dgac

STAC

Évaluation et maîtrise de la sûreté de fonctionnement des logiciels

État de l'art - Avril 2009

Ressources, territoires et habitats
Énergie et climat
Développement durable
Prévention des risques
Infrastructures et transports

Présent pour l'avenir

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

Ministère de l'Écologie, de l'Énergie, du Développement durable et de l'Aménagement du territoire

service technique de l'Aviation civile

www.stac.aviation-civile.gouv.fr

GESTION DOCUMENTAIRE

Validation du document

Nom	***Responsabilité***	***Date***	***Visa***
Pelletier Céline Chargée d'études	Rédacteur	27.04.09	*(Signé)*
Frédéric Le Tennier Chef du programme Sûreté de Fonctionnement	Vérificateur, Approbateur	27.04.09	*(Signé)*

Diffusion du document

Type de diffusion : électronique.

Gestion des versions

Version	***Date***	***Synthèse des évolutions***	***Auteur(s)***	***Page(s)***
1.0	27.04.09	Version initiale.	C. Pelletier	Toutes

Sommaire

Table des illustrations

1. OBJECTIF DE L'ETUDE

La prise en compte des logiciels dans les études de sécurité menées par les PSNA (Prestataires de Services de Navigation Aérienne) est une nouvelle composante introduite par le règlement CE 482/2008 paru le 1er mai 2008. Si, jusqu'à présent, ces études s'attachaient à démontrer que les systèmes matériels étaient fiables, elles n'intégraient pas toujours de démonstration d'une confiance justifiée dans le fait que les logiciels étaient sûrs. Cette composante est rendue obligatoire par la parution de ce nouveau règlement (CE 482/2008).

Le but de cet état de l'art est de faire un point, non seulement sur les méthodes existantes d'évaluation de la sûreté de fonctionnement des logiciels (détermination d'une fiabilité logicielle ou approche par les processus), mais également sur les outils permettant de donner la confiance justifiée de cette sûreté.

2. GLOSSAIRE

- Système : Un système est entendu ici en tant que système matériel, et non pas en tant que système global (comprenant à la fois le matériel, le logiciel et l'humain.)
- Conception : On entend par conception d'un logiciel la phase en amont de son utilisation opérationnelle. Cette phase comprend entre autre les phases de définition, réalisation, codage…
- Cycle de vie : C'est le délai compris entre la décision de produire ou de modifier un produit logiciel et le moment où ce produit est retiré du service.

3. INTRODUCTION

La sûreté de fonctionnement peut être définie ainsi : « Propriété qui permet aux utilisateurs du système de placer une confiance justifiée dans le service qu'il leur délivre ». [LAPRIE1]

Le diagramme suivant issu de [QUASSI1], illustre bien la façon dont s'articulent les différents concepts de la sûreté de fonctionnement.

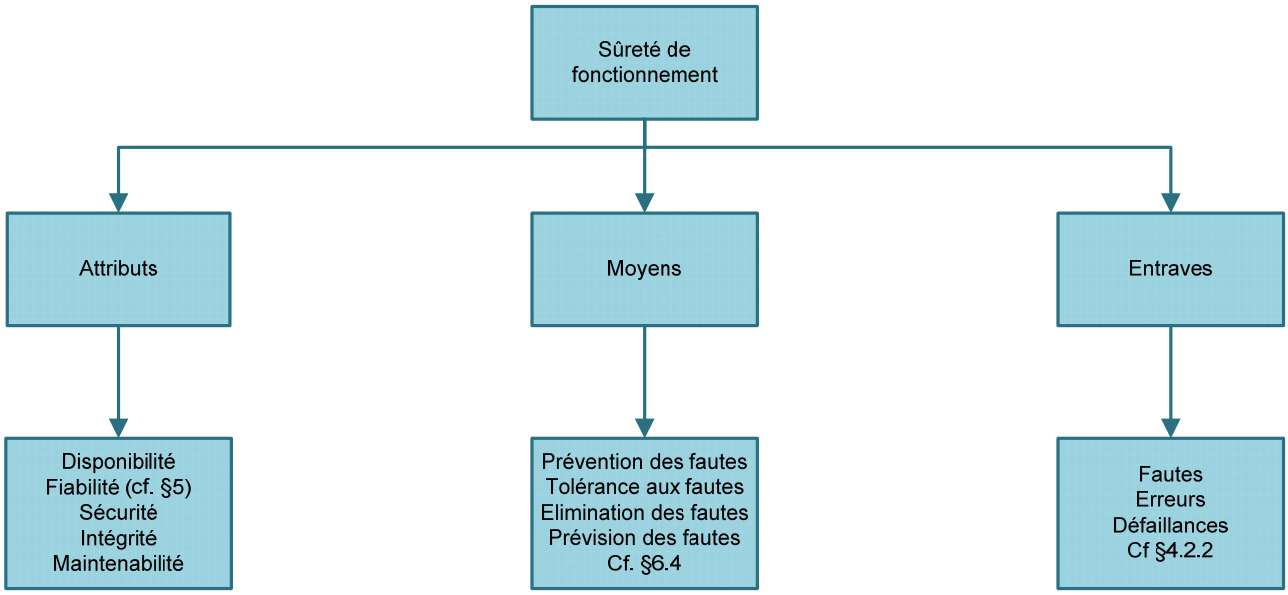


Figure 1: Schéma conceptuel de la notion de sûreté de fonctionnement

Si la sûreté de fonctionnement des logiciels a le même but d'obtenir une confiance justifiée dans le service délivré par le logiciel, deux axiomes permettent de comprendre pourquoi ils ne peuvent être traités de la même façon que le matériel :

1. Un matériel répond à une sollicitation donnée avec une imprécision statistiquement estimable et qui évolue dans le temps (vieillissement...) alors que le logiciel répond de façon précise à une sollicitation donnée et ce de façon stable.
2. Les défaillances matérielles sont assimilables à des phénomènes aléatoires alors que le comportement d'un logiciel est purement déterministe et qu'une défaillance logicielle provient systématiquement d'une faute de conception.

Ainsi, la sûreté de fonctionnement logicielle nécessite la mise en œuvre de moyens différents de ceux utilisés par la sûreté de fonctionnement système. Nous allons donc tout d'abord voir comment se positionne la sûreté de fonctionnement logicielle avant de détailler les différentes approches qui peuvent être utilisées pour l'évaluer et/ou la maîtriser.

4. POSITIONNEMENT DE LA SÛRETÉ DE FONCTIONNEMENT LOGICIELLE

4.1. PRÉAMBULE

Malgré toutes les méthodes mises en place en termes de processus de conception, développement et tests, il est impossible d'assurer qu'un logiciel ne contient aucun défaut. De ce fait, contrairement au domaine des systèmes (réduit au niveau matériel) où l'on recherche la fiabilité du matériel, dans le domaine logiciel, l'approche consiste à mesurer la « confiance » qui peut être attribuée au logiciel. Dès lors on trouve deux approches :

- La première qui cherche à évaluer la fiabilité prévisionnelle des logiciels.
- La seconde qui considère qu'il est impossible de déterminer la fiabilité prévisionnelle d'un logiciel donc qu'il faut mettre en place des processus permettant d'avoir confiance dans le fait que le logiciel est suffisamment sûr (approche qualité, maîtrise des processus).

L'approche fiabiliste ne permet d'obtenir au mieux qu'une estimation, elle ne dispense donc pas de la mise en place de processus destinés à donner confiance dans le fait que cette fiabilité supposée s'avérera correcte.

L'approche maîtrise du cycle de vie des logiciels, quant à elle, n'est pas évidente à définir. En effet, de part leur caractère abstrait, il n'est pas aisé de déterminer au préalable ce qu'est le niveau acceptable de qualité des logiciels.

Avant de voir comment peut être évaluée la fiabilité d'un logiciel, nous allons donner les définitions des principaux concepts de la sûreté de fonctionnement logicielle.

4.2. LES CONCEPTS

4.2.1. Système logiciel

D'après [Villemeur]: le logiciel peut être considéré comme un système. Les composants sont, par exemple, les instructions ou les modules qui le constituent. Un module est alors dénommé « composant logiciel ». Une caractéristique importante du système logiciel est d'être un système immatériel et abstrait.

D'après le règlement [CE 482/2008] un logiciel est « les programmes informatiques et les données de configuration correspondantes, y compris les logiciels prédéveloppés, à l'exclusion des éléments électroniques tels que les circuits intégrés spécifiques d'une application, les réseaux de portes programmables ou les dispositifs de contrôle de logique sur support physique ».

D'après l'[ED-12B], un logiciel est un programme informatique et, éventuellement, la documentation et les données associées concernant le fonctionnement d'un système informatique.

On retiendra donc que lorsqu'on parle d'un système logiciel, on entend non seulement les séries d'instructions (organisées ou non en modules) mais également les données de configuration qu'elles utilisent.

4.2.2. Erreur / défaillance / faute

Comme nous l'avons déjà dit, contrairement au matériel, une défaillance logicielle est toujours due à une erreur humaine en conception (phase préalable à la phase d'exploitation). La distinction entre les notions

d'erreurs, défauts, défaillance va permettre de savoir à quel niveau de la chaine causale il est possible d'agir. Ainsi on peut retenir les définitions suivantes :

- Faute : toute cause (évènement, action, circonstance) pouvant générer un défaut, conséquence d'une défaillance lors de la conception du système.
- Défaut : écart entre une caractéristique du logiciel et la caractéristique voulue.
- Erreur : état ou partie de l'état du système susceptible de provoquer une défaillance (partie de l'état interne dont les propriétés courantes ne sont pas conformes aux spécifications). Une erreur est susceptible de provoquer une défaillance mais ne la provoque pas nécessairement (ou pas immédiatement) ; on dit d'une erreur qui n'a pas provoqué de défaillance qu'elle est latente. L'erreur est un état atteint par le système qui n'avait pas été spécifié, un état que le système n'était pas supposé atteindre.
- Défaillance : cessation de l'aptitude d'un logiciel à accomplir une fonction requise. Cette dernière s'exerce dans des conditions données qui sont, par exemple, la définition du système informatique pour lequel il a été conçu et les conditions prévues dans ses spécifications.

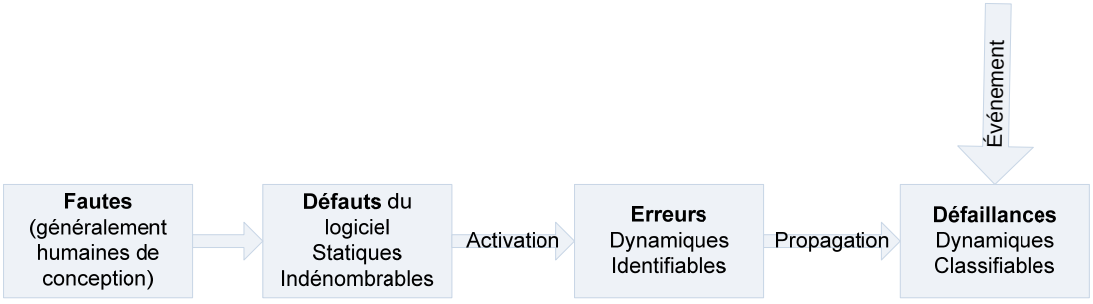


Figure 2: Articulations des concepts Fautes/ Défauts/ Erreur/ Défaillances

On peut également définir la notion de mode de défaillance : effet par lequel une défaillance du logiciel est observée, par exemple :

- Arrêt du système d'exploitation ;
- Arrêt du programme avec l'impression d'une explication claire ;
- Arrêt sans diagnostic ;
- Le programme tourne mais fournit des résultats incohérents ;
- Le programme tourne en donnant des résultats apparemment corrects pour la configuration d'entrée choisie mais erronés en réalité.

5. APPROCHE FIABILISTE

5.1. PROBLÉMATIQUE

La fiabilité d'un logiciel est son aptitude à accomplir une fonction donnée, dans des conditions données, pendant une durée donnée.

Considérant le caractère déterministe du fonctionnement d'un logiciel, il est possible de définir la fiabilité de la façon suivante :

- Soit on considère que le logiciel ne contient pas de défaut et dans ce cas sa fiabilité est de 1 ;
- Soit on considère qu'il en contient et alors :
  - dans les configurations où le défaut se réalise sa fiabilité est de 0 ;
  - et dans les autres configurations sa fiabilité est de 1.

Cependant le nombre de défauts varie au cours de la vie du logiciel lorsqu'après une défaillance, le logiciel est corrigé ; soit :

- On augmente la fiabilité en diminuant le nombre de défauts ;
- On diminue la fiabilité en introduisant de nouveaux défauts.

On comprend donc que cette vision simpliste de la fiabilité du logiciel ne permet pas d'obtenir des résultats satisfaisants. C'est pourquoi de nombreux modèles de calcul ont été mis au point pour tenter de définir la fiabilité d'un logiciel.

5.2. EVALUATION DE LA FIABILITÉ LOGICIELLE

Cette démarche vise à mettre en place des métriques, qui permettent d'obtenir un chiffre vis-à-vis de la fiabilité prévisionnelle.

La fiabilité logicielle peut être évaluée a priori par rapport à la complexité du logiciel ou par rapport aux tests menés mais il existe aussi des modèles dits de croissance de fiabilité destinés à évaluer la fiabilité du logiciel au cours de son cycle de vie.

5.2.1. Par rapport à la complexité

On peut rapporter la fiabilité à trois types de complexité :

- La complexité du texte est basée sur les considérations de linguistiques mathématiques et de résultat de psychométrie. Elle permet de déterminer le nombre N de défauts contenus dans un programme de complexité donné (technique développée par Halstead). On mesure l'effort mental W exigé par la fabrication d'un programme (en fonction du volume et de la longueur de ce programme) et empiriquement on trouve la relation suivante :

$$N = \frac{W^{2/3}}{3200}$$

$$avec \begin{cases} N \ Nombre\ de\ défauts \\ W \ effort\ mental \end{cases}$$

- La complexité de la structure est fondée sur les caractéristiques du graphe associé au programme. Ce graphe orienté a une entrée et une sortie unique et chaque sommet correspond à un ensemble

d'instructions purement séquentiel. Un arc joint un sommet x à un sommet y si les instructions représentées en x précèdent directement celles représentées en y. La complexité du programme correspond alors au nombre cyclomatique de ce graphe.

- La complexité vis-à-vis du comportement à l'exécution pour laquelle l'analyse consiste alors à adjoindre à l'analyse du comportement du programme une analyse de sa structure. Puis la fiabilité est évaluée par une estimation de l'exhaustivité des tests, définis suite à l'analyse du logiciel, qui lui ont été appliqués (taux de couverture des tests).

5.2.2. Par rapport aux tests

On trouve également des propositions d'évaluation de la fiabilité logicielle à partir des tests :

- Echantillonnage dans le domaine des entrées : Le domaine des données d'entrée est partitionné en fonction du profil d'utilisation opérationnelle du programme. Des points sont tirés au hasard dans les sous-ensembles et on exécute le programme un certain nombre de fois. Après N essais si D défauts ont été détectés, la fiabilité du programme est évaluée comme tel :

$$R = 1 - \frac{D}{N}$$

$$avec \begin{cases} R\ Fiabilité \\ D\ nombre\ de\ défauts \\ N\ nombre\ d'\ essais \end{cases}$$

- Echantillonnage dans le domaine des défauts : on introduit dans le programme à tester un ensemble de défauts connus et représentatifs par leur nature et leur fréquence des défauts inconnus et contenus dans le programme dans le but d'estimer le nombre de ces derniers en fonction du nombre de défauts introduits et de défauts détectés au cours du test.

5.2.3. Par utilisation de modèle de croissance de fiabilité

Une troisième méthode d'évaluation prévisionnelle de la fiabilité est l'utilisation de modèles de croissance de fiabilité parmi lesquels on trouve :

- Modèle de J.D Musa : on observe les intervalles de temps entre les défaillances (mesurés en temps CPU) T_i. L'hypothèse de ce modèle est que les T_i sont distribués suivant une loi exponentielle de densité :

$$f_i(t) = \lambda_i e^{-\lambda_i t} \quad t \geq 0$$

Dès lors le taux de défaillance est proportionnel au nombre de défauts résiduels

$$\lambda_i = \frac{c}{N_0 T_0}\left(N_0 - (i-1)\right)$$

$$avec \begin{cases} N_0\ Nombre\ d'erreur\ à\ t_0 \\ T_0\ MTTF\ à\ t_0 \\ c\ paramètre \\ i\ indice\ de\ mesure \end{cases}$$

La validité réplicative de ce modèle est d'environ 45%.

- Modèle de B. Littlewood : On relève également les intervalles de temps entre les défaillances (mesurés en temps CPU) T_i. L'hypothèse de ce modèle est que les T_i sont distribués suivant une loi de Pareto de densité :

$$f_i(t) = \frac{\alpha(\psi_i)^{\alpha}}{(t_i + \psi_i)^{\alpha+1}} \quad .$$

$$de\ paramètres\ \alpha\ et\ \psi$$

Le taux de défaillance suit la loi

$$\lambda_i = \frac{\alpha}{t + \psi_i} \times (loi\ \Gamma)$$

La validité réplicative et prédictive de ce modèle est la plus élevée et est d'environ 65%.

- Modèle de Goel-Okumoto : On relève le nombre de défaillance observées N_i sur un intervalle $[0, t_i]$ L'hypothèse de ce modèle est que les N_i sont distribués suivant une loi de Poisson non homogène de paramètre m(t) avec

$$m(t) = N(1 - e^{-bt})$$

$$avec\ N\ \ Nombre\ de\ défaillances$$

On peut ainsi évaluer l'espérance du nombre de défauts détectables quand $t \to \infty$, la probabilité de bon fonctionnement jusqu'à t et également l'espérance du nombre de défauts résiduels à t. La validité de ce modèle est de 45% mais il permet d'obtenir de bons résultats dans 20% des cas non traitables à l'aide du modèle de Littlewood.

5.2.4. Conclusion

Si ces modèles permettent d'obtenir un chiffre pour la fiabilité prévisionnelle des logiciels, leur utilisation n'est cependant pas aisée. En effet les hypothèses d'application (en terme de réparation, nombre de composants...) de chacun de ces modèles sont très contraignantes et si ces hypothèses ne sont pas remplies, les résultats obtenus peuvent être erronés.

Ainsi avant de choisir un modèle d'évaluation de la fiabilité il est nécessaire de se poser les problématiques suivantes :

- Premièrement les hypothèses nécessaires à la validité du modèle sont-elles vérifiées ?
- Deuxièmement le modèle est-il un bon modèle (simplicité, signification physique des paramètres, estimation de la précision, validité des résultats obtenus…) ?
- Enfin, le modèle doit être confronté aux données, afin de vérifier la compatibilité des données avec le modèle choisi a priori (test d'adéquation).

Pour conclure, s'il est possible d'obtenir un chiffre pour la fiabilité prévisionnelle des logiciels, ce calcul ne permet pas pour autant d'améliorer la confiance dans leur sûreté. En effet ces estimations ne permettent que de constater le niveau de sûreté atteint par un logiciel mais contrairement à une approche de la sûreté logicielle par processus, l'estimation de la fiabilité ne permet pas de prescrire les activités qui améliorent la sûreté d'un logiciel.

6. APPROCHE MAÎTRISE DU CYCLE DE VIE

L'origine des défaillances d'un logiciel est, d'après le retour d'expérience, répartie de la façon suivante :

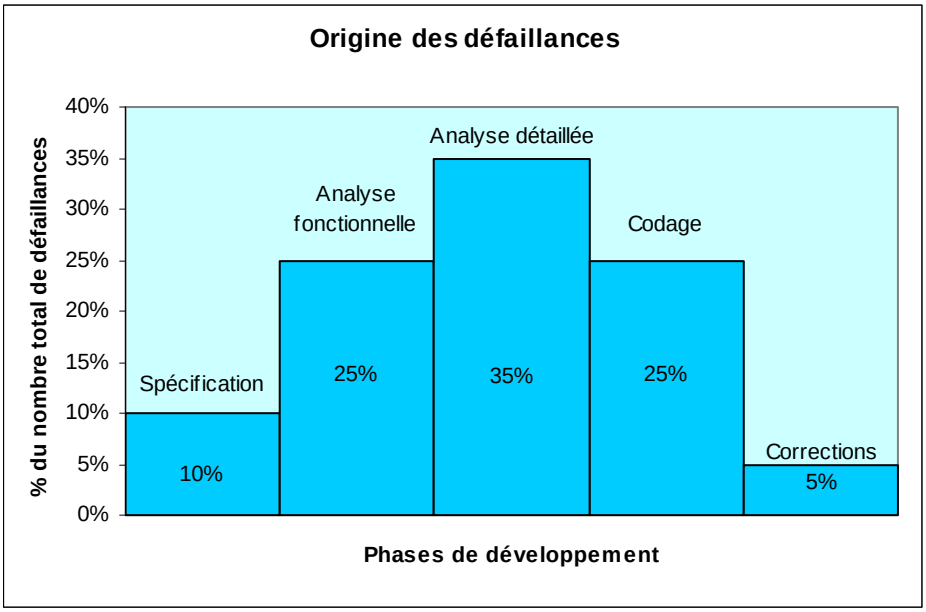


Figure 3: Répartition de l'origine des défaillances en fonction des différentes phases de développement d'un système logiciel

Le retour d'expérience permet également de constater que les premières défaillances à être détectées sont celles dues au codage ; viennent ensuite celles liées à l'analyse détaillée puis à l'analyse fonctionnelle, et enfin celles qui sont généralement détectées le plus tardivement sont les défaillances liées à des erreurs de spécification. Or, plus une défaillance de spécification ou d'analyse est détectée tardivement, plus sa correction coûte cher. De là provient l'impératif de détecter les erreurs au plus tôt.

Ainsi, afin d'obtenir une confiance justifiée dans la SdF d'un logiciel, il est possible de travailler à plusieurs niveaux :

- au niveau de la spécification du système ;
- au travers d'une analyse qualitative du système ;
- lors du développement en mettant en place des méthodes :
  - d'évitement des fautes afin d'empêcher l'occurrence de fautes (fautes) ;
  - d'élimination des fautes afin de détecter, identifier et enlever les fautes du système (fautes et défauts) ;
  - de tolérance aux fautes afin de préserver le service malgré l'occurrence de fautes (erreurs).

Nb : le terme faute est ici utilisé selon les expressions consacrées, de façon plus ou moins abusive par rapport aux termes définis au § 4.2.2.

Les méthodes exposées ci-après visent à maîtriser la qualité du logiciel en intervenant à ces différents niveaux.

6.1. LES NORMES RELATIVES À LA SÛRETÉ DE FONCTIONNEMENT LOGICIELLE

Il existe de nombreuses normes en matière de SdF logicielle. En effet chaque grand domaine (aviation, nucléaire, ferroviaire...) possède sa propre norme.

Ces normes permettent de décrire le processus à suivre au cours de tout ou partie du cycle de vie des logiciels afin de permettre d'avoir une confiance justifiée dans leur sûreté. Le suivi de ces normes peut permettre d'obtenir une certification des logiciels, à un niveau plus ou moins élevé (notion de niveau d'assurance : AL, SIL, SWAL…).

On peut citer les normes suivantes :

- Au niveau européen :
  - CEI 61508, Sécurité fonctionnelle des systèmes électriques, électroniques, électroniques programmables relatifs à la sécurité.
- Dans le domaine nucléaire :
  - CEI 61226 : Centrales nucléaires – Système d'instrumentation et de contrôle-commande importants pour la Sûreté –Classification Processus d'étude et de réalisation des logiciels équipements de contrôle commande de sécurité
  - CEI 60880 : Logiciel pour les calculateurs utilisés dans les systèmes de sûreté des centrales nucléaires
- Dans le domaine aéronautique :
  - DO 178B / ED12B : « Considérations sur le logiciel en vue de la certification des systèmes et équipements de bord» distribué en France par l'EUROCAE, traite des aspects logiciels en vue de la certification des systèmes et équipements de bord.
  - DO 278B / ED109 : « Guidelines for communication, navigation, surveillance, and air traffic management (CNS/ATM) systems software integrity assurance »
- Dans le domaine militaire :
  - Def Stan 00-56/Issue2 : « Safety management requirements for defence systems » (Ministry of Defence of UK)
- Dans le domaine ferroviaire :
  - CENELEC 50128 ("Logiciel pour les systèmes de contrôle et de protection ferroviaires") qui décrit des niveaux de garantie de sécurité et identifie les exigences en matière de personnel et de leurs responsabilités. La norme explique en détails les objectifs à atteindre, les documents à produire, les problèmes potentiels liés au cycle de vie du système et à sa documentation, son architecture, son design et son implémentation, de même que sur la vérification du système, son test, l'intégration logiciel/ matériel, la validation du logiciel, les assurances en matière de qualité et la maintenance.
  - CENELEC 50129 ("Systèmes de signalisation électroniques reliés à la sécurité")
  - CENELEC 50126 (Applications ferroviaires – Spécifications et démonstration de fiabilité, de la disponibilité, de la maintenabilité et de la sécurité FMDS

Nb : EN 50128 et EN 50129 sont des adaptations de la norme internationale CEI 61508 ("Sécurité fonctionnelle des systèmes de sécurité électriques, électroniques et/ou programmables") dans le domaine ferroviaire.

6.2. MÉTHODES FORMELLES

La spécification du système peut être réalisée à un degré plus ou moins élevé de formalisation.

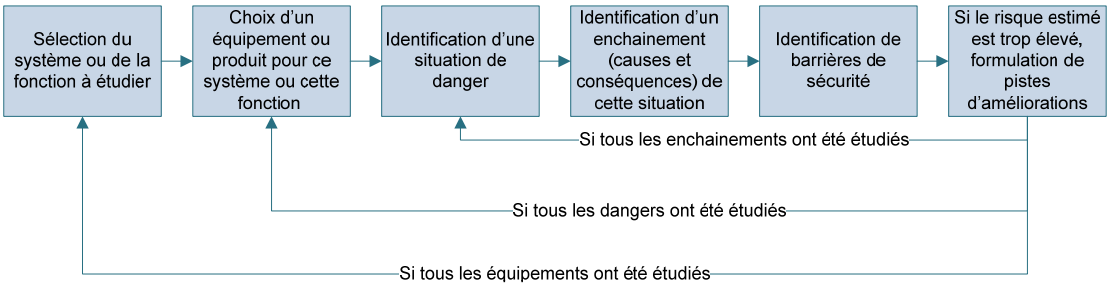
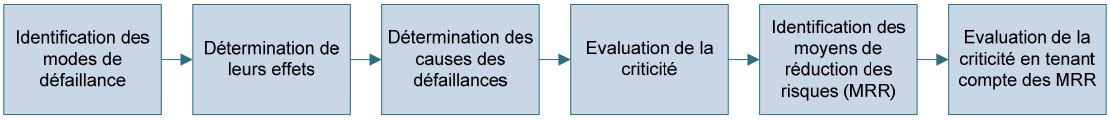
La façon la plus évidente est d'utiliser le langage naturel pour formuler ce qu'on attend du système. Mais il est également possible d'utiliser des méthodes formelles (Petri, Lustre, Z, B..) ou semi-formelles (SADT, SART). Ces méthodes font l'objet d'une autre étude et ne sont donc pas développées ici.

6.3. ANALYSE QUALITATIVE DU SYSTÈME

L'analyse du système peut participer à la sûreté des logiciels. Ces méthodes proviennent des différents domaines de la sûreté de fonctionnement (système SSI, logiciel…).

On peut distinguer les méthodes d'analyse en fonction du domaine où elles sont apparues.

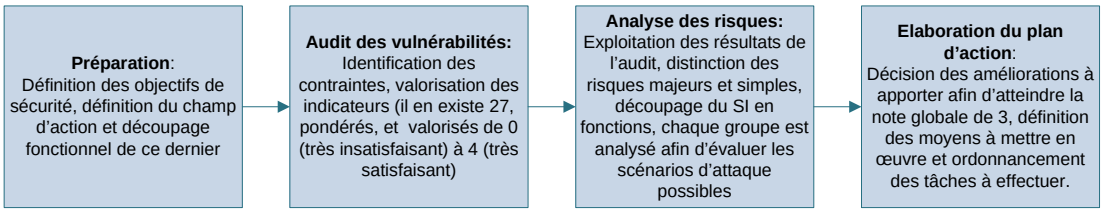
6.3.1. Méthodes et outils d'analyse issues de la SdF des systèmes

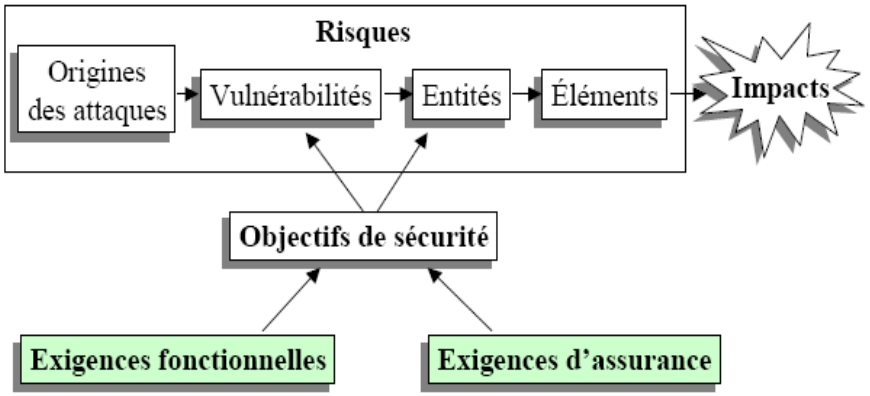
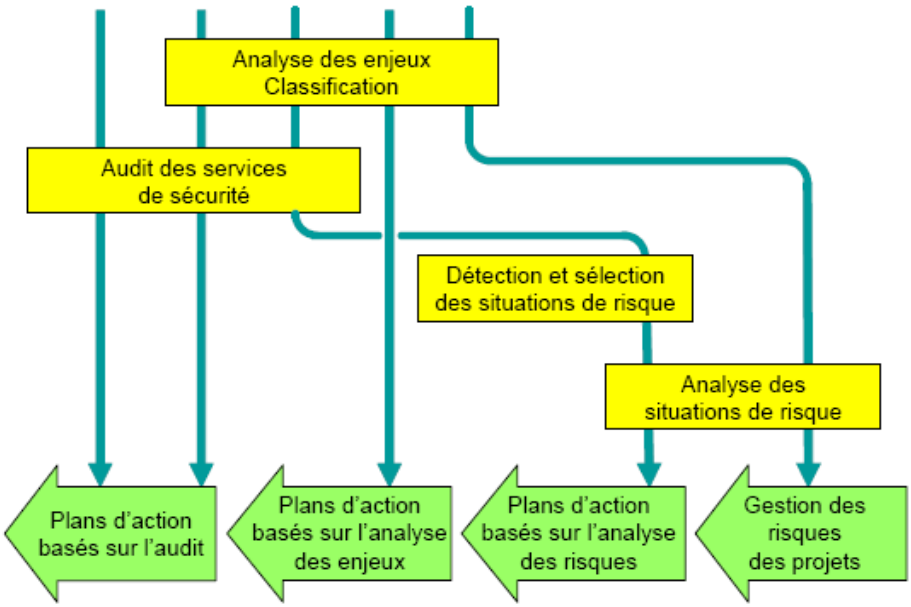
APR	Analyse préliminaire de risques. C'est une méthode d'identification et d'évaluation des risques liés à un système, à partir du retour d'expérience sur des systèmes équivalents dans le but d'établir une première liste de risques inhérents aux systèmes et d'y associer le plus tôt possible des propositions de réduction des risques. Cette méthode se déroule suivant le schéma suivant:  Cette méthode permet un examen rapide des situations dangereuses d'un système mais elle ne permet pas de décrire l'enchainement des évènements conduisant à la défaillance du système.
AMDEC (inductive)	Analyse des modes de défaillances, de leurs effets et de leur criticité. Elle permet, en partant d'une revue de l'ensemble des fonctions du logiciel, de déterminer les effets des défaillances des constituants et d'apprécier la gravité des conséquences ainsi que de rechercher des actions correctives. 
Diagramme de fiabilité	C'est un diagramme fonctionnel qui décrit toutes les fonctions nécessaires pour réaliser la fonction principale du système, en prenant en compte les moyens de secours. Cette méthode ne met pas en évidence les dysfonctionnements et ne précise pas les modes de défaillance du système.
Arbres de défaillance	Cette méthode d'analyse déductive vise à identifier les combinaisons d'évènements élémentaires à l'origine d'un évènement redouté.
Diagramme causes-	Cette autre méthode graphique est à la fois inductive et déductive, elle permet la modélisation des phénomènes transitoires propres au système étudié et la prise en compte des événements

conséquence	externes ou anormaux
Graphe d'état de Markov	Méthode graphique permettant de représenter les états d'un système et les transitions entre ces états.
Réseaux de Petri	Méthode graphique d'analyse des systèmes complexes, ils permettent en particuliers de mettre en relief les dépendances qui existent entre les différents composants (transitions).
Sneak analysis (analyse des conditions insidieuses)	L'objectif de cette méthode est de déterminer les conditions latentes pouvant causer un évènement redouté ou l'inhibition d'un évènement attendu, ceci sans lien avec une panne de composant. Cependant l'application de cette méthode dans le domaine logiciel ne s'est pas avérée pertinente

6.3.2. Méthodes d'origine SSI (sécurité des systèmes d'information)

La sécurité des systèmes d'information a elle aussi apporté ses méthodes d'analyse.

MARION	Elle permet d'évaluer la cohérence et le niveau de criticité, d'apprécier de façon qualitative et quantitative les risques encourus et de bâtir un plan d'orientation. Pour ce faire on procède à une analyse de l'existant (évaluation des risques et de leur gravité, audit) puis à une évaluation des contraintes avant de déterminer les mesures de protection et de prévention qui peuvent être mises en place. 
MELISA	Elle permet, par le biais d'une analyse du système, d'en déterminer les points faibles afin d'atteindre progressivement un niveau de vulnérabilité résiduelle tolérable. Cette vulnérabilité est évaluée à l'aide de simulations.
EBIOS	Méthode promue par la DCSSI, et utilisée par les administrations françaises. Elle se décompose en cinq étapes 1. Etude du contexte (entités -> éléments) ; 2. Expression des besoins de sécurité (entités / éléments -> impacts) ; 3. Etudes des menaces (Origines des attaques / vulnérabilités) ; 4. Expression des objectifs de sécurité ; 5. Détermination des exigences de sécurité ;

	
MEHARI Méthode Harmonisée d'Analyse des RIsques	Développée et proposée par le CLUSIF, cette méthode vise à différencier les modes de mise en œuvre des services de sécurité. Elle se compose de 5 grandes étapes : 1. Définition des objectifs, du périmètre et validation des métriques de sécurité. 2. Analyse des enjeux (focalisée sur les processus clés de l'entreprise) et classification des actifs (i.e. définition pour chaque type d'information et pour chaque ressource du système d'information et pour chacun des critères de classification (e.g. disponibilité, intégrité, confidentialité), des indicateurs représentatifs de la gravité d'une atteinte à ce critère pour cette information ou cette ressource) afin de ne retenir que les risques pesant sur les actifs essentiels. 3. Diagnostic des vulnérabilités, afin d'identifier la probabilité de concrétisation des menaces, de les évaluer et de les pondérer (risques acceptables ou non acceptables). 4. Définition de mesures de sécurité efficaces et robustes par rapport aux risques prédéterminés. 5. Planification et pilotage des travaux à mener. 

6.3.3. Méthode d'analyse particulière au logiciel, AEEL

Enfin le domaine des logiciels a mis au point une méthode d'analyse particulière aux logiciels, l'AEEL (Analyse des Effets des Erreurs de Logiciel, pendant de l'AMDEC système).

C'est une méthode systématique qui se déroule en trois temps :

1. Pour chaque module on fait une hypothèse d'erreur,
2. On étudie l'impact de chacune des hypothèses sur les sorties du module,
3. On analyse la propagation sur les modules environnant le module analysé.

Elle permet d'obtenir des recommandations de conception et de codage des modules (e.g. vérification ou filtrage des entrées, recommandations sur le séquencement…) afin de garantir des propriétés énoncées en phase de spécification.

Les analyses AEEL commencent lors de la phase de spécification du logiciel et s'achèvent après les tests et la phase d'intégration.

6.4. TECHNIQUES PARTICULIÈRES DE SÛRETÉ LOGICIELLE

6.4.1. La preuve de programme

Le principe est le suivant : on précise des propriétés que doivent vérifier les variables, sous forme de prédicats associés à chaque arc du programme. Une fois ces prédicats identifiés, il reste à prouver qu'ils sont vérifiés. Cette technique est relativement lourde (la charge dépend pour beaucoup de l'outil utilisé).

6.4.2. Evitement des fautes

L'objectif de l'évitement des fautes, également appelé prévention des fautes, est d'empêcher par construction l'occurrence ou l'introduction de fautes, de diminuer la probabilité de leur apparition.

On utilise pour cela des techniques de prévention telles que :

- mise en place d'un cycle de développement en V,
- analyse des besoins (afin d'éviter les erreurs de frontière avec l'environnement et celles d'incomplétude ou de non-conformité),
- utilisation de méthodes de spécification (de la spécification textuelle à la spécification formelle avec exploration de modèle, évaluation de propriétés ou preuve de propriétés, en passant par les spécifications semi formelles de type SADT, SART ou UML),
- mise en place de techniques particulières en phase de conception et de codage telles que la définition de règles de dérivation du modèle de spécification vers le code, de règles de codage et d'annotation de programmes, l'utilisation de modèles de conception, d'assertion et enfin des techniques d'inspection et d'analyse (statique ou non) de code,
- Utilisation de la technologie notamment par :
  - L'implantation des logiciels sur un noyau exécutif,
  - La maîtrise de la production de l'exécutable (compilateur validé ou certifié, gestion des configurations),
  - Le choix judicieux des composants matériels,
- Définition des procédures de maintenance en particulier en ce qui concerne la maintenance préventive.

6.4.3. Tolérance aux fautes

L'objectif de la tolérance aux fautes est de diminuer la probabilité de défaillance malgré la présence éventuelle de fautes ; on s'intéresse à la façon de délivrer, par redondance, un service conforme à la spécification. Une des principales failles de cette technique est l'existence de pannes de causes communes. Afin de limiter ces dernières il est possible d'utiliser des techniques de :

- Diversification ;
- Localisation des unités redondées (défaillance liée à la proximité des unités redondées) ;
- Désynchronisation des traitements (défaillance liée à une faute transitoire).

NB : Les défauts de « causes communes » doivent être traités en priorité. En effet si les composants sont tous indépendamment tolérants aux fautes mais qu'ils peuvent défaillir du fait d'une panne d'ordre général, les efforts portés sur chaque composant ne seront pas suffisants pour garantir la sûreté du logiciel.

La tolérance aux fautes de chaque module peut être réalisée à l'aide de techniques de détection d'erreur ou encore de confinement ou de recouvrement.

6.4.3.1. Détection d'erreur

La détection d'erreur peut se faire à l'aide de :

- Tests cycliques (ou périodiques) : l'exécution d'un module est suspendue et on déroule une routine de test pour détecter la présence éventuelle d'erreurs dans le module. Ce mécanisme ne peut détecter les pannes temporaires que si elles apparaissent pendant la période de test, ce qui implique des problèmes de couverture du test et d'optimisation de la fréquence de test.
- Chiens de garde (watchdog) : des timers sont réglés pour effectuer des contrôles à des points pré-établis dans le programme qu'exécute un module. Pour un point de contrôle particulier, le chien de garde est armé et décroit pendant que le module exécute ses fonctions. En l'absence d'erreur, le chien de garde est réarmé avant que le prochain point de contrôle soit atteint. L'occurrence d'une faute (hard ou soft) empêche le module de réarmer le chien de garde.
- Circuits autotestables, qui sont conçus pour produire une sortie correcte ou indiquer la présence d'une faute dans le module :
  - Par codage : Lors de la création d'une donnée c, une donnée r=h(c) est calculée. On peut vérifier ultérieurement que cette relation est vérifiée ;
  - Par vote : Plusieurs versions d'une même donnée sont calculées « indépendamment ». Une erreur est détectée lorsqu'une version au moins est distincte des autres.

    Si l'on dispose de suffisamment d'informations redondantes, et sous une hypothèse d'étendue limitée de l'erreur, les deux techniques précédentes permettent de masquer certaines erreurs.
  - Par contrôle dynamique :
    - Contrôles temporels (du séquencement des instructions ou des séquences d'évènements) ;
    - Contrôles de vraisemblance ;
    - Contrôle d'intégrité portant sur la structure des données ;
    - Contrôle de signature (détection des erreurs sur le séquencement et sur la sélection des opérandes d'une opération) ;
    - Autotests afin de révéler les fautes dormantes.

6.4.3.2. Confinement

L'objectif du confinement est d'éviter la propagation d'un état erroné, que cette propagation soit

- Spatiale (barrières portant sur les interfaces. La structuration des données doit permettre de détecter les erreurs lors des transmissions de données entre modules) ;
- Temporelle (l'utilisation de tests en ligne permettant d'éviter les erreurs opérationnelles en activant une erreur latente avant son activation par l'application).

6.4.3.3. Recouvrement

Les blocs de recouvrement constituent des hypothèses de validité des fonctions spécifiées.

Le recouvrement peut avoir lieu :

- Par reprise arrière, nécessitant une sauvegarde périodique de l'état du système et la détermination de l'état global cohérent le plus proche avant reprise des traitements à partir de ce dernier.
- Par reprise avant, en reconstruisant le contexte à partir de données externes (capteurs…) et à l'aide d'une programmation qui prévoit les traitements de cas erronés ou dégradés (exceptions, signaux d'erreur).
- Par masquage, en utilisant la redondance de manière à ce que le système continue à fonctionner correctement même en présence d'erreur.

6.4.4. Elimination des fautes

L'objectif de l'élimination des fautes est de détecter, identifier les fautes du système, par exemple à l'aide de tests ou de relecture critique de code, et d'enlever ces fautes.

6.4.4.1. Les Tests du logiciel

On distingue quatre grandes familles de tests :

- Les tests statiques : lecture des spécifications et du code, avant tout passage en machine. Le but est de détecter les défauts de structure, de logique et de syntaxe et de vérifier que les normes qui s'appliquent sont respectées.
- Les tests symboliques : c'est une exécution symbolique des spécifications (si elles sont formelles) ou du code en calculant sur les noms symboliques des variables. Le but est d'analyser le comportement du programme le long d'un chemin.
- Les tests dynamiques : c'est le test (classique) qui correspond à faire passer des jeux d'essai et à en vérifier la bonne exécution et le résultat. Ils peuvent être fondés :
  - Sur la structure du logiciel (boîte blanche, tests structurels), le programme est vu comme un graphe, les nœuds décrivant les points de décision et les arcs des séquences d'instruction. On cherche alors à couvrir toutes les instructions et toutes les décisions ;
  - Sur la définition de la fonction à réaliser (boîte noire, tests fonctionnels), vérification de la conformité du logiciel à sa spécification ;
  - Sur les défauts à détecter.
- Les tests d'intégration permettent de vérifier la cohérence des interfaces entre modules mais également les propriétés de comportement au niveau des appels.

6.4.4.2. Relecture critique de code

Elle a pour objectif de vérifier que le code est correct, et nécessite pour cela l'établissement de règles de codage en amont. La RCC permet de s'assurer que ces règles ont bien été respectées.

6.5. CONCLUSION

Lorsqu'on veut donner confiance dans le fait qu'un logiciel est suffisamment sûr, il est possible d'utiliser les nombreuses méthodes présentées ici, qu'elles soient la déclinaison au niveau logiciel de méthodes existantes dans d'autres domaines (système, SSI, facteurs humains) ou qu'elles aient été définies spécifiquement pour la sûreté logicielle.

Si elles ne permettent pas de conclure sur une fiabilité chiffrée du logiciel, elles permettent d'apporter des preuves que le logiciel est maîtrisé.

La variété des méthodes permet de déterminer pour chaque logiciel, en fonction de sa nature (logiciel propriétaire, libre, COTS…) et de sa criticité, une démarche de démonstration propre.

7. CONCLUSION

Comme nous avons pu le voir en introduction, de part leur nature déterministe, les logiciels ne peuvent se voir attribuer un taux de fiabilité prévisionnel de façon analogue au matériel.

Pour pouvoir évaluer et maîtriser la SdF des logiciels, il existe deux écoles reposant sur des philosophies différentes. L'approche fiabiliste vise à obtenir un chiffre pour la fiabilité des logiciels ; L'approche « maîtrise du cycle de vie » vise à obtenir des assurances sur le fait qu'il est suffisamment sûr. Aujourd'hui, l'état de l'art tend à nettement préférer la deuxième approche en raison des difficultés et limitations de la première.

Par conséquent, de nombreuses techniques, méthodes et outils de sûreté de fonctionnement ont été définis de façon spécifique aux logiciels afin de pouvoir justifier d'une confiance suffisante dans le fait qu'ils sont sûrs. Ces méthodes s'appliquent tout au long du cycle de vie d'un logiciel et permettent de réduire au maximum les erreurs humaines en conception qui sont les seules sources des défaillances du logiciel en exploitation.

Ainsi, hormis certaines méthodes transverses (AEEL...), pour chaque grande phase du développement d'un logiciel il existe des méthodes destinées à diminuer au maximum l'introduction de fautes. Le schéma suivant illustre d'une part la répartition des défaillances suivant les grandes étapes du développement d'un logiciel, et d'autre part les méthodes qui peuvent être utilisées afin de réduire les risques d'introduction de fautes.

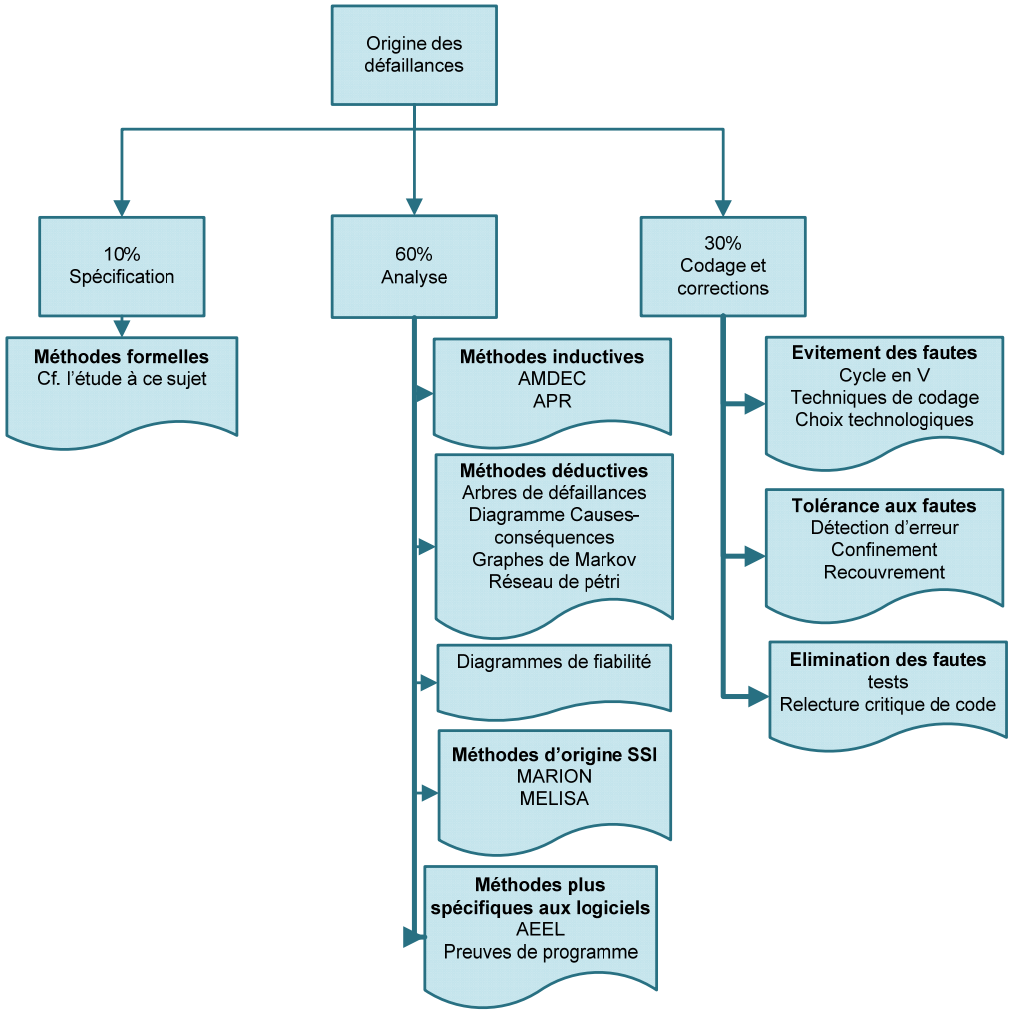


Figure 4: Cartographie des méthodes applicables lors du développement d'un logiciel

Parmi toutes ces méthodes, la plus usitée est la mise en place de tests. La figure ci-dessous recense les différents types de tests possibles en fonction de l'approche désirée.

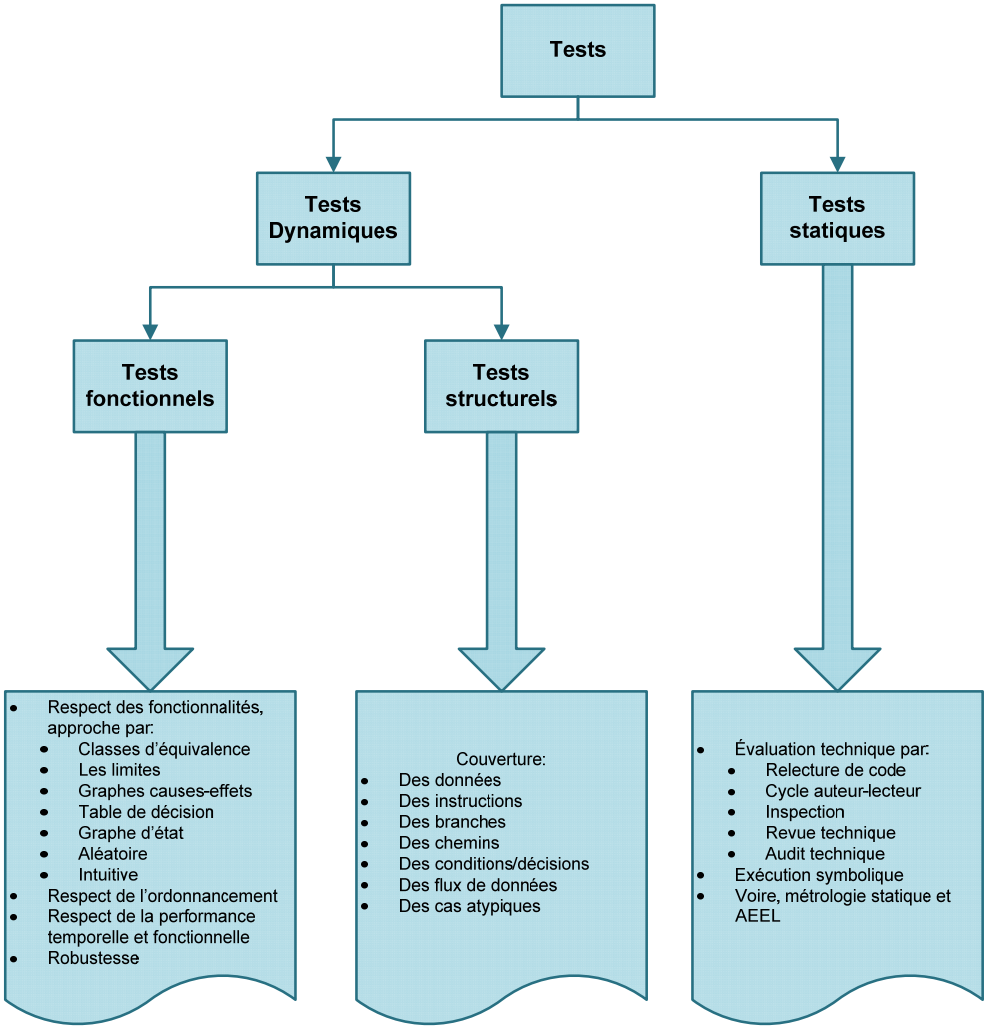


Figure 5: Cartographie des types de tests et des techniques complémentaires d'aide relative à chacun d'eux

Comme nous venons de le voir il existe de nombreuses méthodes pour réduire au maximum l'occurrence des défaillances logicielles. Si, en ce qui concerne les méthodes d'analyse et de spécification de système, peu d'innovations ont eu lieu ces dernières années, l'accent est actuellement mis sur les méthodes de vérification et validation formelles de code. C'est en effet dans ce domaine que le plus grand nombre de travaux ont été publiés ces dernières années dans le cadre de la sûreté logicielle.

ANNEXE 1 - BIBLIOGRAPHIE

[EUROSAE-08] Support de la formation Eurosae « Sûreté de fonctionnement des systèmes logiciels » , Stage ICA003.

[QUASSI1] Garin C., Tessier J., Beguigneau M. Présentation de Projet Génie Logiciel, DESS QUASSI de l'université d'Angers, La métrologie orientée objet appliquée à la sûreté de fonctionnement.

[QUASSI2] Mahamoud N., Attelly S, Présentation de Projet Génie Logiciel, DESS QUASSI de l'université d'Angers, La Tolérance aux fautes dans les systèmes informatiques, Février 2002.

[QUASSI3] Gil C., Le Guyader G., Planchot L. Présentation, DESS QUASSI de l'université d'Angers, La Tolérance aux fautes : programmation défensive, décembre 2003.

[CNAM] Florin G, Laboratoire CEDRIC, Présentation La tolérance aux pannes dans les systèmes répartis.

[LAPRIE1] J-C. Laprie, Guide de la Sûreté de fonctionnement, Cépaduès, Toulouse, Mai 1995, 369.

[LAPRIE2] J-C. Laprie, Présentation Sûreté de fonctionnement informatique : concept, défis, directions, 15-17 novembre 2004.

[Villemeur] Villemeur A., Sûreté de fonctionnement des systèmes industriels, Eyrolles, Mars 1997.

[CE 482/2008] Commission Européenne, Règlement (CE) N° 482/2008 établissant un système d'assurance de la sécurité des logiciels à mettre en œuvre par les prestataires de services de navigation aérienne et modifiant l'annexe II du règlement (CE) no 2096/2005, 30 Mai 2008.

[ED-12B] Considérations sur le logiciel en vue de la certification des systèmes et équipements de bord.

Présent pour l'avenir

Service technique de l'aviation civile
31, avenue du Maréchal Leclerc
94381 BONNEUIL-SUR-MARNE CEDEX
Tél. 33 (0)1 49 56 80 00
Fax 33 (0)1 49 56 82 19

Site de Toulouse
9, avenue du Docteur Maurice Grynfogel - BP 53735
31037 TOULOUSE CEDEX 1
Tél. 33 (0)1 49 56 83 00
Fax 33 (0)1 49 56 83 02

Centre de test de Biscarrosse
Centre d'essais de lancement de missiles - BP 38
40602 BISCARROSSE CEDEX
Tél. 33 (0)5 58 83 01 73
Fax 33 (0)5 58 78 02 02