pour lequel des ensembles logiciels sont réutilisés).*	**C**	**E**
12	Les tests formalisés par un compte rendu doivent pouvoir être rejoués (en présence de l'analyste). *Certains essais, nécessitant certains moyens spécifiques, peuvent cependant n'être rejouables qu'avec un préavis suffisant.* *Cette exigence donne la possibilité à l'analyste de s'assurer de la réalité de l'exactitude de tous les tests présentés lors de l'évaluation.*	**C**	**E**
N°	**Vérification des spécifications du logiciel**	**Niv. 1**	**Niv. 2**
13	La couverture des tests doit être explicitée dans une matrice de traçabilité et respecter les exigences suivantes : - chaque élément de spécification doit être couvert par un test de validation, y compris les mécanismes de sécurité, - il doit être possible de vérifier le comportement « temps réel » du logiciel dans tous les modes opérationnels. De plus, la validation doit être effectuée dans des conditions représentatives de l'utilisation. *Cette exigence permet notamment d'assurer que le logiciel réagit tel que fonctionnellement prévu. Elle ne s'applique pas aux cas où les conditions de test sont destructives pour le matériel (défaut physique non simulable d'un composant par exemple).* *Pour être significative, la validation doit se faire dans les conditions de fonctionnement opérationnel du système (c'est-à-dire avec le logiciel et le matériel dans leurs versions finales, le logiciel étant chargé dans le système cible). Toute autre combinaison peut diminuer l'efficacité des essais réalisés et nécessite une analyse de sa représentativité.*	**E**	**E**
14	Les résultats de la validation doivent être enregistrés dans un rapport de validation qui doit contenir a minima : - la version du logiciel et du système objet de la validation, - la description des essais de validation réalisés (entrées, sorties, procédures d'essai), - les outils et les équipements utilisés pour effectuer la validation ou évaluer ses résultats, - les résultats permettant d'établir si chaque essai de validation a abouti ou échoué, - le bilan de la validation : non-conformités identifiées, impact sur la sécurité, décision d'acceptation ou de refus de validation. Un rapport de validation doit être disponible pour chaque version de logiciel livrée et doit correspondre à la version finale de chaque logiciel livré. *L'existence de ce rapport permet d'apporter la preuve que les essais ont été joués et donnent des résultats corrects (ou comportent des écarts expliqués). Il permet aussi de rejouer ultérieurement ces tests pour une version ultérieure du logiciel ou pour un autre projet.*	**E**	**E**

A3.2. Exigences relatives aux tests de validation

Les tests de validation sont la composante principale de la vérification des spécifications du logiciel.

Cf. tableau A-II : exigences nos 13 et 14.

TABLEAU A-II (FIN)

EXIGENCES SUR LE TEST DU LOGICIEL -
SOFTWARE TEST REQUIREMENTS

N°	Vérification des spécifications du logiciel	Niv. 1	Niv. 2
14	*La seconde exigence a pour objectif de garantir que chaque version livrée a été validée, dans sa version ultime. Par contre, l'exigence n'impose pas une validation complète pour chaque modification incorporée dans une version : une analyse d'impact peut, dans certains cas, justifier une validation partielle.*		
N°	**Vérification de la conception du logiciel**	**Niv. 1**	**Niv. 2**
15	Les tests d'intégration du logiciel doivent permettre de vérifier : - le séquencement correct de l'exécution du logiciel, - les échanges de données entre modules, - le respect des performances, - la non altération des variables globales. La couverture des tests doit être explicitée dans une matrice de traçabilité fournissant une correspondance entre les tests à réaliser et les objectifs de tests définis.	C	E
16	Les modifications du logiciel, réalisées durant l'intégration du logiciel, doivent être analysées pour identifier l'impact sur les modules concernés et la nécessité éventuelle de réitérer certaines vérifications. *Tous les tests réalisés doivent être représentatifs du logiciel final. Une modification du code en cours d'essais peut invalider les résultats d'essais antérieurs.*	C	E
17	Les résultats de l'intégration doivent être enregistrés dans un rapport de test d'intégration du logiciel, qui doit contenir a minima : - la version du logiciel intégré, - la description des tests réalisés (entrées, sorties, procédures), - les résultats des tests d'intégration et leur évaluation.	C	E
N°	**Vérification de la conception détaillée**	**Niv. 1**	**Niv. 2**
18	Chaque module logiciel doit être soumis à des essais vérifiant au moyen de données fournies en entrée, que le module remplit les fonctions demandées dans la conception détaillée. La couverture des tests doit être explicitée dans une matrice de traçabilité fournissant une correspondance entre les tests à réaliser et les objectifs de tests définis.	/	C
19	Les résultats des essais du module doivent être consignés dans un rapport comprenant a minima : - la version du module en essai, - les entrées appliquées, - les résultats attendus et obtenus, - l'évaluation des résultats (essai positif ou non). *L'objectif est d'assurer que chaque module est vérifié dans sa dernière version et que l'essai sera reproductible (au titre de la non-régression par exemple).*	/	C

A3.3. Exigences relatives aux tests d'intégration

L'objectif de cette vérification est centré sur le bon assemblage des modules et sur les relations mutuelles entre les composants logiciels. Elle permet de révéler les erreurs du type :

- initialisation incorrecte des variables et des constantes,
- erreurs dans le passage de paramètres,
- altération de données, en particulier de données globales,
- résolution numérique de bout en bout inadéquate,
- séquencement incorrect d'événements et d'opérations.

Les tests d'intégration du logiciel sont la composante principale de cette vérification.

Cf. tableau A-II : exigences nos 15 à 17.

A3.4. Exigences relatives aux tests unitaires

L'objectif des tests unitaires est centré sur le module logiciel et sa conformité par rapport à la conception détaillée. Cette activité, indispensable pour des logiciels complexes et de taille importante, est seulement conseillée pour les logiciels de petite taille auxquels s'adresse le présent document. Cette conformité peut également être démontrée par des techniques statiques (relecture de code par exemple).

Cette phase de vérification permet de révéler les erreurs du type :

- inaptitude d'un algorithme à satisfaire une spécification du logiciel,
- opérations de boucle incorrecte,
- décision logique incorrecte,
- inaptitude à traiter correctement des combinaisons valides de données d'entrées,
- réponses incorrectes à des données d'entrées manquantes ou altérées,
- violation de limites de tableaux,
- séquence de calcul incorrecte,
- précision, justesse ou performances inadéquates d'un algorithme.

Cf. tableau A-II : exigences nos 18 et 19.

■

INSTITUT NATIONAL DE RECHERCHE ET DE SÉCURITÉ - 30, rue Olivier-Noyer, 75680 Paris cedex 14

Tiré à part des Cahiers de notes documentaires - Hygiène et sécurité du travail, 4e trimestre 2000, n° 181 - ND 2140 - 1 200 ex.
N° CPPAP 804 AD/PC/DC du 14-03-85. Directeur de la publication : J.-L. MARIÉ. ISSN 0007-9952 - ISBN 2-7389-0875-6

JOUVE - Paris