Valeur de la diversité de compilateurs pour les systèmes critiques

par

Gilles Brunet

Essai présenté au CeFTI

En vue de l’obtention du grade de maître en Génie logiciel

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Longueuil, Québec, Canada, 5 juillet 2018

Sommaire

L'avènement de l'Internet des objets (IdO) projette notre société à l'aube d'une quatrième révolution industrielle. Des centaines de milliards d'objets interconnectés vont bientôt déferler aussi bien dans notre vie domestique que dans nos usines. Leur implantation rapide, économique et sans aucune borne impliquent une dangerosité. La question ne consiste pas à savoir si une catastrophe peut se produire, mais à déterminer quand. Les acteurs qui concourent dans ce mouvement doivent redoubler d'efforts pour maîtriser cette menace, mais sans compromettre leur retour sur investissement (ROI). Les compilateurs utilisés pour la production du logiciel comportent leurs limites, mais les concepteurs de logiciels présupposent leur invulnérabilité. Dès lors, employer un seul compilateur augmente le risque d'injecter des anomalies inattendues. À quel point la combinaison d'une variété de compilateurs peut-elle atténuer de façon mesurable, et pour un moindre coût, les défectuosités qui s'incrustent au logiciel embarqué sur ces objets ?

Les désastres industriels poussent nos sociétés à régulariser les procédés et maitriser les risques dangereux. En 1998, la CEI publie une première édition de la norme 61508, outillée d'une politique de sécurité fonctionnelle et documentée pour encadrer la conception de systèmes électriques, électroniques ou électroniques programmables (É/É/ÉP). Le CEI quantifie les probabilités de pannes du matériel pour échelonner leur sûreté sur quatre niveaux d'intégrité et de sécurité (*Safety Integrity Level* ou SIL). En ce qui a trait au logiciel, le niveau de sécurité se caractérise par le concept de capabilité systématique (*Systematic Capability* ou SC). Cette mesure se répartit d'une manière analogue au SIL, mais découle d'une analyse subjective du cycle de vie et des processus de développement, plutôt que d'un calcul. Objectiver une partie de l'indice SC devient nécessaire pour mesurer l'effet d'une variété de compilateurs sur la capabilité systématique du logiciel.

La norme CEI 61508 comporte une mesure pour évaluer le taux approximatif de défaillances de cause commune (*Common Cause Failure* ou CCF), aussi appelé « Effet β ». Un CCF se caractérise par les pannes coordonnées de sous-systèmes É/É/ÉP redondants en conséquence d'un même événement. La technique de défense consiste à diversifier les

technologies de chaque composant pour diminuer la probabilité de panne simultanée de l'ensemble. La diversité s'applique généralement au matériel, mais pour le logiciel, l'hypothèse conduit à utiliser une diversité de compilateurs. Combiner des compilateurs différents implique que les programmes produits n'héritent pas d'une vulnérabilité commune et peuvent former un système plus robuste. L'analyse d'un *Soft PLC* homologué et destiné aux déploiements en double redondance permet d'observer les variations de probabilités de pannes en relation avec l'effet β. Une lecture attentive des tables de calculs offerts par les auteurs de la norme CEI 61508 permet de constater que le taux CCF doit réduire au-delà de 50 %, pour se répercuter sur l'indice SC.

Un sondage destiné aux professionnels et experts homologués en sécurité fonctionnelle pose dix questions pour valider la crédibilité et la portée d'une diversité de compilateurs sur l'indice CCF. La communauté réserve un accueil tiède à diversité de compilateur comme technique défensive contre les CCF. Au-delà de 85 % des professionnels et experts interrogés se rabattent sur les exigences de la norme concernant les logiciels et rejettent les mesures issues de calculs pour considérer l'effet β de la diversité de compilateurs. Seuls 8 % des participants au sondage accordent à la technique une diminution supérieure à 50 % du CCF. Ces résultats rendent négative l'hypothèse qu'une variété de compilateur puisse agir sur l'indice SC, mais révèlent un besoin d'objectiver les formules pour déterminer la capabilité systématique du logiciel.

La communauté admet toutefois la pertinence de mesurer la couverture de tests d'un logiciel. Un automatisme embarqué se vérifie par des tests par classes d'équivalences. En ce qui concerne les composants de type *Soft PLC*, la combinaison de deux recherches se veut prometteuse, et permet d'assurer une mesure encore plus complète de l'indice SC. Le module d'exécution d'un *Soft PLC* représente une complexité comparable à celle d'un compilateur. La technique que X. Yang et *al* appellent « test flou » [1] devient une formule idéale pour calculer le taux de pannes du composant. Le CCF perd ainsi de son utilité. Cependant, employer les probabilités de pannes logicielles calculées permet ensuite d'utiliser la technique par combinaison SIL que proposent Y. Langeron et *al* [2]. L'association de ces techniques vient alors objectiver l'indice SC.

Remerciements

Je tiens à remercier tout spécialement le professeur Patrice Roy, mon directeur académique, pour la générosité, pertinence et la précision de ses recommandations.

Je tiens également à remercier mes professeurs Lynn Legault et Vincent Echelard qui ont su m'encourager aux pires moments et guider la rédaction l'essai. Yasmine Lee, qui s'est dévouée corps et âme au moment de réserver le local de conférence, pour les répétitions ainsi que le grand soir. Merci à Claude Cardinal, pour ses précieux conseils en orientation académique et les choix de cours.

Gros merci, à Christina Deguire, collègue de travail et rédactrice technique, pour la relecture du sondage. Christina révisait mon sondage rédigé en langue anglaise.

Je m'en voudrais d'oublier mes deux fils, Nicolas-Gilles et Louis-Philippe, qui devaient sacrifier leur temps de qualité en ma présence et auquel ils avaient pourtant bien droit. Sans oublier leur mère, Christine, qui a toujours su faire preuve de compréhension et assumer sans broncher plus de temps de garde. Ce cheminement se termine.

Table des matières

Liste des tableaux

Liste des figures

Glossaire

1oo1	Architecture de système simple, qui comporte une sortie par canal (*One out of one,* ou *simplex*).
1oo2	Architecture de système à double redondance, qui comporte une sortie provenant de deux canaux (*One out of two,* ou *duplex*).
CCF	Défaillance pour cause commune (en anglais: *Common Cause Failure*).
CCF	Défaillance de cause commune (en anglais: *Common Cause Failure*).
CEI 61508	Norme de la CEI appliquée à l'industrie qui traite de sécurité fonctionnelle des systèmes É/É/ÉP.
CFSE	Expert homologué en sécurité fonctionnelle (*Certified Functional Safety Expert*).
CFSP	Professional homologué en sécurité fonctionnelle (*Certified Functional Safety Professional*).
DC	Couverture du diagnostic (en anglais: *Diagnostic Coverage*).
É/É/ÉP	Électrique, électronique, électronique programmable.
MTTF	Temps moyen de bon fonctionnement (*Mean time to fail*).
MTTR	Temps moyen de réparation (*Mean time to repair*).
PFD	Probabilité de défaillance dangereuse en mode sollicitation sur demande (*Probability of dangerous failure on Demand*).
PFH	Probabilité de défaillance dangereuse en mode de sollicitation élevée (*Probability of dangerous failure in high demand*).
SC	Capabilité systématique (en anglais: *Systematic Capability*).
Sécurité fonctionnelle	Pratique de la maîtrise et du contrôle des risques dangereux associés aux opérations des systèmes industriels.
SIL	Niveau d'intégrité de sécurité (*Safety Integrity Level*).
β	Facteur ou indice CCF.

Liste des sigles, des symboles et des acronymes

ADA	Langage de programmation orienté objet fortement typé et favorisant la programmation par contrat.
C	Langage de programmation structuré le plus répandu pour la programmation de systèmes embarqués.
C++	Langage de programmation multi-paradigme apparenté au langage C.
CEI	Commission électrotechnique internationale.
CEI 61131-3	Langage de programmation dédié aux automates programmables.
CompCert	Compilateur C qui comporte une autovérification formelle.
Coq	Langage de spécification formelle.
Csmith	Outil capable de produire des programmes en langage C valides au hasard, pour tester les compilateurs C.
Design by contract	Programmation par contrat. Paradigme par lequel l'exécution d'un programme reste encadrée de règles qui précisent les responsabilités entre le consommateur et le fournisseur d'une pièce de code.
DSL	Langage de programmation dédié (*Domain Specific Language*).
GCC	*GNU Compiler Collection*, un ensemble de compilateurs ouverts et régi par la fondation GNU. Il propose de compiler entre autres les langages C et C++.
GNU	Projet de système d'exploitation libre et ouvert.
IdO	Internet des objets.
LLVM	Infrastructure de compilateur ouverte (*Low Level Virtual Machine*).
MISRA C	Norme de programmation en langage C qui comporte des règles de sécurité.
MISRA C++	Norme de programmation en langage C++ qui comporte des règles de sécurité.
Monitoring	Mesure d'un système électronique ou électrique pour en assurer la supervision.
OS	Système d'exploitation (*Operating System*).
PLC	Automate programmable.
ROI	Retour sur investissements (*Return on Investment*).
Soft PLC	Automate programmable sous la forme d'un logiciel.
Spark	Implémentation d'un compilateur ADA qui comporte une autovérification formelle.

SSA	Phase d’optimisation d’un compilateur (*Single Static Assignment*).
Test flou	Technique de tests en boite noire qui s’appuie sur la génération automatique de classes d’équivalences.
UCT	Processeur ou unité centrale de traitement (*Central Processing Unit* ou CPU).

Introduction

Le présent essai pose la question de l'impact d'une diversité de compilateurs sur le degré de sécurité fonctionnelle d'un système électronique programmable [3] équipé d'unités de programmation logiques redondantes [4]. La valeur d'une telle diversité découle de sa simplicité et du faible coût d'implémentation [5].

L'industrie emploie les unités de programmation logiques (*Programmable Logic Controller* ou PLC) pour opérer la production de biens, ou contrôler les instruments de mesure [3], etc. Le standard CEI 61508 [6]–[12] propose la redondance de PLC pour prévenir les dysfonctionnements au sein de systèmes électroniques programmables impliqués dans une fonction de sécurité [4], [13]. Parallèlement, les architectures matérielles se multiplient et des fabricants d'automatismes industriels s'emploient depuis deux décennies à développer des PLC sous forme de logiciel embarqué : le *Soft PLC* [14]–[16]. Ces composants logiciels agissent comme une forme d'abstraction du matériel et se distribue en trois niveaux :

1. Un système de base (*Operating System* ou OS) – Agit comme intermédiaire entre le matériel et le logiciel. Il gère les accès aux ressources et l'exécution des tâches, comme celle de la machine virtuelle. Un outil permet d'écrire le système de base dans la mémoire de l'unité de traitement.
2. La machine virtuelle embarquée — Un premier compilateur traduit le code source de la machine virtuelle en logiciel optimisé pour cette unité [15]. Un outil permet d'écrire ce logiciel dans la mémoire de l'unité de traitement.
3. Le code d'instruction binaire, optimisé pour la machine virtuelle embarquée — La plupart des *Soft PLC* s'accompagnent d'une gamme d'outils [17] pour assurer la traduction des spécifications de contrôle en instructions optimisées pour la machine virtuelle [15]. La machine virtuelle exécute [18] ces instructions.

Plus précisément, la présente étude porte sur l'apport d'une diversité de compilateurs ANSI-C89, qui n'offrent individuellement aucun moyen de vérification, mais qui augmentent néanmoins, pour un faible coût, l'intégrité d'un système critique équipé de *Soft PLC* redondants

[16]. Par extension, l'essai soulève le gain additionnel d'intégrité, lorsqu'un compilateur C homologué pour les systèmes critiques produit un des *Soft PLC* redondants.

L'étude pose l'hypothèse que la qualité et la diversité des compilateurs permet aux fabricants de *Soft PLC* de répondre au paradoxe de l'augmentation de sécurité par des moyens simples et abordables. Beaucoup de fabricants de systèmes électroniques programmables proposent des infrastructures de migration pour *Soft PLC* [19]. Plusieurs d'entre eux tendent vers le marché domestique par l'intermédiaire du concept « Internet des objets » (IdO, *Internet of Things* ou IoT) [20]. Comme Gartner prévoit compter 25 milliards d'objets interconnectés d'ici 2020, dont un quart de milliards installés dans les véhicules seulement [21], l'opportunité d'étendre l'offre des unités logiques programmables au marché domestique se présente, mais impose une sévère réduction des coûts de conception. Une aussi forte croissance introduit de nouveaux dangers [22] et rehausse l'importance des homologations de sécurité fonctionnelle, mais réduit d'autant les marges bénéficiaires et les délais de commercialisation.

La sécurité fonctionnelle prend la forme d'une fonction de surveillance, qui s'ajoute aux processus opérationnels, pour contrôler le risque de défaillances dangereuses susceptibles de compromettre la vie, l'environnement, les acquis économiques et le matériel [23]. Le standard CEI 61508 [6]–[12] propose un modèle de prévention des dangers par la formalisation des indices de capabilité systématique (*Systematic Capability* ou SC) qui officialisent le degré de conformité associé à chaque composant intégré dans ce type de fonction [9, Chap. 3.5.9]. Les indices SC de chaque composant se combinent pour déterminer à l'échelle du système, une gradation de niveaux d'intégrité de sécurité (*Safety Integrity Level* ou SIL) [9, Chap. 3.5.8]. La norme considère la machine virtuelle [15] de type *Soft PLC* comme un composant à part entière dont le niveau de sécurité fonctionnelle s'exprime en indice SC [9, Chap. 3.5.9].

La présente étude tente de déterminer l'impact de la qualité et de la diversité des compilateurs sur l'indice SC d'un *Soft PLC*. Le document se distribue en cinq chapitres :

1. Mise en contexte – Ce chapitre présente le défi de certifier la sécurité fonctionnelle d'un système électronique programmable. Il décrit la norme CEI 61508, qui formalise les démarches de vérification et validation. Il survole la problématique qui accompagne toute migration d'une machine virtuelle sur un système électrique,

électronique, électronique programmable (É/É/ÉP) impliqué dans une fonction de sécurité.

2. Revue de la littérature – Ce chapitre présente un survol des articles sur la sécurité fonctionnelle, la vulnérabilité des compilateurs et l'apport de la diversité en réponse à ces limites.
3. Problématique – Ce chapitre détermine en quoi la diversité et la qualité d'un compilateur permettent d'attester le niveau de sécurité fonctionnelle d'un système électronique programmable.
4. Approche proposée – Ce chapitre documente la méthodologie de diversification et de quantification de l'indice SC.
5. Analyse des résultats – Ce chapitre présente l'avis de la communauté sur l'impact de la diversification et la qualité de compilateurs sur la quantification de l'indice SC.

Chapitre 1

Mise en contexte

Les désastres industriels survenus depuis la première révolution industrielle incitent les sociétés à produire et parfaire des standards de sécurité [5]. L'homologation d'un degré SIL s'applique à un système É/É/ÉP impliqué dans une fonction de sécurité [23], en vue de prédire les défaillances dangereuses pour la vie, l'environnement, les acquis économiques et le matériel. L'indice SIL indique la fréquence et le temps maximal d'indisponibilité en conséquence d'une défaillance [23]. La gradation SIL s'échelonne d'un à quatre. Comme par exemple, un système sollicité plus qu'une fois l'an [9, Chap. 3.5.16] au degré SIL1 reste inaccessible une heure d'opération sur 100 000, alors que l'échelon SIL4 ne l'est seulement qu'une sur 100 millions [24]. Cependant, devancer le danger exige une politique technique susceptible d'encadrer l'analyse, les indicateurs et les mesures relatives à la sécurité [6].

Au début des années 1980, les modèles de cycle de vie liés à la sécurité fonctionnelle sont apparus dans une volonté de maîtriser les dangers que représente l'automatisation [25]. Les consommateurs et la société exercent une pression toujours accrue pour que les organisations susceptibles de représenter un danger se conforment à des règles de sécurité vérifiables [26]. En 1998, la CEI publie la première édition du standard CEI 61508, proposé comme une approche générique outillée d'une politique de sécurité fonctionnelle documentée, rationnelle et cohérente [23]. Depuis sa première édition, d'autres normes appliquent un dérivé de CEI 61508 adapté à des réalités sectorielles, telles que :

- *CEI 62061* — machinerie [23] ;
- *CEI 61511* — instrumentation des processus [23] ;
- *CEI 61513* — automatisation des centrales nucléaires [23], [27] ;
- *ISO 26262* — automobile [27] ;
- *DO-178B* — aviation [27] ;
- *EN 50 129* — ferroviaire [27] ;
- *CEI 61601* — secteur médical [27].

CEI 61508 se positionne comme une norme de base, que d'autres comités doivent adapter à des réalités sectorielles [23]. La liste qui précède énumère quelques normes dérivées.

1.1 Système électronique programmable

Un système électronique programmable forme un assemblage de composants, comprenant de la mémoire, souvent de masse, des ports de communications, des interfaces d'entrées-sorties (E./S.) et des unités de traitements configurables ou exécutant un logiciel embarqué [3]. Qu'ils soient impliqués ou non dans une fonction de sécurité, les systèmes électroniques programmables se définissent par un ensemble de caractéristiques communes, tel que présenté dans la Figure 1 [18].

L'illustration qui suit en expose les sous-composants et offre une vue externe typique :

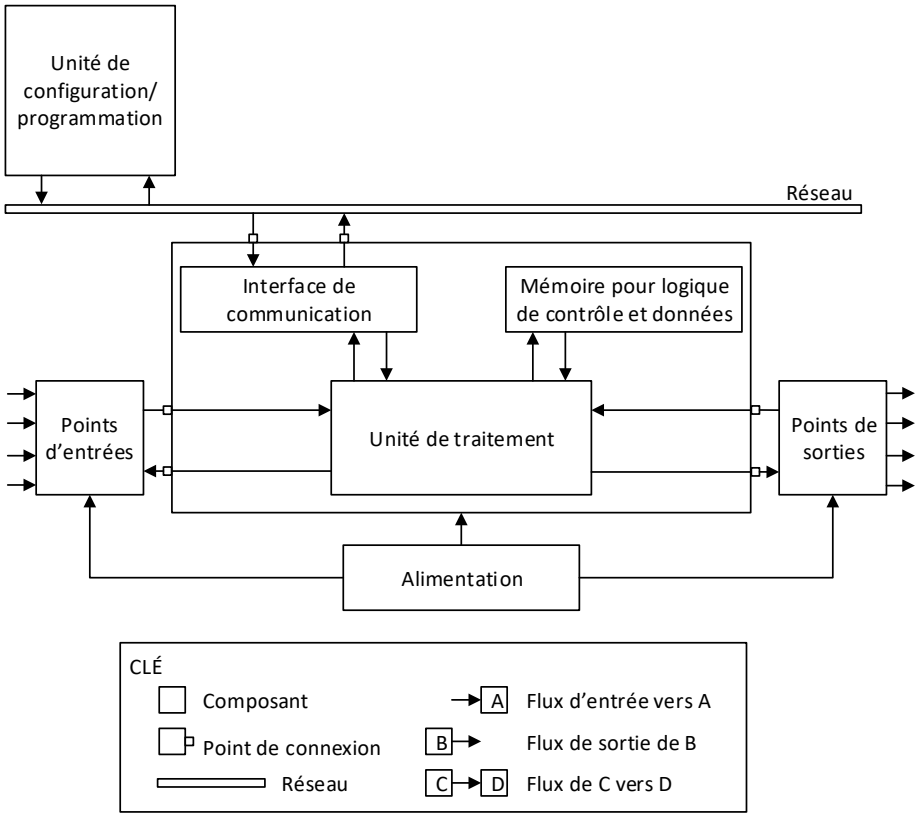


Figure 1 Le système électronique programmable (inspiré de [18, Fig. 1.4]**)**

Tableau 1 Composants de la Figure 1

Élément	Description
Unité de configuration/ programmation	Une chaîne d'outils pour la programmation la configuration et le monitoring, pouvant échanger la logique de contrôle ou d'autres données avec la mémoire du système électronique programmable.
Réseau	Bus de communication externe.
Points d'entrée	Fait l'interface entre le système électronique programmable et des senseurs ou d'autres instruments d'acquisition de données prises de son environnement.
Points de sortie	Fait l'interface entre le système électronique programmable et des actionneurs ou d'autres instruments permettant d'avoir un effet sur son environnement.
Alimentation	Convertit le voltage électrique pour alimenter le système électronique programmable.
Unité de traitement	Exécute les instructions de programmes stockés en mémoire, répond aux requêtes de communication en obéissant à un protocole, traduit les signaux transmis par les points d'entrée, ou envoie des signaux vers les points de sortie.
Interface de communication	Fait l'interface entre le système électronique programmable et le réseau, et gère les flux de communication.
Mémoire pour logique de contrôle et données	Mémoire au sens large, rassemblant la mémoire RAM, ROM, et la mémoire de masse (ex. : disque dur). Renferme les programmes exécutés par l'unité de traitement, les données reçues des points d'entrées, celles envoyées aux points de sorties ou échangées par l'interface de communication.

L'unité de traitement occupe la position centrale, puisqu'elle exécute l'ensemble du contrôle [18]. La logique se présente sous la forme d'un logiciel embarqué codifié spécifiquement pour ce composant.

Les fabricants de systèmes électroniques programmables installent souvent une machine virtuelle, elle-même accomplie par l'unité de traitement, mais qui exécute des codes de commandes structurés [19]. Dès qu'ils sont impliqués dans une fonction de sécurité, autant la machine virtuelle que le code de commandes doivent faire l'objet de vérification formelle [8].

Les systèmes électroniques programmables sont très souvent combinés au sein d'un réseau. L'illustration suivante démontre une composition simple qui implique un convoyeur qui déplace des aliments au travers un four :

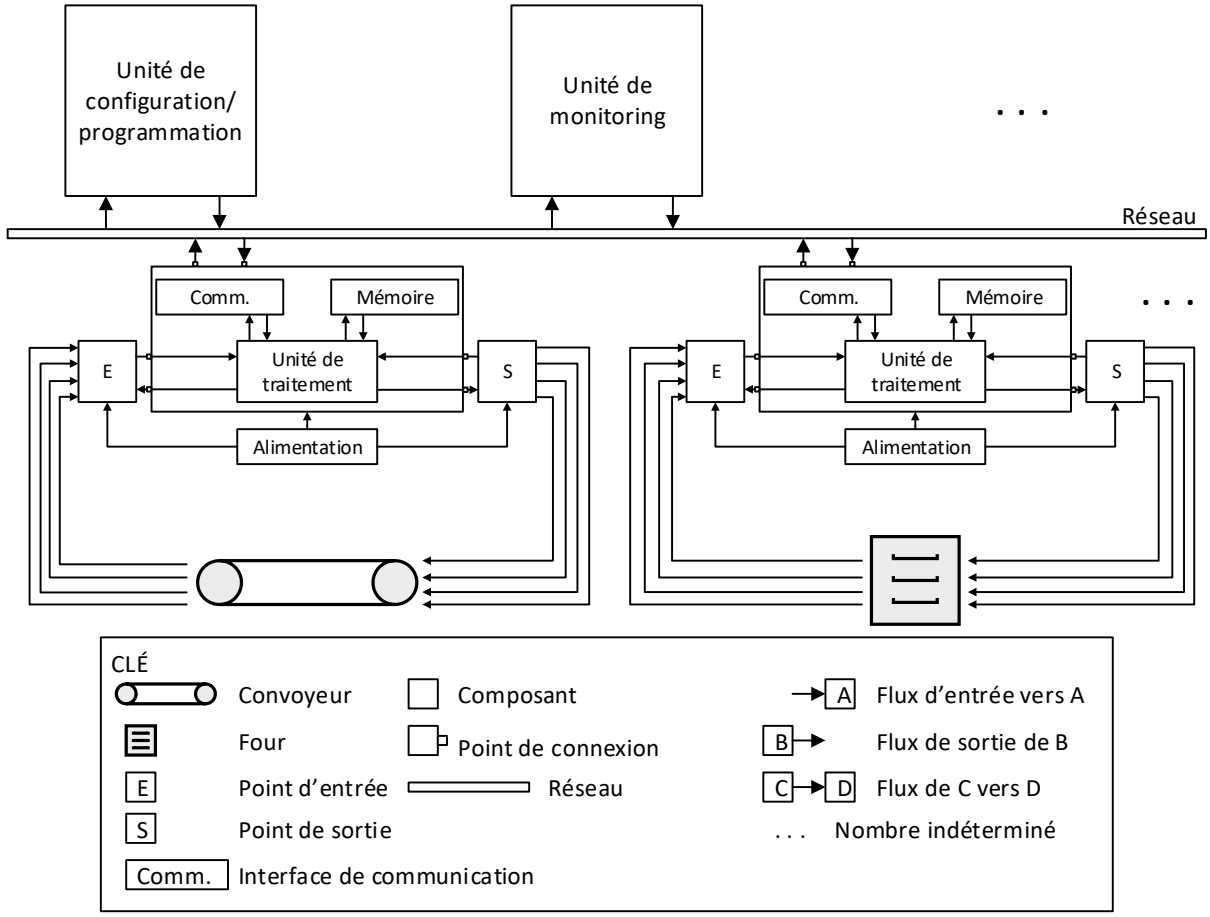


Figure 2 Réseau de systèmes électroniques programmables

Tableau 2 Composants de la Figure 2

Élément	Description
Unité de configuration/ programmation	Une chaîne d'outils pour la programmation, la configuration et le monitoring, pouvant échanger la logique de contrôle ou d'autres données avec la mémoire du système électronique programmable.
Unité de monitoring	Outil de monitoring, pour l'affichage des indicateurs.
Réseau	Bus de communication externe.
Contrôleur du convoyeur	Le système électronique programmable contrôlant la fonction du convoyeur.
Contrôleur du four	Le système électronique programmable contrôlant la fonction du four.

Les contrôleurs sont configurés et programmés pour assurer la cohésion d'une chaîne de fabrication d'aliments. Des unités de configuration et programmation permettent de conditionner et de structurer le système opérationnel. D'autres unités de monitoring permettent le suivi des opérations.

1.2 Sécurité fonctionnelle

La sécurité fonctionnelle agit en prévention des dangers inhérents à l'opération d'un automatisme pour la vie, l'environnement, les acquis économiques et le matériel [13], [23], [28]. La fonction de sécurité s'ajoute au processus opérationnel après une analyse des dangers associés à l'opération du système [23]. Le standard CEI 61508 laisse les parties prenantes libres de déterminer le degré de dangerosité acceptables en fonction de leur contexte d'opération [28]. La norme propose des techniques et mesures pour déterminer les exigences d'un système É/É/ÉP en vue d'opérer à un niveau acceptable de sécurité [13].

Les systèmes électroniques programmables assortis d'une fonction de sécurité s'illustrent typiquement comme suit :

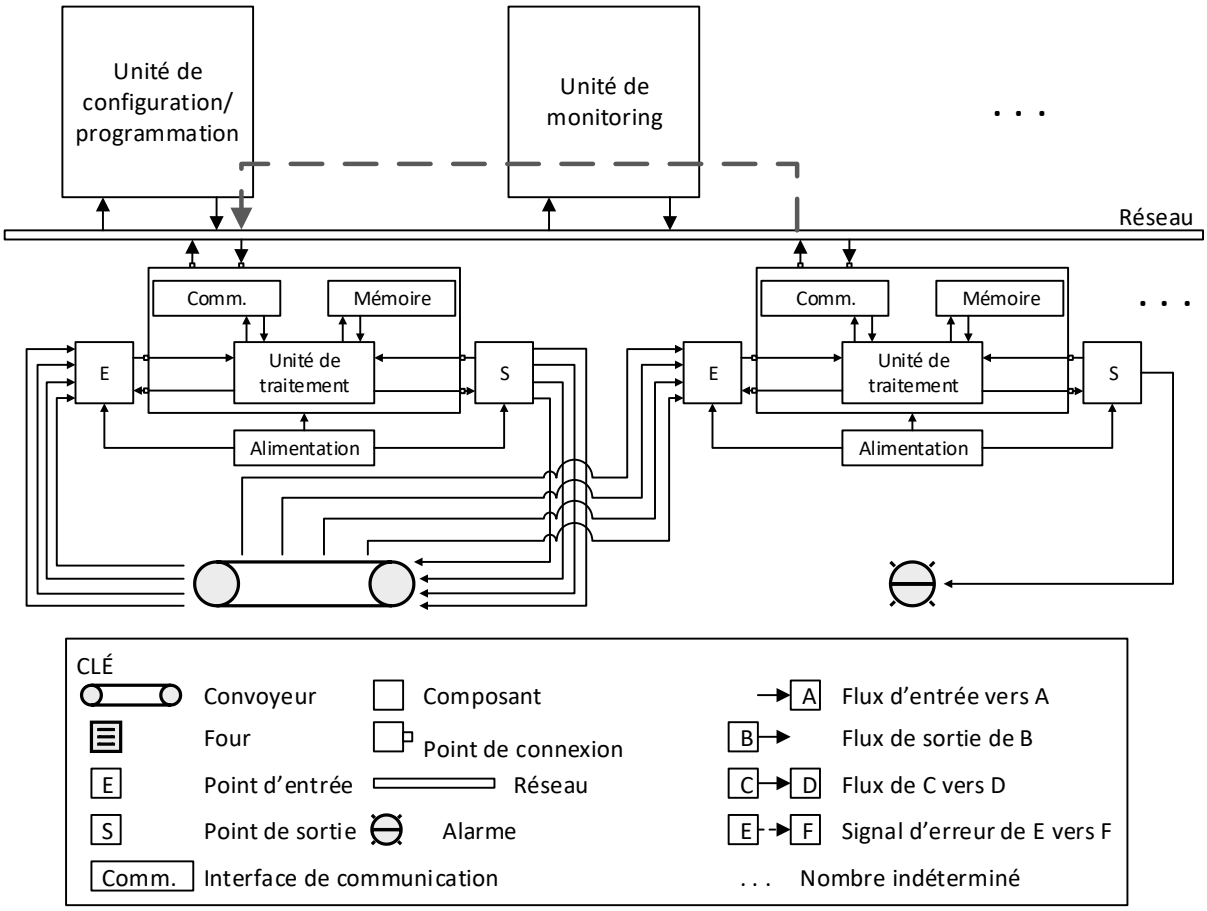


Figure 3 Réseau de fonction de sécurité d'architecture 1oo1

Tableau 3 Composants de la Figure 3

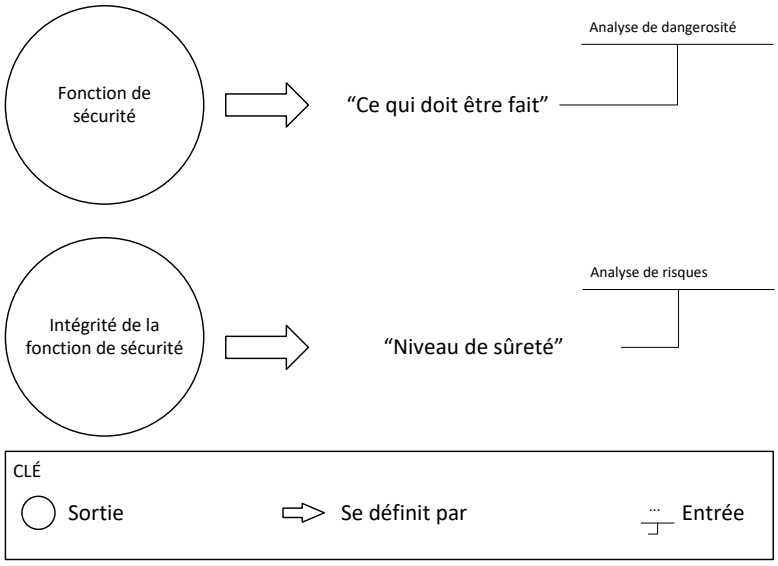
Élément	**Description**
Unité de configuration/ programmation	Une chaîne d'outils pour la programmation la configuration et le monitoring, pouvant échanger la logique de contrôle ou d'autres données avec la mémoire du système électronique programmable.
Unité de monitoring	Outil de monitoring, pour l'affichage des indicateurs.
Réseau	Bus de communication externe.
Contrôleur du convoyeur	Le système électronique programmable contrôlant la fonction du convoyeur.
Fonction de sécurité	Le système électronique programmable contrôlant la fonction de sécurité appliquée au convoyeur.

Des capteurs de la fonction de sécurité acquièrent des données permettant un diagnostic continu du système d'opération. La fonction détermine si les signaux révèlent un état de sécurité levé, qui découle d'un risque potentiellement dangereux [9, Chap. 3.1.13]. Une fonction de sécurité en état potentiellement dangereux lève très souvent une alarme et envoie des commandes de compensation aux autres systèmes.

1.2.1 Conception d'une fonction de sécurité

L'implantation d'une fonction de sécurité dépasse la seule perspective unitaire de son champ d'opération [23]. Les parties prenantes se mobilisent au sein d'un comité de gestion. Ce comité définit ce qui doit être fait en prévention ou en réponse aux situations dangereuses qui découlent du des opérations industrielles et convenir d'un degré d'intégrité acceptable dans les circonstances [28].

Le processus de conception s'illustre comme suit :



EXEMPLE:
Fonction de sécurité: En vue de prévenir le bris du réservoir « X », ouvrir la valve « Y » en moins de deux secondes lorsque la pression atteint le seuil de 2,6 bars.

Intégrité de la fonction de sécurité : Fonction de niveau « SIL2 ».

Figure 4 Conception d'une fonction de sécurité (inspiré de [23, Fig. 3]**)**

Tableau 4 Entités de la Figure 4

Élément	Description
Fonction de sécurité	Sortie du requis de sécurité fonctionnelle.
Intégrité de la fonction de sécurité	Sortie du requis de performance de sécurité fonctionnelle.
« ce qui doit être fait »	Requis de sécurité du processus en opération.
« niveau de sécurité »	Requis de solidité de la fonction elle-même.
Flèche vers la droite	Le contenu du cercle à gauche produit par l'assertion faite à droite.
Déterminé par…	Entrée du requis.

La description de « Ce qui doit être fait » donne forme à la fonction de sécurité. Cette description découle de l'analyse des menaces que représente le processus pour la vie, l'environnement, les acquis économiques et le matériel [23]. Le comité de gestion reste toutefois libre de déterminer les seuils de dangerosité acceptables pour le contexte d'opération [6].

L'indicateur « *performance de sécurité* » produit le niveau SIL de la fonction de sécurité. Cet indicateur découle de l'analyse des risques que représentent les composants impliqués dans la fonction. Le calcul diffère selon la catégorie de défaillance et le degré de sollicitation du processus [23][6].

La norme répartit les défaillances en deux catégories [12]:

- Les défaillances systématiques – proviennent d'accumulation d'erreurs induites en phase de développement. Pour l'essentiel, le calcul du potentiel de dangerosité se base sur la rigueur méthodologique (Figure 24), ainsi que sur des statistiques de défaillances [23] [8]. La démonstration de rigueur augmente le degré de confiance et l'indice SIL [23].
- Les défaillances aléatoires – proviennent de l'usure et du bris matériel dans un contexte d'opération normale. Le calcul du potentiel de dangerosité s'appuie exclusivement des statistiques de défaillances. La probabilité de défaillance dangereuse détermine l'indice SIL [23] [6], avec un degré de confiance accru lorsqu'une couverture diagnostique est documentée par le fabricant [23].

Le degré de sollicitation influence l'échelle de tolérance aux défaillances de chaque niveau SIL [23]. Une forte sollicitation diminue la tolérance alors qu'une demande faible l'augmente.

La norme distribue l'échelle de sollicitation en deux groupes :

- Sollicitation sur demande – lorsque la fonction opère de façon intermittente. La norme ne précise pas de fréquence précise [2]. Le seuil est déterminé à l'analyse de dangerosité [28]. La mesure se présente sous la forme d'une probabilité de défaillance sur demande (*Probability of Dangerous Failure on Demand* ou PFD) [23].
- Sollicitation forte ou continue – lorsque la fonction opère fréquemment ou de manière continue. Là aussi, la norme ne précise pas de fréquence précise [2] et le seuil est déterminé à l'analyse de dangerosité [28]. Cette mesure se présente sous la forme d'une probabilité de défaillance dangereuse à l'heure (*Probability of Dangerous Failure per Hour* ou PFH) [23].

Les statistiques de défaillances calculées en fonction de leur catégorie (*systématique* ou *aléatoire*) sont converties en probabilité d'échec [23] [2]. La norme fixe l'échelle SIL en fonction de la probabilité d'échec modulée à l'échelle de sollicitation (sur demande, forte ou continue).

Les indices SIL se répartissent ainsi [23] :

Tableau 5 SIL pour fonction exécutée sur demande

SIL	Probabilité d'échec (PFD)
4	≥ 10^{-5} P < 10^{-4}
3	≥ 10^{-4} to < 10^{-3}
2	≥ 10^{-3} to < 10^{-2}
1	≥ 10^{-2} to < 10^{-1}

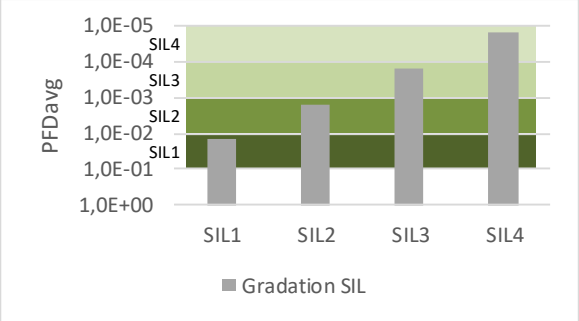
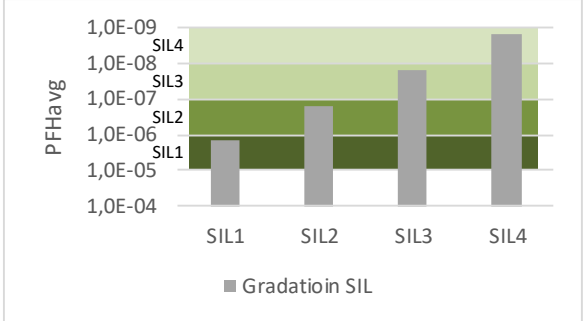


Tableau 6 SIL pour fonction continue ou fortement sollicitée

SIL	Probabilité d'échec / heure (PFH) [29]
4	≥ 10^{-9} to < 10^{-8}
3	≥ 10^{-8} to < 10^{-7}
2	≥ 10^{-7} to < 10^{-6}
1	≥ 10^{-6} to < 10^{-5}



Le comité de direction décide le niveau SIL ciblé en fonction du niveau de danger acceptable que représente le processus opérationnel. Les choix méthodologiques et de composants seront fortement influencés par la probabilité d'échec combinée aux stratégies de réduction de risques [23], [28].

1.2.2 Exigences concernant les logiciels

À l'aube des années 1980, les standards de cycle de vie gravitant autour des systèmes électroniques programmables focalisent principalement sur le logiciel [23]. Ils entretiennent l'objectif commun de produire des composants d'un degré de sécurité fonctionnelle adéquat en fonction du contexte d'opération [13]. La définition du niveau approprié relève toutefois du comité directeur ou des attentes d'auditeurs externes [13]. Dès lors, les exigences qui en découlent sont susceptibles de varier en fonction du contexte [28].

À l'opposé, la figure suivante démontre que la plupart des défectuosités logicielles sont induites en phase de conception :

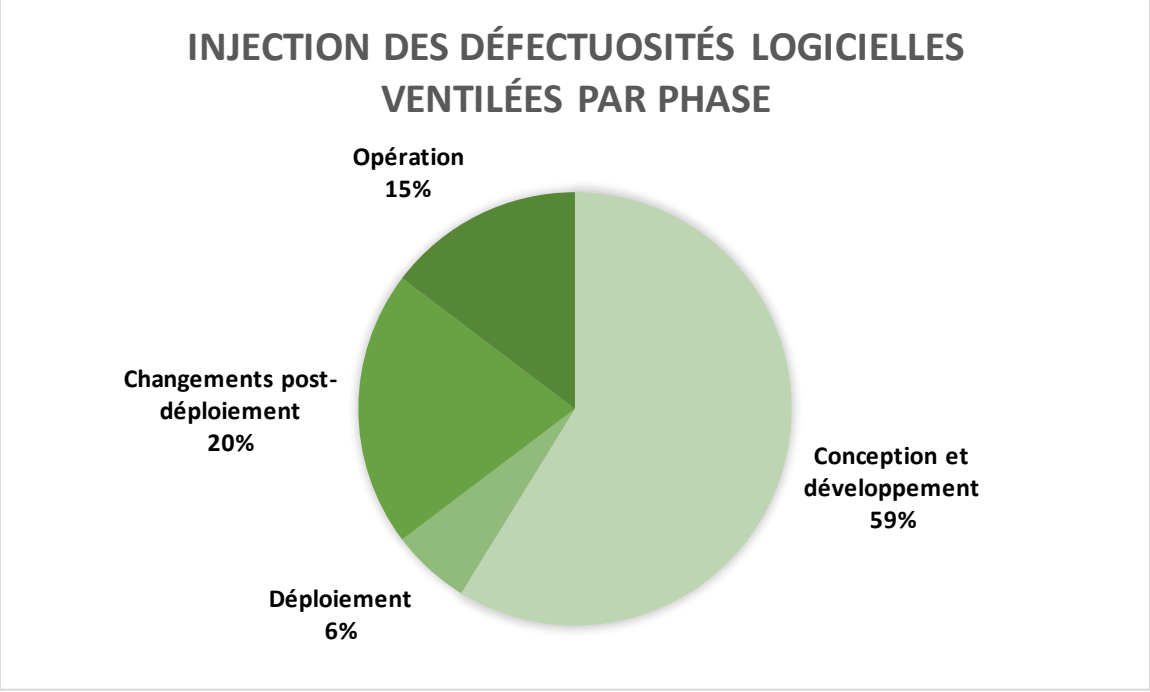


Figure 5 Statistiques d'injection des défectuosités logicielles ventilées par phase [23]

Le comité CEI 61508 classifie les anomalies du logicielles parmi les *défaillances systématiques* qu'il convient de contrôler par le constat d'une rigueur susceptible d'augmenter le degré de confiance et l'indice SIL [23]. La maitrise et le contrôle des défaillances logicielles imposent le consensus d'une approche holistique qui englobe l'ensemble du cycle de vie de sécurité (Figure 24 et Figure 25) [13], [23].

1.3 Capabilité systématique

L'indice SC informe sur la mesure du degré de confiance qu'un composant de la fonction de sécurité satisfait aux exigences de sécurité fonctionnelle. Elle se calcule différemment entre le matériel et le logiciel [23].

Le matériel s'accompagne de statistiques de couverture diagnostic (*Diagnostic Coverage* ou DC [9, Chap. 3.8.6]) qui servent au calcul de l'indice SC. La norme déconseille l'usage de matériel distribué sans une telle documentation. Le niveau dépend d'un calcul appliqué sur chaque composant matériel et combiné à l'échelle du système.

À l'opposé, le logiciel est soumis à un nombre élevé de facteurs pouvant causer des anomalies systématiques susceptibles d'affecter l'indice SC. Il n'existe aucun algorithme en mesure de déterminer les indicateurs de dangerosité. Plusieurs recherches proposent des modes de calcul en fonction des analyses statiques du code source [30], [31] ou des statistiques de défaillances [24], [26], [32], [33], mais la pertinence est laissée au jugement des auditeurs. Les documents de la norme CEI 61508 proposent des listes de contrôle appliquées au cycle de vie du logiciel et aux outils employés pour le développer ou l'entretenir. Les listes de contrôle imposent un moyen d'expliciter les bonnes pratiques de développement et des outils peuvent aider l'opérationnalisation [26], mais l'auditeur exige les preuves permettant de les avérer en vue de l'homologation [8, Chap. Annexe C].

1.3.1 Cycle de vie logiciel

Pour toute classe de logiciel, la plus forte probabilité d'anomalies reste associée à la phase de conception logicielle. Les concepteurs de la norme imposent une politique de vérification et

validation stricte, dès qu'un logiciel en développement doit s'intégrer à une fonction de sécurité [8, Chap. 7.1.2.4]. Le standard CEI 61508 propose le modèle en V illustré ci-dessous :

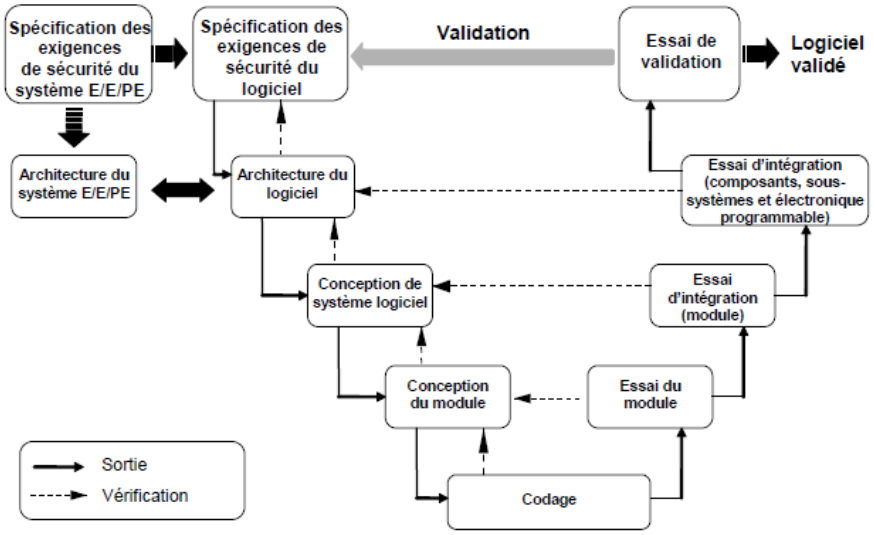


Figure 6 Capabilité systématique du logiciel et cycle de vie de développement (modèle en V) ([8, Fig. 6]**)**

La Figure 6 présente un modèle où chaque sortie de processus sert d'entrée au suivant. Le résultat en sortie d'un processus est vérifié en fonction de ses entrées. Les jeux d'essais permettent d'attester la conformité du matériel issu du processus suivant [8, Chap. 7.3.2]. Par exemple, à la quatrième rangée, le processus « conception de module » va de pair avec son jeu d'essai, mais ce dernier atteste la conformité du processus « codage de module » qui découle de cette même conception. Chaque échelon forme alors un cycle de validation cohérent avec le précédent.

Par le cycle de vie seul, les fabricants obtiennent rarement la capabilité systématique ciblée, pour leurs composants logiciels [5]. La plupart devront recourir aux architectures redondantes pour atteindre leur objectif de sécurité fonctionnelle à l'aide du facteur de composition [2], mais s'exposent du même coup à de nouveaux risques [26], [34].

1.3.2 Redondance

En sécurité fonctionnelle la redondance combine plusieurs canaux multiplexés en vue de prévenir une défaillance. La documentation décrit le type de redondance par la forme « MooN », d'où « N » canaux exécutent la fonction pour « M » [9, Chap. 3].

En revanche, l'illustration ci-dessous propose une architecture 1oo2, soit la forme de redondance la plus simple, où deux canaux accomplissent la fonction de sécurité pour ensuite la multiplexer en un seul :

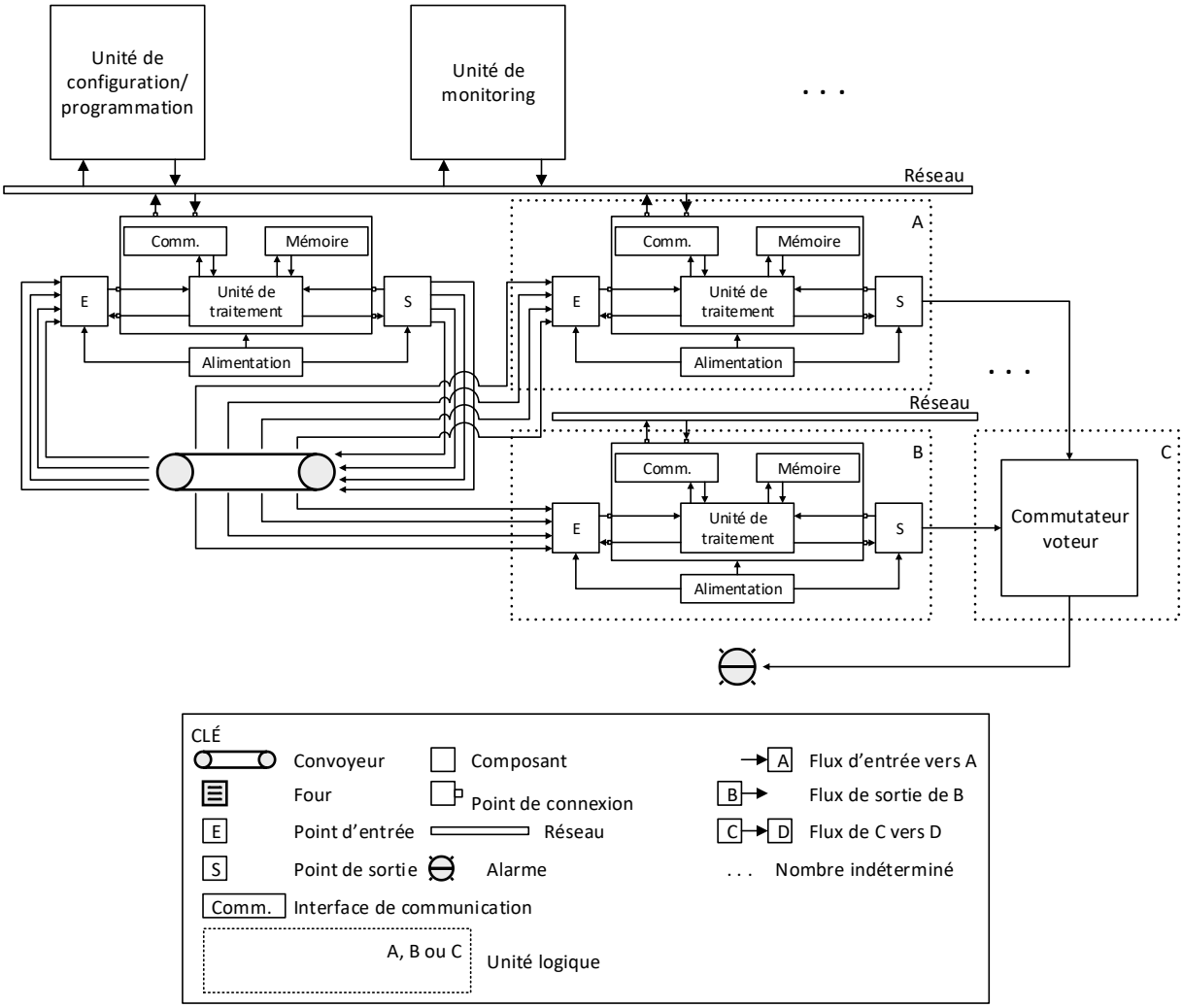


Figure 7 Architecture redondante de type 1oo2

Les capteurs du convoyeur se connectent sur deux canaux. Les signaux traités par des systèmes électroniques programmables distincts représentés par les unités logiques « A » et « B ». La sortie de chaque unité suit son cours vers l'unité logique « C » qui combine les sorties sur un seul canal.

Pour comparaison, la Figure 3 illustre une structure 1oo1. Un canal accomplit le contrôle pour une seule sortie, et aucune redondance ne s'applique. Une défaillance du canal compromet la fonction de sécurité [11, Chap. B.3.2.2.1]. Dans l'exemple de la Figure 7, une structure 1oo2, l'intégrité de la fonction n'est compromise que si les deux canaux tombent en défaillance [11, Chap. B.3.2.2.2] ; l'échec de l'un est corrigé par l'autre, à moins qu'une anomalie ne se présente simultanément sur les deux.

Un système redondant composé de matériel et logiciels identiques induit une conception optimiste, qui tend à produire des systèmes où les unités redondantes restent sensibles aux mêmes stress et deviennent susceptibles de tomber en défaillance simultanément [34], [35]. Ce phénomène est une défaillance de cause commune (*Common Cause Failure* ou CCF) contre laquelle le mode de défense consiste à produire un système redondant équipé de composants équivalents, mais conçus avec une diversité technologique [34], [35]. La norme illustre l'interférence des CCF à l'aide de diagrammes de fiabilité [11, Chap. B.2.1] :

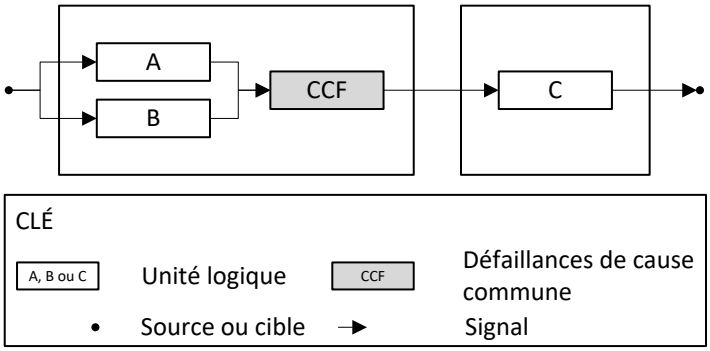


Figure 8 Diagramme de fiabilité pour la vue présentée dans Figure 7

La redondance permet la détection des défaillances non-détectées lors des tests, alors que la diversité ajoute une défense contre les CCF [34], [36]. La documentation emploie β pour exprimer la proportion de CCF lors des calculs.

Le tableau ci-dessous compare l'effet de la redondance sur la probabilité de défaillance d'un système électronique programmable en mode de fonctionnement continu ([11], tableau B.13) :

Tableau 7 Impact de l'architecture sur la probabilité de défaillance

Architecture	DC	λ_D	β	PFH	SIL
1oo1	90%	2,50E-5	N/A	2,50E-06	SIL 1
1oo2	90%	2,50E-5	20%	5,40E-07	SIL 2
1oo2	90%	2,50E-5	10%	3,00E-07	SIL 2
1oo2	90%	2,50E-5	2%	1,00E-07	SIL 3

Le Tableau 7 permet de constater que la redondance permet de rehausser le niveau SIL d'un système. En prime, l'emploi de mesures de défenses efficaces contre les CCF amplifie le degré SIL dans une proportion suffisante pour le porter à l'échelon suivant. À l'échelle d'un composant, comme un *Soft PLC*, la mesure se présente en indice SC, plutôt qu'en indice SIL [37]. L'hypothèse à l'étude propose que la diversité de compilateurs offre une défense suffisamment forte pour réduire la proportion de CCF (β) à 2 %, et relever la capabilité sémantique de SC3 à SC4 sans augmentation de coûts significative.

1.4 Rôle des compilateurs

Le compilateur traduit une spécification en code binaire optimisé pour l'unité de traitement [38]. Les artéfacts s'inscrivent dans sa mémoire en vue de leur exécution par la fonction de sécurité.

Le produit du compilateur se positionne dans le PLC et au cœur de l'automatisme. Toute anomalie injectée silencieusement par le compilateur devient susceptible de compromettre la fonction de sécurité et de la rendre inapte à répondre correctement lorsque le processus surveillé se montre instable. Sans une fonction de sécurité viable, le risque dangereux se transforme en enjeu, voire en catastrophe.

L'illustration ci-dessous démontre son impact sur un système électronique programmable :

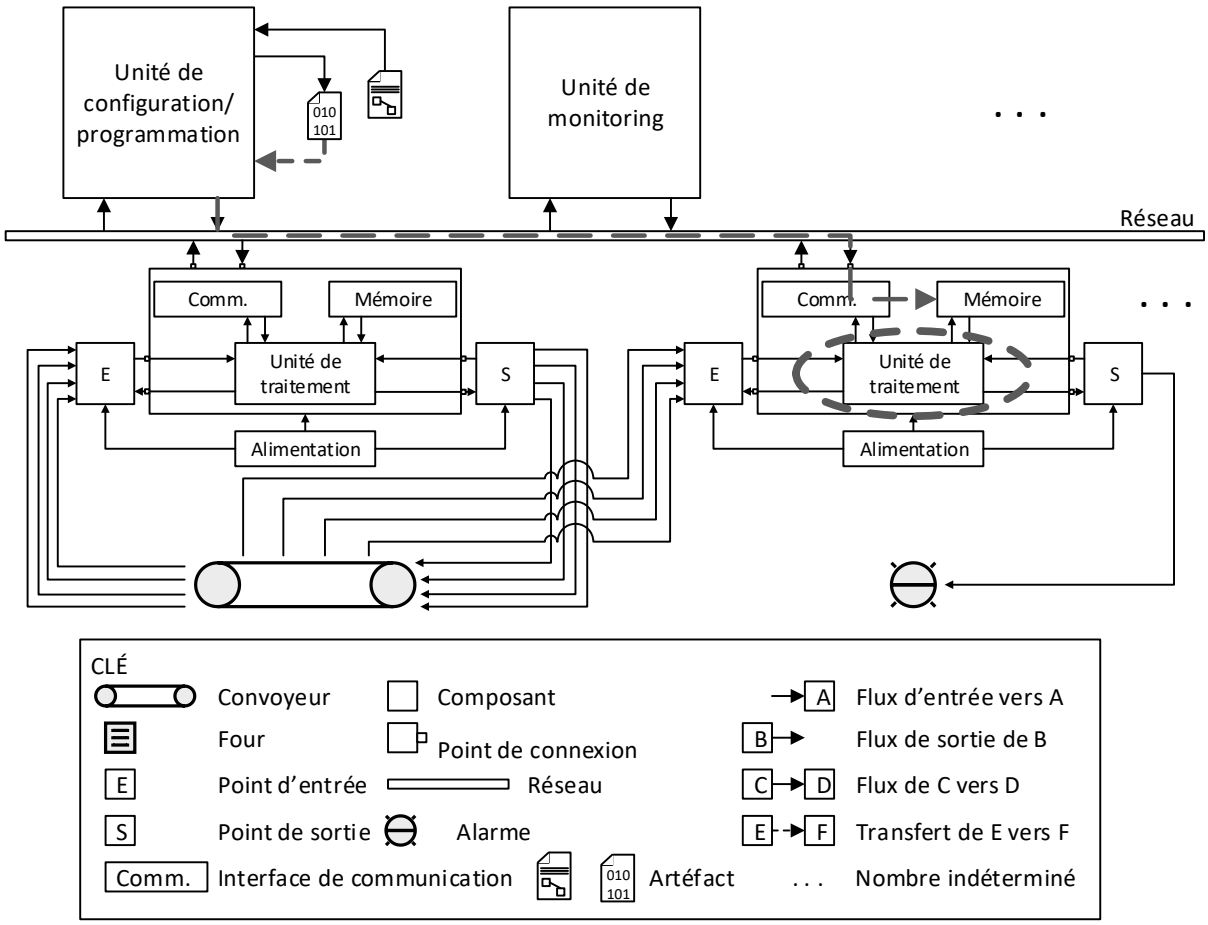


Figure 9 Compilation et inscription d'algorithmes dans une fonction de sécurité

Tableau 8 Composants de la Figure 9

Élément	**Description**
Unité de configuration/ programmation	Une chaîne d'outils pour la programmation, la configuration et le *monitorage*, capable d'échanges de logique de contrôle ou d'autres données avec la mémoire du système électronique programmable.
Artéfacts consommés ou produits par le compilateur	Code source de programmes, code binaire généré.
Inscription de la logique de contrôle sur la fonction de sécurité.	Impact sur la fonction de sécurité.
Unité de monitoring	Outil de monitoring, pour l'affichage des indicateurs.
Réseau	Bus de communication externe.
Contrôleur du convoyeur	Le système électronique programmable contrôlant la fonction du convoyeur.
Fonction de sécurité	Le système électronique programmable contrôlant la fonction de sécurité appliquée au convoyeur.

Dans la Figure 9, la fonction de sécurité reçoit du code exécutable susceptible d'en affecter le comportement.

Pour le standard CEI 61508, le compilateur compte parmi les outils « hors ligne » de catégorie T3, puisqu'il produit le logiciel embarqué qui détermine le comportement de la fonction de sécurité [9]. La norme distribue les outils « hors ligne » en trois catégories [9] :

- T1 – Ne produis aucune donnée susceptible d'affecter le code exécutable. Par exemple, les éditeurs textes, les navigateurs de traces...
- T2 – Vérifie les erreurs dans le code exécutable ou la configuration, comme les outils d'analyse statique ou les simulateurs.
- T3 – Produit des données susceptibles d'affecter le code exécutable, comme les compilateurs, y compris ceux qui émettent des programmes optimisés pour une machine virtuelle.

Les spécifications d'une fonction de sécurité destinée au compilateur se présentent sous la forme de code source conforme à une structure sémantique, syntaxique et lexicale stricte qu'on appelle communément : Langage de programmation [39]. Le compilateur extrait, analyse et transforme la spécification en suivant plusieurs cycles et génère du code optimisé pour une unité de traitement ou une machine virtuelle précise [19], [39].

Les langages de programmation se répartissent en deux grandes catégories :

- Dédiés — des langages à variabilité limitée, dont l'étendue des possibilités se limite aux besoins applicatifs [9], [17]. Ils suivent très souvent des règles de typologie stricte [40]. Ces caractéristiques facilitent leur utilisation et l'attestation de sécurité fonctionnelle. Les quatre langages du standard CEI 61131 ou le langage par blocs fonctionnels distribués du standard CEI 61499 peuvent servir d'exemples [8], [41].
- Généralistes — dont l'étendue des possibilités transcende les besoins applicatifs [17]. Certains, comme le langage ADA, suivent des règles de typologie strictes [40] qui facilitent l'attestation de sécurité fonctionnelle [12]. D'autres, comme le langage C, sont également très répandus dans l'industrie des systèmes électroniques programmables, mais problématiques pour l'homologation de sécurité fonctionnelle

[12]. L’envergure des communautés permet toutefois de tirer parti de plusieurs recherches en vérification de compilateurs [42] — [44].

L’homologation de sécurité fonctionnelle reste exigeante, tant avec les compilateurs de langages dédiés que généralistes [45]. Les communautés plus abondantes pour les généralistes offrent plus de possibilités de recherche en vérification formelle [42], [46] privilégiées par la norme CEI 61508 [12]. Les communautés restent moins abondantes autour des langages dédiés, mais leur spécialisation pour certains besoins applicatifs rend toutefois l’homologation possible [12], [43], [47].

Chapitre 2

Revue de la littérature

Ce chapitre donne un aperçu de l'état des connaissances en vérification de compilateurs. Il décrit la méthodologie de recherche et résume les articles parcourus.

2.1 Méthodologie de recherche

Les recherches s'exercent avec l'« Outil Découverte » de l'Université de Sherbrooke et le site *Google Scholar*. Les deux ont des possibilités complémentaires.

L'« Outil Découverte » permet de filtrer les articles par terme de sujet, titre ou résumé. Des recherches ciblées obtiennent des résultats plus pertinents.

Le service *Google Scholar* permet de ratisser plus large et extrait un grand nombre d'œuvres (dans une proportion de 15 pour 1), mais confond les langues et les domaines d'études.

2.1.1 Mots-clés utilisés

L'essai implique la vérification de compilateurs pour homologuer la conformité d'une fonction de sécurité aux requis de la norme CEI 61508. Les mots clés les plus utilisés pour la recherche s'énumèrent comme suit :

- *« IEC 61508 »* – L'outil d'aide à la recherche extrait 3 143 articles parus de 1988 à 2016 inclusivement. Les articles obtenus abordent les techniques de contrôle de risques et de dangerosité. Le domaine de la vérification de compilateurs s'y perd dans la multitude. Le service *Google Scholar* propose près de 7 640 œuvres, mais beaucoup sortent du domaine de l'informatique toute langue de publication confondue.
- « *61508 common cause failure* » - L'outil d'aide à la recherche extrait 661 articles parus de 1985 à 2016 inclusivement. Les articles obtenus abordent les techniques

de contrôle et de défense contre les défaillances pour causes communes. Le service *Google Scholar* propose près de 4 470 œuvres, mais beaucoup sortent du champ d'intérêt de l'étude.

- *« Compiler Verification »* — L'outil d'aide à la recherche extrait 899 articles parus de 1976 à 2015 inclusivement. Les articles obtenus couvrent plusieurs techniques de vérifications de compilateurs en confondant tous les langages et les types de compilateurs. Le service *Google Scholar* en propose près de 103 000, toutes langues de publication confondues.
- *« Formal compiler verification »* — L'outil d'aide à la recherche extrait 246 articles parus de 1981 à 2015 inclusivement. Les articles obtenus se recoupent avec la recherche « Compiler Verification ». Le service *Google Scholar* en propose près de 99 600, toutes langues de publication confondues.
- *« Verifying Compiler »* — L'outil d'aide à la recherche extrait 204 articles parus de 1978 à 2016 inclusivement. Les articles obtenus se confondent en genre, mais sont tous centrés sur la vérification de compilateurs. Le service *Google Scholar* en propose près de 896, toutes langues de publication confondues.
- *« DSL Compiler »* — L'outil d'aide à la recherche extrait 95 articles parus de 1992 à 2015 inclusivement. Les articles obtenus abordent tous la conception de langages dédiés. Le service *Google Scholar* en propose près de 372, toutes langues de publication confondues.
- *« GPL Compiler »* — L'outil d'aide à la recherche extrait 60 articles parus de 2001 à 2015 inclusivement. L'objectif consiste à trouver des sources à propos des compilateurs pour langages généralistes (*General Programming Languages* ou GPL). La plupart des articles obtenus abordent les algorithmes de simulation et traitements des signaux, ce qui les sort du champ d'intérêt. Le service *Google Scholar* en propose près de 51, abordant les langages généralistes comme sujet de comparaison avec les langages dédiés.

2.2 Internet des Objets (IdO)

Gartner explique dans [21] la forte expansion et généralisation de l'Internet des choses ainsi qu'un aperçu des risques associés à leurs intrusions dans la vie domestique. L'institution prédit la mise en réseau d'au-delà de 25 milliards objets d'ici 2020, dont six milliards 250 millions pour les transports seulement. L'industrie connait déjà une révolution analogue [48]. Les risques dangereux se posent autant pour le marché domestique qu'industriel.

Dans un autre article [49], Gartner remarque l'influence de la forte croissance de l'IdO sur la revitalisation de certain langages fortement documentés et standardisés, en l'occurrence, le C et le C++. Gartner explique cette tendance par les systèmes existants auxquels se greffent les nouveaux objets, dans une quête continue d'optimisation des retours sur investissements (Return On Investment ou ROI).

Dans leur article [48], YangSun, Junho et Yunsik abordent plus spécifiquement l'importance du compilateur pour la fabrication d'applications embarquées sur un composant IdO. Comme beaucoup d'auteurs, ces derniers étudient essentiellement les modes de défense à l'encontre d'intrusions malveillantes. Les auteurs proposent cependant des techniques analogues aux systèmes de vérification statique décrits par Yang dans son autre article [1] qui traite de la détection de bogues dans les compilateurs.

2.3 Système électronique programmable

Dans son manuel [18], Bolton introduit au concept de système électronique programmable. L'auteur décrit l'architecture des systèmes industriels, les composants matériels et les langages de programmation d'automaticiens.

Avec l'article [19], Zhou Chunjie et Chen Hui abordent le développement d'une machine virtuelle avec plus de détails. Les auteurs décomposent et décrivent les niveaux d'abstractions, situés entre le matériel et l'automatisme implémenté avec le langage CEI 61131-3. Leur décomposition, plus concrète que [20], permet de situer l'impact d'un compilateur sur l'intégrité de la machine virtuelle et spécialement lorsqu'elle est susceptible d'exécuter une fonction de sécurité.

Comme Croker l'explique dans son article [50], la plupart des logiciels embarqués se conçoivent en langage C ou C++. Cette tendance se perpétue avec les machines virtuelles de type *Soft PLC*. Les méthodes formelles de vérification demeurent au stade académique [38] et la vérification par traçabilité ascendante implique de renoncer aux meilleurs atouts des compilateurs C ou C++ [43]. Bien que la norme CEI 61508 semble privilégier les méthodes formelles de vérification [12, Chap. C.4.1] ou la traçabilité ascendante [12, Chap. C.4.4.1], il existe la voie de la diversification logicielle [38], moins fréquentée parce qu'elle implique une redondance des machines virtuelles.

2.4 Sécurité fonctionnelle

Le standard CEI 61508 aborde le sujet de la sécurité fonctionnelle avec une abondante documentation distribuée sur sept volumes [7], [9]–[12], [41], [51]. Les résultats de calculs présentés par l'essai proviennent tous des tableaux produits dans ces documents et plus spécialement [11].

Quelques auteurs comme Ron Bell et Simon Brown proposent des articles d'introduction [13], [23] qui abordent les concepts de base de la sécurité fonctionnelle et du standard CEI 61508. Les deux auteurs offrent un survol de l'ensemble du standard. À l'inverse, Bill Black donne une vision complémentaire de ces introductions par un article [28] où il énumère les non-dits de la norme et spécialement l'ambiguïté qui gravite autour de l'analyse de dangerosité.

Dans leur article sur la combinaison des niveaux SIL [2], Langeron, Barros, Grall et Bérenguer abordent la technique de combinaison des degrés SIL. Ils démontrent l'effet de levier que procure la redondance pour combiner les systèmes en vue d'optimiser l'intégrité et le niveau SIL d'une fonction de sécurité. Les mêmes techniques s'appliquent pour le calcul d'indice SC (chapitre 1.3 Capabilité systématique).

2.5 Capabilité systématique

Dans leur article [30], Tang et Lu tentent d'établir un modèle de fiabilité commun entre le matériel et le logiciel. Leur but consiste à objectiver les mesures d'intégrité du logiciel. Quant

aux auteurs Xia et Yan, leur article [31] décrit certaines techniques pour calculer de probabilité de défaillances logicielles équivalente à l'indice λ_D du matériel. Les deux articles cherchent à établir l'indice SC des composants logiciels avec des formules analogues à celles s'appliquant au matériel.

Les trois auteurs Mayr, Plösch et Saft proposent dans [26] des moyens d'outiller la gestion du cycle de vie logiciel. Ils déterminent les points qui peuvent s'intégrer à un système d'intégration continue pour automatiser la production de rapports utiles à l'homologation de l'indice SC pour un composant logiciel.

2.5.1 Calcul de probabilité de défaillance

Dans son article [32], Alena Griffiths soulève la question des intervalles d'essais appliqués au logiciel. L'auteure considère l'importance des tests cadencés en présence d'usures matérielles et tente un rapprochement avec le caractère incorporel du logiciel. Son étude ne conclut pas sur une méthode en particulier, mais permet d'expliciter les difficultés que représente la conception d'un modèle de qualité commun pour le matériel et le logiciel.

Avec son article [24], Peter B. Ladkin ajoute un complément d'information (aux propos de Alena Griffith [32]) sur le poids relatif du logiciel dans l'homologation SIL d'un système. L'auteur démontre par un exemple pourquoi, en fin de compte, c'est principalement le matériel qui détermine le PFD ou le PFH. L'article explique d'un autre angle que le logiciel se plie au contexte, ce qui permet de déduire une nuance capitale entre les indices SC et SIL. L'indice SC prescrit la cible SIL du système auquel un élément peut s'intégrer ; un système doit rétrograder son degré SIL à l'échelle SC minimale de ses composants.

Dans leur article [33], les auteurs T. Fujiwara, M. Kimura, Y. Satoh et *al* soutiennent, qu'au contraire, la détermination du degré SIL d'un logiciel devrait suivre une courbe de maturité. Les auteurs proposent que tout logiciel soumis à des tests réguliers au cours de son cycle de vie, devient susceptible de réduire la probabilité de défaillances non-détectées et d'augmenter l'indice SIL en proportion. Leur position semble opposée à Alena Griffiths (dans [32]) et Peter B. Ladkin (dans [24]), mais pour peu qu'un lecteur avisé remplace SIL par SC, l'article permet d'objectiver le calcul de capabilité systématique du logiciel.

Quant à D. John Satur, J. Kim, M. ki Bae et *al*, leur article [5] décrit une technique qui découle des analyses d'arbres de défaillance pour déterminer la probabilité de défaillance d'un système. Les auteurs appuient le calcul de probabilité de défaillances dangereuses sur les probabilités bayésiennes et développent un modèle intégré pour le logiciel et le matériel.

2.5.2 Redondance et diversité technologique

Dans leur article [34], M. A. Lundteigen et M. Rausand abordent la problématique des défaillances de cause commune (« CCF ») et leur impact sur le calcul du niveau SIL d'un système avec architecture redondante. L'écrit caractérise le phénomène et propose la diversité technologique comme moyens de défense.

Pour complémenter les propos de M. A. Lundteigen et M. Rausand, les auteurs A. C. Torres-Echeverría, S. Martorell et *al*, ajoutent, dans l'article [35], la notion de coût au calcul de défaillances de cause commune. La gravité des défaillances et les choix de modes de défense sont fortement modulés par l'échelle de coût. Leur regard joint la variable du coût économique aux critères de sélections technologiques.

Finalement, dans leur article [37], H. Jahanian et Q. Mahboob combinent les facteurs risque et coût, associés au choix d'une cible de niveau SIL. Les auteurs se réfèrent au processus de sélection que tout individu « rationnel » doit emprunter une voie et supposent que la plupart des gens adoptent l'option optimale, qui lui procure le plus de valeur pour le moins de risque possible. C'est un mode de réflexion complémentaire à l'opinion de M. A. Lundteigen et M. Rausand, dans [34], ou de A. C. Torres-Echeverría, S. Martorell et *al*, dans [35], qui prend en compte la valeur escomptée par chaque alternative.

L'essai s'appuie sur les modes de défense contre les CCF, pour déterminer si la diversité de compilateurs peut augmenter la valeur d'une machine virtuelle embarquée.

2.6 Vulnérabilité des compilateurs

Dans leur article [52], Eric Eide et John Regehr exposent une problématique avec les déclarations décorées de l'attribut « *volatile* », issue d'ambiguïtés parues dans la norme du langage « C ». Ce qualificatif permet d'informer le compilateur qu'une entité extérieure peut

modifier la donnée. Les logiciels de base (OS), les systèmes embarqués et spécialement les systèmes critiques l'utilisent fréquemment. Les auteurs ont constaté que 13 compilateurs parmi les 13 analysés ne tiennent pas compte de l'attribut et optimisent l'accès au bloc de mémoire.

Les auteurs constatent que les compilateurs tentent par tous les moyens de classer les blocs de mémoire dans des registres, et de réordonner les calculs et d'éliminer ceux qui paraissent redondants. Avec le qualificatif « volatile », le compilateur doit éviter d'optimiser l'accès à cette zone comme il le ferait pour une autre plage de données [52]. Au contraire, le compilateur doit garantir que ces accès ne sont pas optimisés. Chaque opération doit avoir une marque correspondante dans le code produit. Les systèmes embarqués comptent sur cet attribut pour l'accès d'E/S reliés à la mémoire, la communication entre plusieurs fils d'exécution ou les interceptions d'interruptions matérielles et le programme.

Quant aux auteurs Yang et *al*, leur article [1] propose une méthode nommée « test flou » pour assurer une vérification des compilateurs de langage C. La technique consiste à injecter au hasard des blocs de code source valides, le compile avec au moins trois compilateurs et exécute tour à tour les programmes. Les compilateurs à l'origine des programmes qui produisent des résultats différents de la majorité sont réputés problématiques.

L'équipe construit l'outil nommé : « Csmith ». L'instrument génère, compile, exécute et compare les résultats obtenus avec le même programme produit de trois compilateurs différents et détecte les failles comme illustrées dans la Figure 10 :

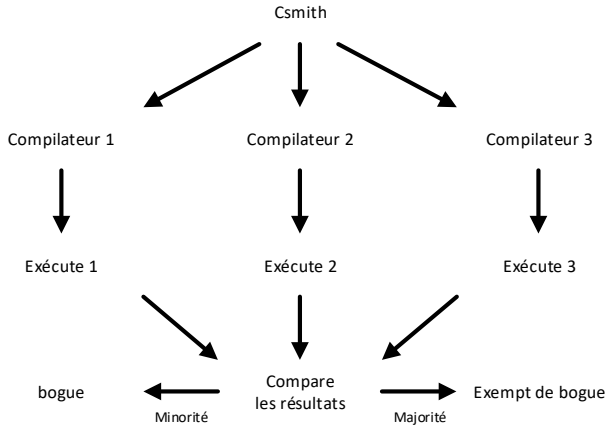


Figure 10 Technique dite « randomized differential testing » [1, Fig. 2]

L'approche « test flou » a permis de répertorier 325 bogues sur une période de trois ans. Les compilateurs testés cessent tout bonnement de fonctionner ou produisent un code exécutable erroné à partir d'une source valide.

Anecdote intéressante, l'outil Csmith découvre six bogues de génération de code avec CompCert [42], [53]. Les défectuosités se concentraient toutes dans la partie non vérifiée des étapes de compilation frontales, mais aucune de ses étapes de productions dorsales ne produisent les bogues trouvés parmi les autres compilateurs. La communauté livre une version exempte de bogues au début de l'année 2011, encore intacte malgré près de six années-UCT de tests par « Csmith ».

2.7 Mesures de défense contre les vulnérabilités des compilateurs

Dans son article [42], Xavier Leroy fait état du projet CompCert, un compilateur C-Light (une version réduite du langage C) vérifié par des preuves formelles à chaque phases de traduction. Les preuves formelles ont été réalisées en langage Coq, un langage combinant la logique de premier ordre avec de la programmation fonctionnelle. L'auteur décrit la notion de « prévention sémantique » qui consiste à prouver que la sémantique du code source est préservée tout au long du processus de traduction. CompCert est décomposé en 14 phases et huit langages intermédiaires. Le code intermédiaire est analysé à la sortie de chaque phase par un service de validation composé d'un invariant défini en langage Coq. Le service de validation peut augmenter le niveau de confiance, mais peut lui-même renfermer des anomalies. Il doit produire un certificat susceptible d'être vérifié formellement par l'utilisateur ou, à défaut, produire une erreur.

L'équipe autour de Jianzhou [54], proposent une stratégie de vérification formelle pour l'infrastructure de compilation LLVM. La technique consiste à insérer des appels aux services de vérification dans certains fils de génération SSA (« *single static assignment* ») qui servent dans plusieurs stratégies d'optimisation. Un peu comme le propose Leroy dans son article [42], chaque service implémente des preuves formelles sous la forme d'invariants réalisés en langage Coq. Le générateur conserve les optimisations vérifiées et neutralise les autres.

Les auteurs Carré et Garnsworthy [55] présentent ADA comme un des langages conçus pour le développement de systèmes critiques. Le sous-ensemble SPARK renferme des types et fonctions de sécurité ainsi qu'un langage d'annotations qui assure la prédictibilité par vérification formelle des programmes ADA. Dernièrement, l'article de Courtieu, Aponte et *al* [56] décrit plus en détail le fonctionnement des annotations permettant l'ajout de prédicats et leur interaction avec le programme à l'exécution. Le programme peut s'auto vérifier à l'exécution.

Croker [50] s'engage dans une comparaison, entre un compilateur C et C++ conçu pour la production de logiciels vérifiés par méthodes formelles, et l'offre SPARK intégrée au langage ADA. Contrairement aux techniques présentées dans les articles [42], [54], Croker étudie un dérivé de C++ affermi par des annotations pour exercer la vérification sémantique à la manière du paradigme « Verified Design-by-contract » [57]. L'expérimentation porte sur le compilateur MISRA-C : 2012, enrichi avec des règles qui prohibent l'usage des constructions du langage C plus vulnérables [58], ainsi que d'une notation de règles contractuelles, semblable à celles offertes avec SPARK. Un analyseur de théorèmes vérifie la cohérence des contrats. Le code généré peut se valider constamment en cours d'exécution. Ces règles deviennent inactives pour assurer la compatibilité avec tout compilateur C standard.

2.8 La tendance de l'industrie

Les constats fait dans l'article [21] de Gartner représentent l'aube d'une nouvelle ère de l'IdO à laquelle l'industrie de l'automatisation industrielle n'échappe pas. Comme le décrivent Goldschmidt, Murugaiah Sonntag et *al* dans leur article [20], les temps ne sont plus propices au déploiement systématique de PLC matériels, coûteux et encombrants, mais favorables à l'évolutivité élastique et à la virtualisation. L'avenir appartient aux réseaux de capteurs et d'actuateurs interconnectés et recensés aux PLC virtuels par les moyens de déploiements de type IdO [20], [48]. Gartner prévoit les interconnections de 25 milliards d'IdO d'ici à 2020 dont un quart de milliard de véhicules [21]. D'aussi grands nombres permettent de supposer l'existence d'un relatif potentiel de dangerosité.

Dans leur article [38], Wei, Xiaohong et Tingdi expliquent l'impact de la diversité logicielle sur les systèmes critiques. Le standard CEI 61508 présente la diversité logicielle pour assurer l'intégrité d'un système[12, Chap. C.3.1]. Les trois auteurs appliquent ce principe au compilateur lui-même et s'appuient sur la diversité pour détecter les erreurs injectées en phase de compilation.

Les chapitres suivants tentent de déterminer si la diversification logicielle appliquée aux compilateurs permet de rehausser au niveau SC4, une machine virtuelle de type *Soft PLC* présentement homologuée SC3.

Chapitre 3

Problématique

L'essai tente de déterminer si la diversité logicielle appliquée au compilateur peut se substituer aux techniques de vérification formelles ou de traçabilité ascendante, pour rehausser l'indice de capabilité systématique SC d'une machine virtuelle écrite en langage C89 ANSI. La norme présente toute machine virtuelle comme un composant autonome. Chaque composant d'un système destiné à une fonction de sécurité doit se conformer à un processus d'homologation pour obtenir son indice SC. La mesure de chaque composant se combine pour former l'indice SIL qui reflète l'échelle de sécurité fonctionnelle du système.

3.1 Description

Le chapitre 2.2 présente les systèmes électroniques programmables comme des composants physiques à l'origine [18], que le génie logiciel permet de virtualiser sous la forme de *Soft PLC* [20], [48].

La plupart des fabricants qui proposent des PLC virtuels les produisent en langage C ou C++ [15], [16], [50], mais les compilateurs comportent leurs vulnérabilités [1], [52], [59]. La diversité s'appuie sur l'hypothèse qu'ils sont produits par des groupes distincts, n'injectent pas les erreurs aux mêmes endroits dans le code exécutable. Dès lors, employer des compilateurs différents pour deux instances d'application en redondance représente un mode de défense valable et économique contre les CCF.

La norme CEI 61503 fait référence aux techniques formelles de vérification [12, Chap. B.2.2], mais la documentation propose peu d'exemples concrets. Les documents de techniques et mesures décrivent également la démonstration d'une vérification par analyse de traçabilité ascendante. L'exemple compare le code source avec le programme binaire d'une puce PROM [47], mais l'expérimentation date des années quatre-vingt-dix et s'applique sur un langage généraliste déclassé.

Le fardeau de la preuve d'intégrité logicielle repose sur les groupes de conception [8, Chap. 8]. La norme CEI 61508 propose un cycle de vie logiciel et des exigences, telles que la traçabilité entre les phases de conception (gestion d'exigences, d'architecture, de spécifications, d'inspections, d'essais, et de simulations) pour déterminer l'indice SC de chaque composant [8, Chap. 7]. Les indices SC se combinent pour calculer l'indice SIL du système [2]. La sélection ainsi que la diversité des compilateurs bonifient la capabilité systématique pour la porter au degré suivant et augmenter la valeur du composant en contrepartie d'un effort minimal.

Parallèlement, l'avènement de l'IdO soulève de nouveaux enjeux [21]. Des systèmes électroniques programmables inondent le milieu familial en vagues déferlantes d'une envergure inégalée. Les groupes de conception logicielle subissent la pression des affaires pour raccourcir les délais de mise en service. Les situations dangereuses pour la vie, l'environnement, les acquis économiques et le matériel se multiplient à une vitesse exponentielle. La société risque à tout moment la stupeur d'une crise, devant laquelle la société doit légiférer en s'appuyant sur des normes de sécurité fonctionnelle comme le standard CEI 61508. La diversité des compilateurs représente un pare-feu déterminant pour le contrôle et l'atténuation du danger.

3.1.1 Objectifs et hypothèses

L'illustration ci-dessous représente les variables évaluées dans cet essai. D'abord, déterminer si la diversité de compilateurs permet d'augmenter l'indice SC. En une seconde étape, dans l'éventualité où la diversité à elle seule ne suffit pas, l'évaluation est reprise avec un compilateur vérifié par une méthode formelle.

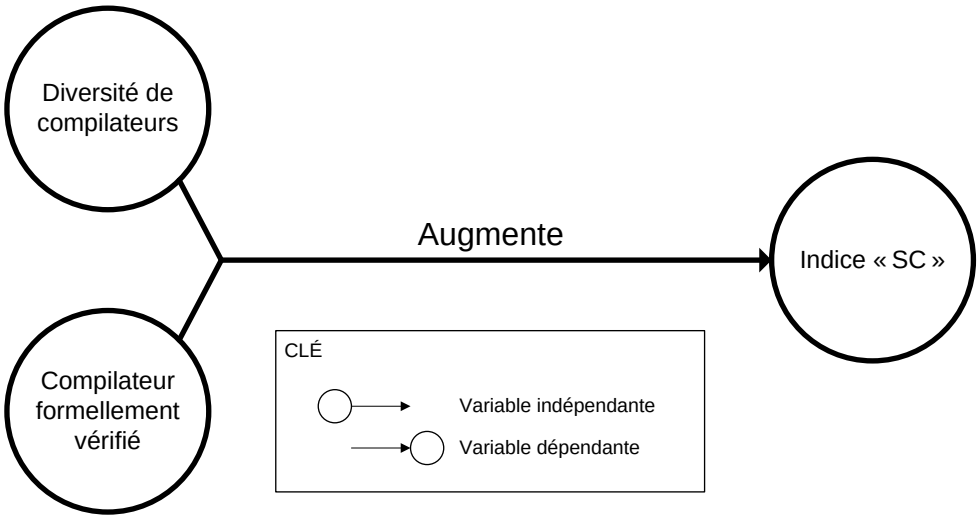


Figure 11 Question de recherche

Est-ce que le choix d'une combinaison de compilateurs, dont un qui est vérifié par des méthodes formelles, produisant une machine virtuelle écrite en langage C ANSI et homologuée « SC3 », rehausse l'indice de capabilité systématique (SC) de telle sorte qu'elle soit homologuée « SC4 » ?

L'hypothèse est négative. Le choix d'une combinaison de compilateurs rehausse le niveau de confiance, mais pas suffisamment pour bonifier l'indice de capabilité systématique en présence d'une architecture de redondance. La combinaison en elle-même produit une diversité logicielle, parce que chaque compilateur provient d'une équipe distincte, tout en se référant au même standard, et tous ces compilateurs produisent des programmes équivalents. L'assurance est toutefois accrue lorsqu'en prime, le choix s'arrête sur des compilateurs homologués pour la sécurité.

3.1.2 Limites

L'étude se borne à une machine virtuelle de type *Soft PLC*, écrite en langage C ANSI et homologuée « SC3 » pour mettre en œuvre une architecture redondante. La diversité logicielle est une conséquence de la redondance, et l'occasion de faire cohabiter un compilateur standard avec un équivalent homologué pour sa sécurité.

La vérification des conclusions devrait recourir à une véritable homologation de sécurité. Les coûts et l'envergure d'une telle démarche rend cette alternative hors de portée. La triangulation s'appuie sur le sondage d'une centaine de concepteurs logiciels certifiés par un organisme d'homologation en sécurité fonctionnelle.

L'essai ne s'étend pas sur toutes les combinaisons de *Soft PLC* avec ou sans logiciels embarqués hors du contexte d'une machine virtuelle. Le chapitre 2.8 évoque le déploiement de type infonuagique pour aider la compréhension du potentiel de dangerosité que l'IdO représente, mais le devis de recherche ne renferme pas d'information précise sur le sujet.

3.2 Méthodologie proposée

L'essai emprunte la structure d'une étude quantitative, descriptive, corrélationnelle. La démarche consiste à décrire des attributs de qualité et déterminer leur poids lors d'un exercice d'homologation de sécurité fonctionnelle. L'approche conduit à examiner l'indice SC d'un même composant en fonction de différents agencements d'attributs de qualité en vue de prévoir l'apport de nouveaux attributs sur l'élévation de sa cote de sécurité.

3.3 Mise en œuvre

L'objet de l'essai consiste à déterminer si la diversité de compilateurs influence l'indice SC d'une machine virtuelle de type *Soft PLC*. La revue de littérature présente différents modes de calcul de la capabilité systématique et techniques de vérification de compilateurs suggérées dans la documentation du standard CEI 61508. Aucune technique de vérification des compilateurs ne se démarque vraiment, ou sinon au prix des méthodes formelles. Quant à l'hypothèse que la diversité de compilateurs augmente la capabilité sémantique, la vérification se morcèle en huit étapes énumérées comme suit :

1. Décrire sommairement l'architecture d'une machine virtuelle homologuée « SC3 » ;
2. Calculer les indices PFD et PFH de la machine virtuelle homologuée « SC3 » présentée au premier point ;

3. Calculer les PFD et PFH de la machine virtuelle homologuée « SC3 » présentée au premier point, mais produits avec une diversité de compilateurs comme mesures défensives contre les CCF ;
4. Préciser quelques nouveaux attributs de qualités susceptibles de bonifier une homologation « SC3 »;
5. Rédiger un questionnaire pour valider les conclusions de l'étude auprès d'au moins cent concepteurs logiciels certifiés en sécurité fonctionnelle ;
6. Faire un appel de sondage auprès des groupes LinkedIn® « IEC 61,508 on Functional Safety », « Safety Integrity Level (SIL) » et « Safety Automation Forum » pour une période d'un mois, de la mi-mai à la mi-juin 2017 ;
7. Collecter les réponses aux sondages, compiler et analyser les résultats ;
8. Conclure.

Chapitre 4

Approche proposée

La présente étude applique aux compilateurs le principe de la diversité logicielle afin de rehausser les probabilités de détection d'erreurs qu'ils sont susceptibles d'injecter. La diversification permet d'élever le niveau de capabilité systématique (SC) d'un composant homologué en fonction de la norme CEI 61508.

Les fabricants d'automatismes industriels critiques s'interdisent l'emploi des fonctions avancées offertes par les compilateurs en vue de réduire les possibilités d'erreurs et les risques qui en découlent pour l'intégrité de leur application. Lorsqu'un système implique la redondance, une diversité de compilateurs permet d'exploiter ces fonctions tout en assurant une détection des anomalies injectées par ceux-ci.

4.1 Échantillon

L'étude fait appel aux professionnels et experts certifiés en sécurité fonctionnelle. Ces spécialistes doivent avoir reçu une formation dédiée à la sécurité fonctionnelle et avoir réussi un examen portant sur leurs connaissances du standard.

Les professionnels (CSFP) font partie du personnel formé en sécurité fonctionnelle et affectés à la conception des systèmes critiques. Ces derniers doivent compter en nombre significatif parmi les équipes de conception.

La formation des experts (CSFE) est analogue à celle des professionnels, et enrichie par une couverture détaillée de la gestion de projets, la gestion des exigences ainsi que la préparation des devis d'homologation. Ces professionnels font office de donneurs d'ordre au sein des organisations qui offrent des produits et services pour systèmes critiques. Ils assument très souvent les tâches reliées à la production et la présentation du devis d'homologation en sécurité fonctionnelle.

La population ciblée se décompose comme suit :

Tableau 9 Population visée par l'étude

Type de population	Nombre estimé	Accessible	Échantillon
Professionnels (CFSP)	8 737	3 408	150
Experts (CFSE)	822	6	2

La validation de l'étude repose sur la révision des calculs du CCF pour corroborer la bonification de la cote SC. Parmi les 822 experts inscrits au registre CFSE, nous comptons six Canadiens, dont deux pour valider les calculs et l'effet sur la l'indice SC. Les retours des deux experts vont permettre l'élaboration d'un questionnaire pour la triangulation auprès de professionnels inscrits au registre du CSFP.

En fonction des 8 737 professionnels inscrits au registre CFSP et des 822 experts au CFSE, nous comptons 3 408 membres des groupes LinkedIn® « IEC 61508 on Functional Safety », « Safety Integrity Level (SIL) » et « Safety Automation Forum ». Les membres ne sont pas nécessairement des professionnels ou experts inscrits. Leurs populations se confondent et la précision manque, d'où l'importance du nombre. La triangulation des résultats repose sur la réponse de 150 professionnels et deux experts sondés sur la pertinence des calculs.

4.2 Description de l'approche

L'approche consiste à refaire le calcul de probabilité de défaillance, avec et sans mesure défensive contre les CCF, et valider avec la population visée.

4.2.1 Facteurs clés de succès

Les indices PFD [11, Chap. B.3.2.3] et PFH [11, Chap. B.3.3.3] utilisés pour soutenir l'hypothèse sont tirés des cahiers de la norme CEI 61508. L'impact de la diversité de compilateurs se mesure par l'indice CCF (« β ») sur les architecture « 1oo2 », appliqué aux taux de défaillances d'une structure « 1oo1 » se qualifiant « SIL2 » avec un DC de 0% ou 60%.

Le sujet visé par l'étude consiste en une machine virtuelle de type (« SoftPLC »). Le composant est doté d'une certification « SC2 » lorsqu'il est déployé dans une architecture « 1oo1 », et

« SC3 » en redondance « 1oo2 ». Cette homologation le rend susceptible de répondre aux exigences d'un système « SIL2 » ou « SIL3 », en fonction du type de conception.

La cote « SC3 » du sujet découle de son adéquation avec une architecture de redondance « 1oo2 ». Une telle structure favorise la diversité de compilateurs et même de plateformes, lorsque les PLC logiciels se déploient sur des circuits distincts. L'hypothèse implique un système programmable homologué SIL2 avec une structure « 1oo1 » et SIL3 en architecture « 1oo2 », qui peut s'élever à « SIL4 » avec un logiciel produit d'une diversité de compilateurs pour contre-mesure aux CCF. Par voie de conséquence, à titre de composant logiciel, la machine virtuelle « SC3 » s'élève à « SC4 » avec une diversité de compilateurs.

Il existe une problématique sur la valeur qu'accorde la communauté à une diversité de compilateurs comme équivalent de diversité logicielle. La question posée consiste à déterminer la valeur attribuée à la diversité de compilateurs comme moyen de défense contre les défaillances de cause commune (« CCF »). La qualité du questionnaire et des exemples sont susceptibles d'influencer l'opinion des experts et professionnels consultés.

Un second point délicat implique une exigence d'anonymat du fabriquant du « SoftPLC » choisi pour modèle d'expérimentation. L'étude doit décrire les calculs de probabilité de défaillance sans évoquer le produit ou la marque.

4.2.2 Approche de validation des résultats

La démarche de calcul du PFD et du PFH se valide par dix questions de sondage en vue de confirmer l'apport de la diversité de compilateurs sur la présentation de CCF. La diversité logicielle apparaît comme attribut déterminant avec le standard CEI 61508 [51, Chap. 7.4.3.2 (a)][51, Sect. Table C.2 (3.b, 3.c)]. Puisque la diversité s'applique aux compilateurs, la démarche de sélection de l'outil ajoute davantage de poids [38], appliquant un effet de levier au bénéfice de la cote SC du composant.

Un questionnaire est élaboré pour obtenir l'aval de la communauté (voir 4.1) à propos de l'effet de la diversité de compilateurs sur l'élévation de la cote SC. Le questionnaire renferme une description des variables impliquées dans le calcul des indices PFD et PFH, les attributs de qualité affectés, leur modulation en fonction du choix des outils.

4.2.2.1 Sondage de validation

Une des questions du sondage permet de scinder les résultats en fonction du degré de certification des répondants. Un premier tableau distribue les compteurs pour chaque question :

Tableau 10 Compilation des réponses au sondage

Q	Question	CFSE	CFSP	Non-homologué
1	Nombre total de répondants			
2	Attester que les tests par classes d'équivalences appliquées au logiciel se substituent au « FIT » (« *Fault Injection Test* ») matériel pour le calcul de la proportion de défaillances en sécurité (« SFF ») ?			
3	Attester qu'une couverture de test sur un logiciel équivaut à l'indice DC pour le matériel ?			
3	Attester qu'un *Soft PLC* avec une couverture de test supérieure à 95 % permet d'attribuer un DC de 99 % [11, Sect. Tableau B.13] ?			
3	Attester qu'un *Soft PLC* avec une couverture de test supérieure à 95 % équivaut à un DC de 90 % [11, Sect. Tableau B.13] ?			
3	Attester qu'un *Soft PLC* avec une couverture de test supérieure à 95 % équivaut à un DC de 60 % [11, Sect. Tableau B.13] ?			
3	Attester qu'un *Soft PLC* avec une couverture de test supérieure à 95 % équivaut à un DC de 0 % [11, Sect. Tableau B.13] ?			
4	Attester qu'un *Soft PLC* intégré à système d'architecture « 1oo2 », mais produit par un seul compilateur est vulnérable aux défaillances de cause communes (« CCF ») ?			
5	Attester qu'un *Soft PLC* intégré à système d'architecture « 1oo2 » et produit par un seul compilateur équivaut un β (« CCF ») de 20 % [11, Sect. Tableau B.13] ?			
5	Attester qu'un *Soft PLC* intégré à système d'architecture « 1oo2 » et produit par un seul compilateur équivaut un β (« CCF ») supérieur à 20 % [11, Sect. Tableau B.13] ?			
6	Attester qu'un *Soft PLC* intégré à système d'architecture « 1oo2 » produit par deux compilateurs est protégé contre les défaillances de cause communes (« CCF ») ?			
7	Attester qu'un *Soft PLC* intégré à système d'architecture « 1oo2 » produit par deux compilateurs équivaut un β (« CCF ») de 20 % [11, Sect. Tableau B.13] ?			

Q	Question	CFSE	CFSP	Non-homologué
7	Attester qu'un *Soft PLC* intégré à système d'architecture « 1oo2 » produit par deux compilateurs équivaut un β (« CCF ») de 10 % [11, Sect. Tableau B.13] ?			
7	Attester qu'un *Soft PLC* intégré à système d'architecture « 1oo2 » produit par deux compilateurs équivaut un β (« CCF ») de 2 % [11, Sect. Tableau B.13] ?			
8	Attester de techniques de surveillance diversifiées [51, Sect. Tebleau A.2, point 3.c] ?			
8	Attester que la redondance diversifiée sur le plan fonctionnel implémente une spécification des exigences de sécurité du logiciel [51, Sect. Tebleau A.2, point 3.e] ?			
9	Attester de la sélection de MISRA-C pour utilisation de composants logiciels sécurisés/vérifiés [51, Sect. Tableau A.2, point 8] ?			
9	Attester de la sélection de MISRA-C pour utilisation de composants logiciels sécurisés/vérifiés [51, Sect. Tableau A.4, point 7] ?			
10	Attester qu'un *Soft PLC* intégré à système d'architecture « 1oo2 » produit par deux compilateurs dont un est certifié (MISRA-C) équivaut un β (« CCF ») de 20 % [11, Sect. Tableau B.13] ?			
10	Attester qu'un *Soft PLC* intégré à système d'architecture « 1oo2 » produit par deux compilateurs dont un est certifié (MISRA-C) équivaut un β (« CCF ») de 10 % [11, Sect. Tableau B.13] ?			
10	Attester qu'un *Soft PLC* intégré à système d'architecture « 1oo2 » produit par deux compilateurs dont un est certifié (MISRA-C) équivaut un β (« CCF ») de 2 % [11, Sect. Tableau B.13] ?			

Le questionnaire retient les attributs de qualité qui ajoutent de la capabilité systématique à la machine virtuelle homologuée « SC3 » et changent le calcul de probabilité de défaillance. L'augmentation du nombre d'attributs de qualité et le résultat du calcul permettent de confirmer ou infirmer l'élévation à l'indice « SC4 » et prouvent ou invalident la valeur de la diversité de compilateurs.

4.3 Mise en œuvre

L'exécution consiste à étudier comment la diminution du CCF (« β ») de 20 % à 10 % se répercute sur les indices PFD [11, Chap. B.3.2.3] et PFH [11, Chap. B.3.3.3] issus des résultats de calculs offerts dans les cahiers de la norme. L'étude s'attarde aux coefficients de défaillances dangereuses (« λ_D »), se qualifiant SIL3 pour un CCF (« β ») de 20 % avec une structure « 1oo2 », et SIL2 avec une structure « 1oo1 ». L'analyse porte exclusivement sur les indices PFD et PFH calculés en fonction d'une couverture de diagnostic (DC) nulle (0 %), pour correspondre à la situation de la machine virtuelle. L'indice SC correspond au SIL et permet de déduire l'élévation proportionnelle de l'indice SC de SC3 à SC4.

L'étude se focalise sur les tableaux d'indices PFD et PFH présentés dans « Annexe II ». Les cahiers de la norme renferment des tableaux, calculés en fonction de six coefficients de défaillances (« λ »), six architectures et huit modes d'entretien.

4.3.1 Calculs du PFDavg et du PFHavg

Les indices PFDavg et PFHavg représentent des valeurs analogues. La première exprime la probabilité moyenne de défaillance en mode faible sollicitation, soit une exécution de la fonction de sécurité par an. La seconde symbolise l'équivalent en mode haute sollicitation on en exécution continue. L'illustration suivante se présente par une courbe en dent de scie utile pour préciser l'impact des intervalles d'essais sur la probabilité moyenne de défaillance [11, Chap. B.5.2.1].

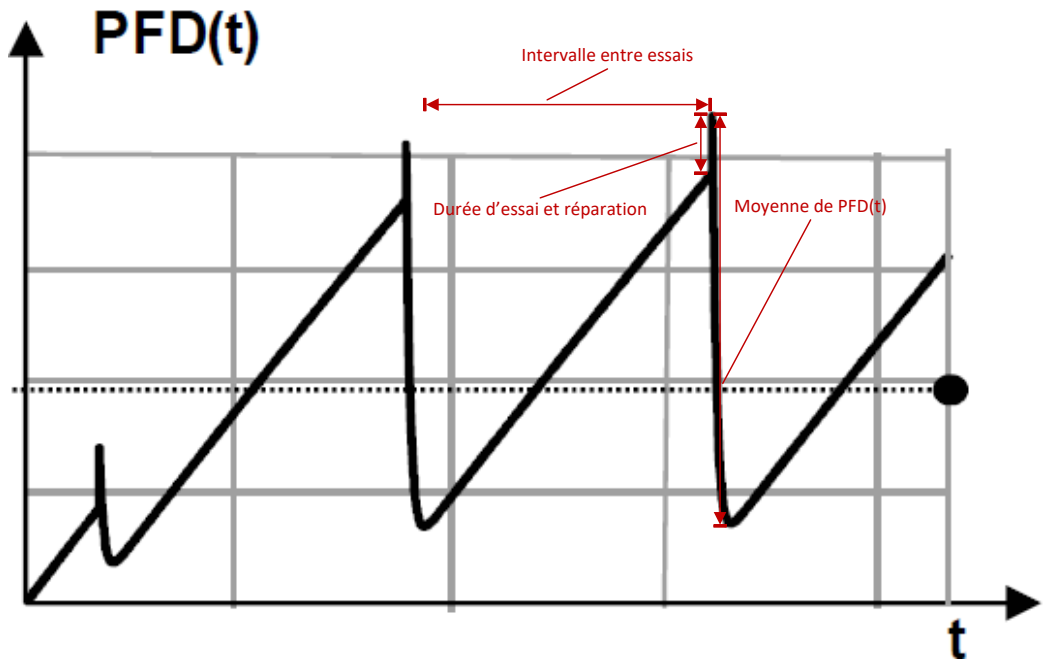


Figure 12 Courbe en dent de scie des probabilités de défaillances, en fonction des intervalles d'essais

La moyenne de PFD (t) produit l'indice PFDavg que la Figure 12 illustre par le pointillé. Le niveau SIL du système correspond à l'intervalle à l'intérieur duquel se situe cette moyenne. Les formules prennent en compte, la probabilité de défaillance entre les essais, ainsi que la période supplémentaire associée au moment où l'équipement se déconnecte pour subir les tests et l'entretien.

4.3.2 Effet de la diversité de compilateurs sur le PFD

Les graphiques facilitent la sélection du coefficient de défaillance et l'observation d'une diminution de CCF (« β ») de 20% à 10%. Un système au PFDavg situé à l'intérieur de l'intervalle SIL2 avec une architecture « 1oo1 » et SIL3 en structure « 1oo2 » avec CCF (« β ») de 20%, doit franchir l'intervalle SIL4 lorsque le CCF (« β ») se réduit à 10%.

4.3.2.1 Intervalle entre essais de six mois et MTTR de huit heures

Le graphique ci-dessous présente les probabilités de défaillance exposées à l'annexe II.1.1. Dans l'hypothèse que la diversité de compilateurs autorise la réduction de l'indice CCF (« β ») à 10 %, les colonnes permettent de constater que le PFDavg reste dans l'intervalle SIL3.

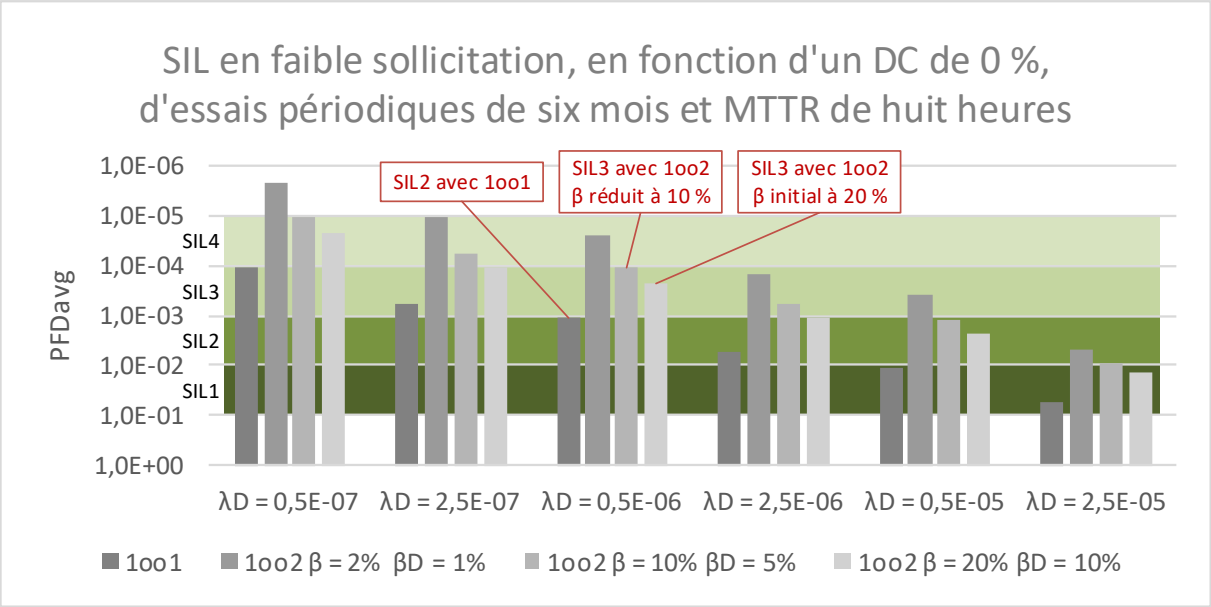


Figure 13 SIL en faible sollicitation, en fonction d'un DC de 0 %, d'essais périodiques de six mois et MTTR de huit heures

La Figure 13 met en relief le coefficient d'anomalie (« λ_D ») de 0,5 E-06, ce qui représente un MTTF de 2 millions d'heures. Il s'agit du seul coefficient pour lequel le PFDavg se situe dans l'intervalle SIL2 pour une architecture « 1oo1 » et SIL3 pour une architecture « 1oo2 » avec un CCF (« β ») de 20 %. La réduction du CCF (« β ») à 10 %, qui caractérise la valeur d'une diversité de compilateurs, reste toutefois dans la borne supérieure de l'intervalle SIL3 ; elle n'atteint pas l'intervalle SIL4, même si elle s'en approche.

4.3.2.2 Intervalle entre essais d'un an et MTTR de huit heures

Le graphique ci-dessous présente les probabilités de défaillance exposées à l'annexe II.1.2. Dans l'hypothèse que la diversité de compilateurs autorise la réduction de l'indice CCF (« β »)

à 10 %, comme à la section 4.3.2.1, les colonnes permettent de constater que le PFDavg reste dans l'intervalle SIL3.

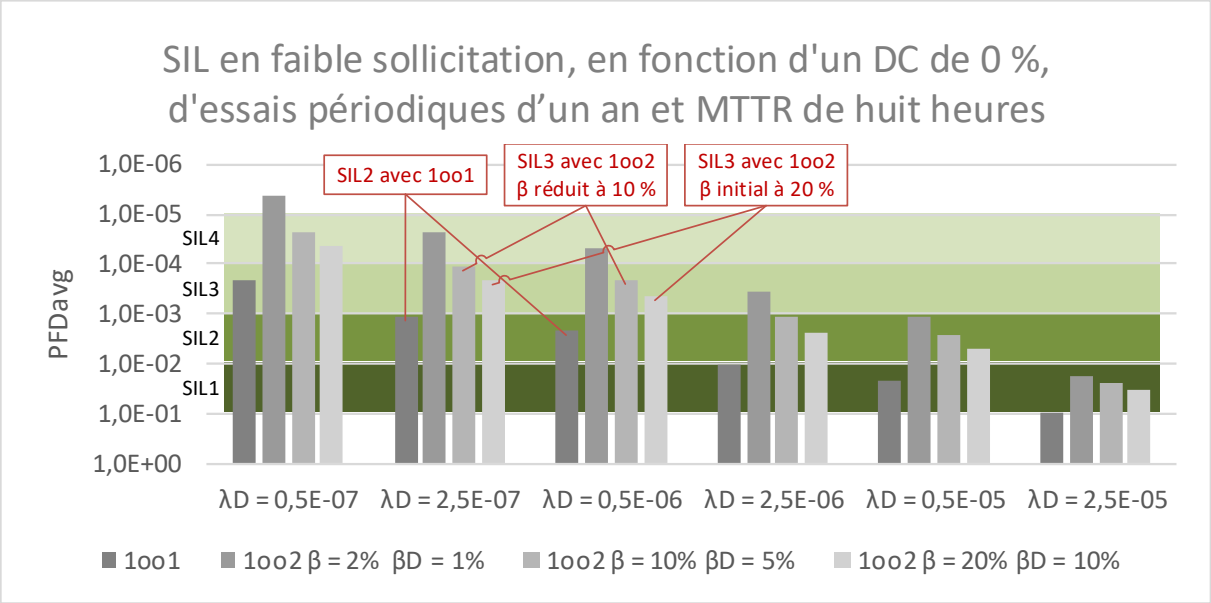


Figure 14 SIL en faible sollicitation, en fonction d'un DC de 0 %, d'essais périodiques d'un an et MTTR de huit heures

La Figure 14 met en relief les deux coefficients d'anomalie (« λ_D ») de 0,5 E-06 et de 2,5 E-06, qui représentent des MTTF de deux millions et quatre millions d'heures. Il s'agit des seuls coefficients pour lesquels le PFDavg se situe dans l'intervalle SIL2 pour une architecture « 1oo1 » et SIL3 pour une architecture « 1oo2 » avec un CCF (« β ») de 20 %. La réduction du CCF (« β ») à 10 %, qui caractérise la valeur d'une diversité de compilateurs, reste toutefois dans la borne supérieure de l'intervalle SIL3 avec les deux coefficients ; ils n'atteignent pas l'intervalle SIL4, même lorsque λ_D = 2,5 E-07 s'en approche.

4.3.2.3 Intervalle entre essais de deux ans et MTTR de huit heures

Le graphique ci-dessous présente les probabilités de défaillance exposées à l'annexe II.1.3. Dans l'hypothèse que la diversité de compilateurs autorise la réduction de l'indice CCF (« β »)

à 10 %, comme à la section 4.3.2.1, les colonnes permettent de constater que le PFDavg reste dans l'intervalle SIL3.

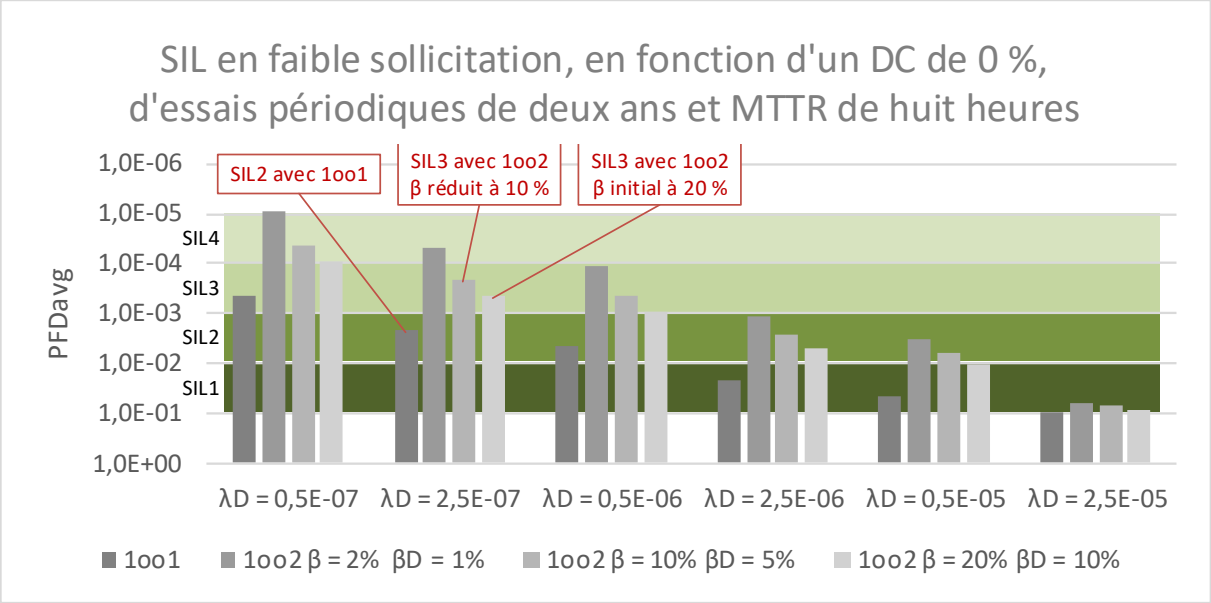


Figure 15 SIL en faible sollicitation, en fonction d'un DC de 0 %, d'essais périodiques de deux ans et MTTR de huit heures

La Figure 15 met en relief le coefficient d'anomalie (« λ_D ») de 2,5 E-07, ce qui représente un MTTF de quatre millions d'heures. Il s'agit du seul coefficient pour lequel le PFDavg se situe dans l'intervalle SIL2 pour une architecture « 1oo1 » et SIL3 pour une architecture « 1oo2 » avec un CCF (« β ») de 20 %. La réduction du CCF (« β ») à 10 % qui caractérise la valeur d'une diversité de compilateurs reste nettement dans l'intervalle SIL3 ; il n'atteint pas l'intervalle SIL4.

4.3.2.4 Intervalle entre essais de dix ans et MTTR de huit heures

Le graphique ci-dessous présente les probabilités de défaillance exposées à l'annexe II.1.4. Dans l'hypothèse que la diversité de compilateurs autorise la réduction de l'indice CCF (« β ») à 10 %, comme à la section 4.3.2.1, les colonnes permettent de constater que le PFDavg reste dans l'intervalle SIL3.

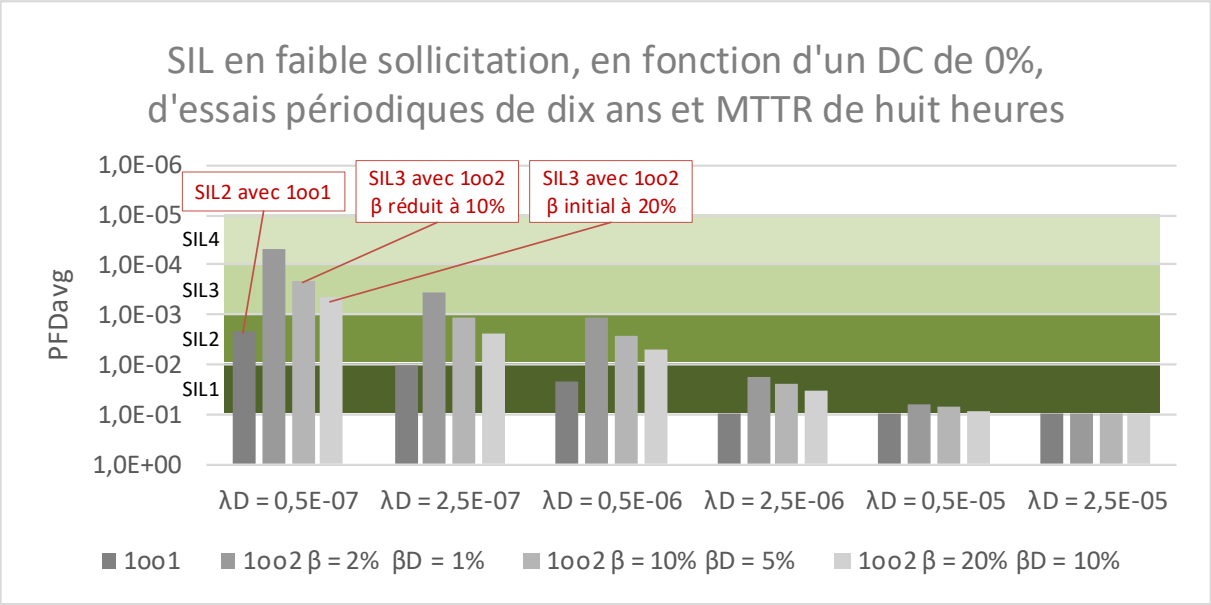


Figure 16 SIL en faible sollicitation, en fonction d'un DC de 0 %, d'essais périodiques de dix ans et MTTR de huit heures

La Figure 16 met en relief le coefficient d'anomalie (« λ_D ») de 0,5 E-07, ce qui représente un MTTF de 20 millions d'heures. Il s'agit du seul coefficient pour lequel le PFDavg se situe dans l'intervalle SIL2 pour une architecture « 1oo1 » et SIL3 pour une architecture « 1oo2 » avec un CCF (« β ») de 20 %. La réduction du CCF (« β ») à 10 %, qui caractérise la valeur d'une diversité de compilateurs, reste nettement dans l'intervalle SIL3 ; il n'atteint pas l'intervalle SIL4.

La portion suivante applique la même méthode d'analyse, avec le mode de sollicitation élevée ou continue.

4.3.3 Effet de la diversité de compilateurs sur le PFH

Le PFHavg reste analogue au PFDavg, mais s'applique aux systèmes qui fonctionnent en mode sollicitation élevée ou continue. De tels systèmes s'exécutent plus d'une fois par an ou au moins à deux reprises à l'intérieur d'une durée, entre une séquence d'essais périodiques [33]. Les intervalles entre les tests se présentent sur de plus courts espaces de temps.

4.3.3.1 Intervalle entre essais d'un mois et MTTR de huit heures

Le graphique ci-dessous présente les probabilités de défaillances exposées à l'annexe II.2.1. Dans l'hypothèse que la diversité de compilateurs autorise la réduction de l'indice CCF (« β ») à 10 %, les colonnes permettent de constater que le PFHavg reste dans l'intervalle SIL3.

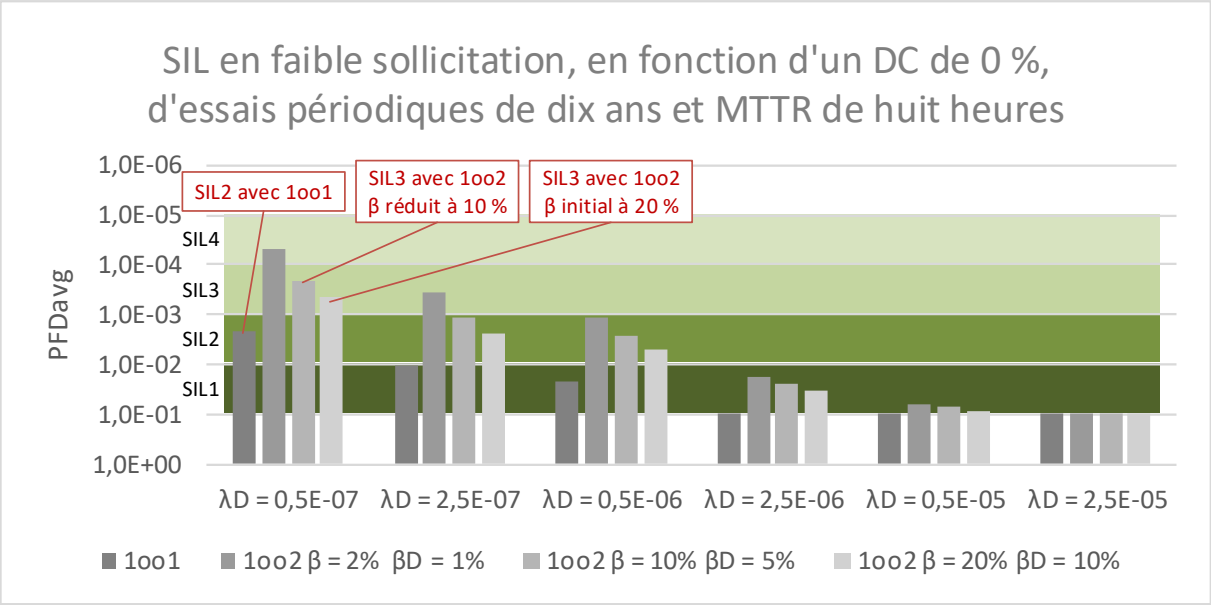


Figure 17 SIL en sollicitation élevée, en fonction d'un DC de 0 %, d'essais périodiques d'un mois et MTTR de huit heures

La Figure 17 met en relief le coefficient d'anomalie (« λ_D ») de 2,5 E-07, ce qui représente un MTTF de quatre millions d'heures. Il s'agit du seul coefficient pour lequel le PFHavg se situe dans l'intervalle SIL2 pour une architecture « 1oo1 » et SIL3 pour une architecture « 1oo2 » avec un CCF (« β ») de 20 %. La réduction du CCF (« β ») à 10 %, qui caractérise la valeur d'une diversité de compilateurs, reste toutefois dans l'intervalle SIL3 ; le PFHavg n'atteint pas l'intervalle SIL4.

4.3.3.2 Intervalle entre essais de trois mois et MTTR de huit heures

Le graphique ci-dessous présente les probabilités de défaillance exposées à l'annexe II.2.2. Dans l'hypothèse que la diversité de compilateurs autorise la réduction de l'indice CCF (« β ») à 10 %, comme à la section 4.3.3.1, les colonnes permettent de constater que le PFHavg reste dans l'intervalle SIL3.

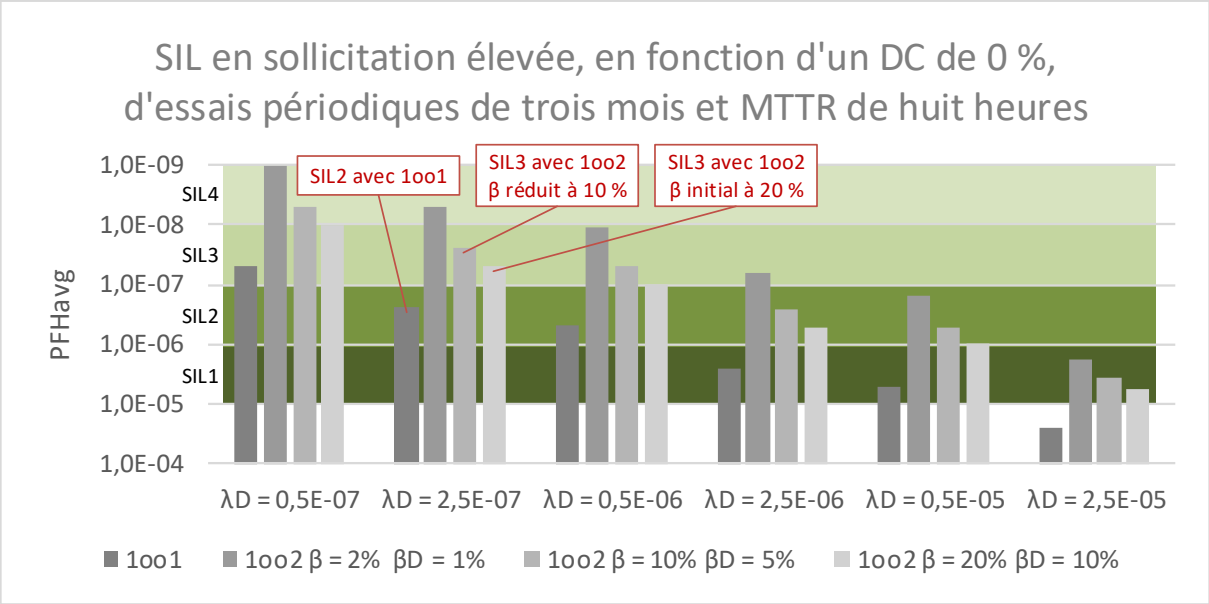


Figure 18 SIL en sollicitation élevée, en fonction d'un DC de 0 %, d'essais périodiques de trois mois et MTTR de huit heures

L'écart de PFDavg varie très peu entre des tests périodiques d'un (Figure 17) et trois mois (Figure 18). La réduction du CCF (« β ») de 20 % à 10 % maintient le PFDavg dans l'intervalle SIL3; il n'atteint pas l'échelon SIL4.

4.3.3.3 Intervalle entre essais de six mois et MTTR de huit heures

Le graphique ci-dessous présente les probabilités de défaillances exposées à l'annexe II.2.3. Les colonnes permettent de constater que le PFHavg reste dans l'intervalle SIL3 comme aux sections 4.3.3.1 et 4.3.2.2.

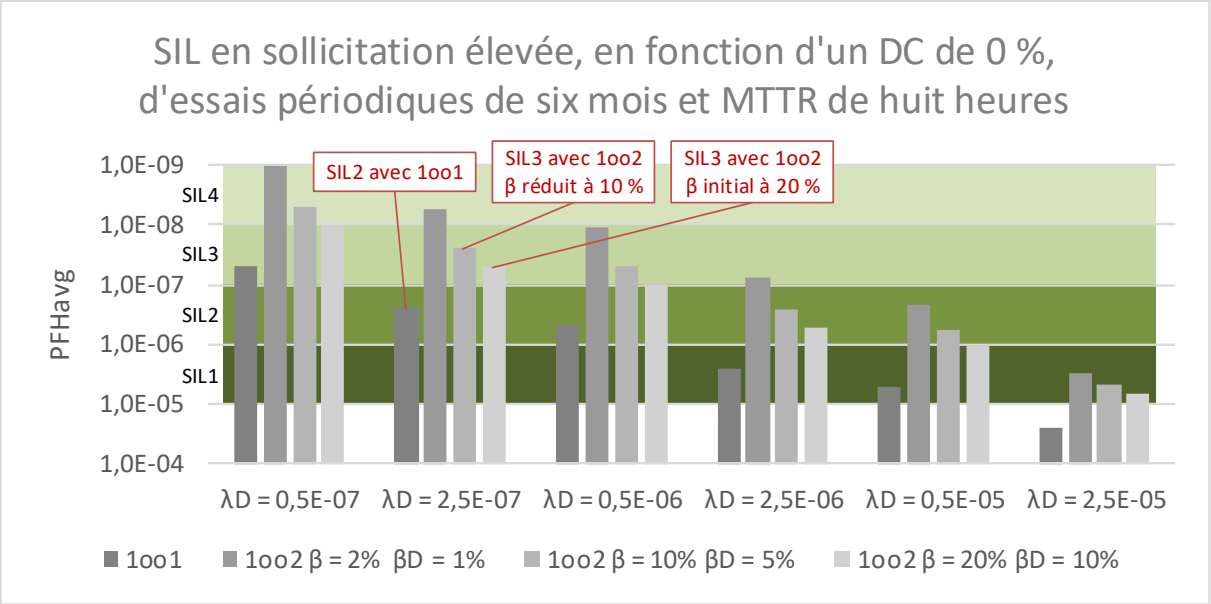


Figure 19 SIL en sollicitation élevée, en fonction d'un DC de 0 %, d'essais périodiques de six mois et MTTR de huit heures

L'écart de PFDavg varie très peu entre des tests périodiques d'un (Figure 17), trois (Figure 18) et six (Figure 19) mois. La réduction du CCF (« β ») de 20 % à 10 % maintient le PFDavg dans l'intervalle SIL3 ; il reste relativement éloigné de l'échelon SIL4.

4.3.3.4 Intervalle entre essais d'un an et MTTR de huit heures

Pour terminer l'étude comparative, le graphique ci-dessous présente les probabilités de défaillance exposées à l'annexe II.2.4. Les colonnes permettent de constater que le PFHavg reste dans l'intervalle SIL3, comme aux sections précédentes.

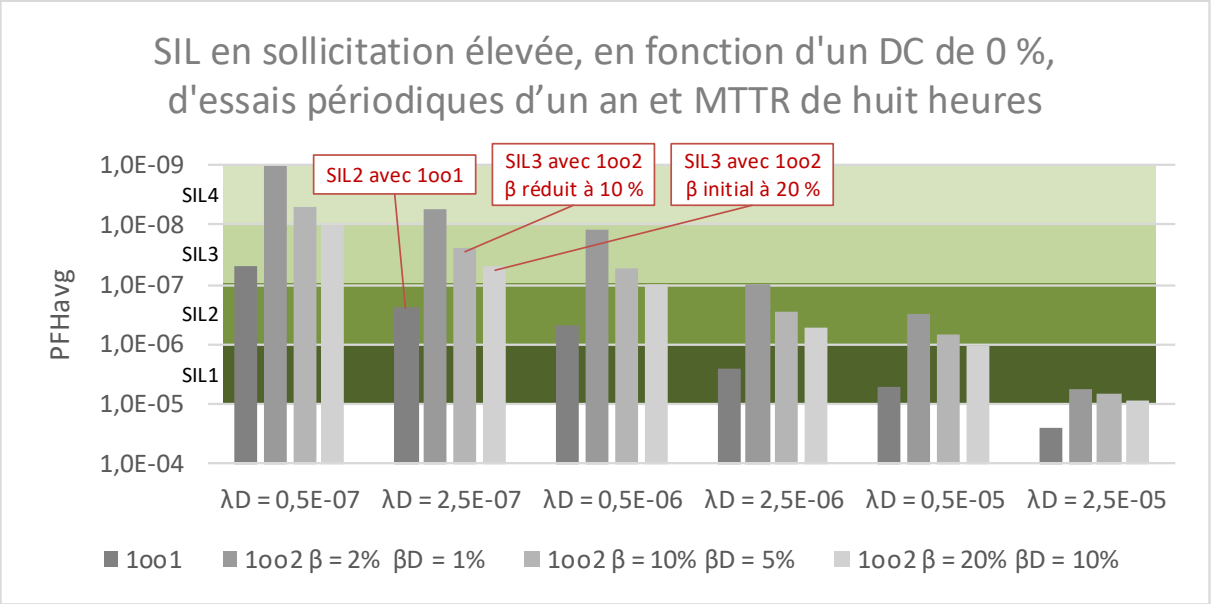


Figure 20 SIL en sollicitation élevée, en fonction d'un DC de 0 %, d'essais périodiques d'un an et MTTR de huit heures

L'écart de PFDavg varie très peu entre des tests périodiques précédents. La réduction du CCF (« β ») de 20 % à 10 % maintient toujours le PFDavg dans l'intervalle SIL3 ; il reste relativement éloigné de l'échelon SIL4.

Toutes les comparaisons permettent de tirer une conclusion sur la valeur d'une diversité de compilateurs pour une fonction conforme à l'architecture « 1oo2 ».

4.3.4 Analyse des calculs

Toutes les combinaisons de périodes d'essais et de fréquences d'activation demeurent dans l'intervalle SIL3 malgré l'emploi d'une diversité de compilateurs. La combinaison la plus inusitée se présente au paragraphe 4.3.2.2, où un coefficient d'anomalie (« λ_D ») de 2,5 E-07, qui représente un MTTF de quatre millions d'heures, s'approche de SIL4 sans l'atteindre. Toutes les autres combinaisons demeurent nettement au-dessous de ce seuil.

Les résultats permettent de conclure que le CCF doit réduire au-delà de 50 %, pour assurer un effet direct sur la mesure SC. La diversité de compilateurs ne représente pas une valeur suffisamment forte pour élever le composant de l'indice SC3 à SC4.

4.4 Résultats attendus

Les experts et professionnels certifiés pourrait se présenter avec plus de sévérité que la présente analyse et réfuter la réduction du CCF (« β ») de 20 % à 10 %, ou en accordant un coefficient supérieur à 20 %. Dans cette dernière éventualité, la diversité de compilateurs représente une valeur encore moindre.

Une faible proportion de l'échantillon pourrait au contraire consentir une réduction du CCF (« β ») de 20 % à 2 % avec le choix d'un compilateur certifié. Dans une telle éventualité, la diversité de compilateurs fait passer le cap SIL4 et pourrait représenter un retour sur l'investissement (*Return On Investment* ou ROI) disproportionné par rapport au coût.

Chapitre 5

Analyse des résultats

Le sondage révèle une polarisation de la communauté. La majorité des formulaires retenus témoignent d'une réserve devant le calcul d'un indice CCF pour les logiciels.

5.1 Résultats obtenus

Pour que la mesure soit considérée, un nombre minimal d'experts et professionnels certifiés doit la juger pertinente. Les réponses obtenues par sondage révèlent la réticence des acteurs du secteur de la sécurité fonctionnelle pour appliquer l'indice CCF aux logiciels.

Des 34 personnes sondées, 13 sont retenues pour déterminer si le groupe accepte d'attribuer un indice CCF à la diversité de compilateurs. À cet effet, la Figure 21 expose une nette polarisation de la communauté :

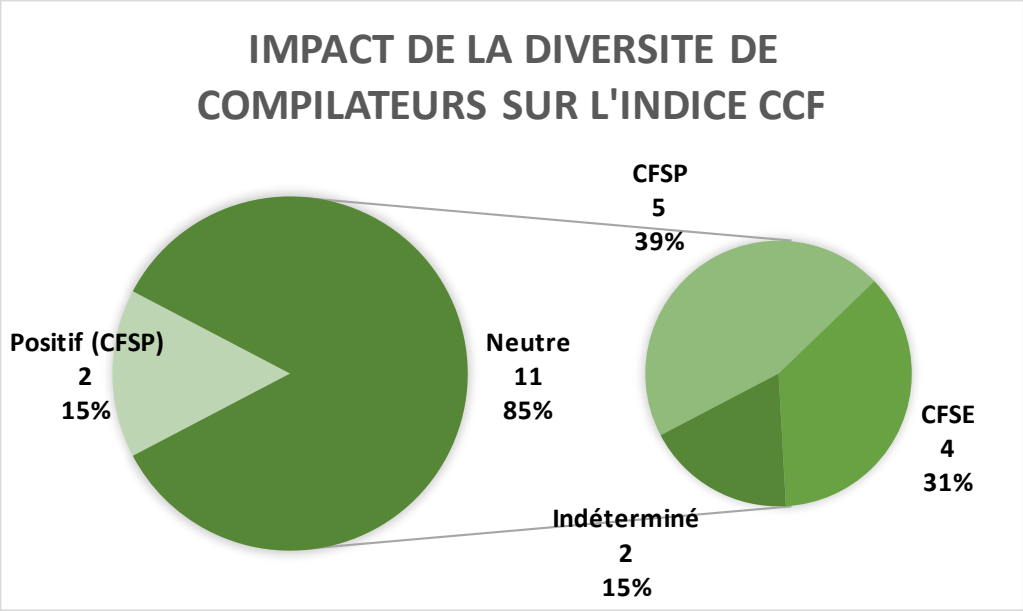


Figure 21 Impact de la diversité de compilateurs sur l'indice CCF

La Figure 21 scinde les 13 participants au sondage en deux diagrammes circulaires.

La partie gauche indique que seulement 15% (deux sur 13) des personnes interrogées, tous des professionnels homologués en sécurité fonctionnelle, jugent que la diversité de compilateurs affecte positivement (en réduisant) l'indice CCF. À l'inverse, 85% d'entre eux, soit la majorité des acteurs impliqués en sécurité fonctionnelle, restent sceptiques, avec ou sans compilateur homologué pour la sécurité fonctionnelle. Aucun des 13 participants n'y trouve d'impact négatif.

La partie droite distribue en fonction de leur niveau de compétence, 85% (11 sur 13) des personnes sondées qui jugent la diversité de compilateurs sans effet sur l'indice CCF. Elle permet de constater que les experts interrogés (CFSE) paraissent unanimes sur ce point.

5.2 Retour sur les hypothèses

Dans l'état actuel de l'industrie, le choix d'une combinaison de compilateurs ne rehausse pas l'indice de capabilité systématique (SC) d'un composant logiciel redondant. L'hypothèse reste négative, telle que décrite au chapitre 3.1.1.

Selon les résultats de l'enquête auprès d'une communauté spécialisée, au-delà de 85 % des spécialistes interrogés font preuve de réserve pour appliquer au logiciel, des mesures telles que le CCF qu'ils estiment adaptées au matériel. De plus, les simulations de calculs exposés aux chapitres 4.3.2 et 4.3.3 démontrent que, pour faire varier l'indice SC, une technique de défense doit réduire le CCF au-delà de 50 %. Un seul spécialiste sur 13 (8 %) estime que la diversité de compilateurs réduit l'indice dans des proportions susceptibles d'augmenter le SC.

Inclure la diversité de compilateurs au calcul de la variable CCF découle d'une tentative analogue à celle de Alena Griffiths [32], Peter B. Ladkin [24] ou T. Fujiwara, M. Kimura, Y. Satoh et *al* [33], pour objectiver l'indice SC d'un logiciel critique. Les formules de la norme intègrent le CCF de telle sorte qu'elle reste susceptible d'influencer de manière prépondérante le taux de panne moyen d'un système [11, Chap. B.3.2.2.2]. Cette variable s'applique normalement au matériel, mais le choix de la valeur prend la forme d'une appréciation subjective [11, Chap. D.1.4]. Néanmoins, pour que la diversité de compilateurs soit prise en compte comme technique défensive contre les CCF, les recherches doivent prouver aux

acteurs de l'industrie que les mesures de qualité logiciel se calculent avec une précision équivalente à leurs homologues établis pour le matériel.

5.3 Démonstration de la validité des résultats

Le sondage dénombre 34 participants au total. Une première ventilation permet de retirer 16 formulaires invalides ou incomplets. Les 18 sondés retenus se distribuent comme suit :

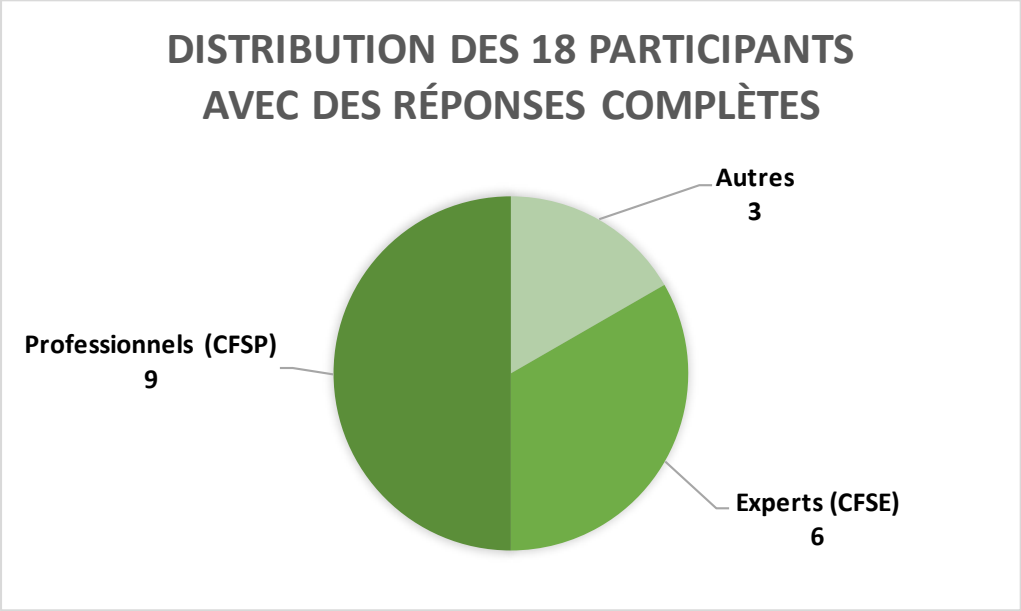


Figure 22 Distribution des formulaires complétés

La Figure 22 illustre le nombre de participations avec des données complètes. Parmi ces 18 formulaires retenus, deux questions d'élimination permettent d'appliquer un filtre additionnel pour juger de la pertinence des réponses :

1. Question no 2 : Équivalence du concept FIT, SFF et DC pour les logiciels.

   Les experts et professionnels certifiés doivent attester ou réfuter l'impact sur les indicateurs FIT, SFF et DC d'une analyse dynamique issue de tests par classes d'équivalences. Or, les méthodes pour calculer ces variables ne sont documentées que pour le matériel. La réponse doit être négative.

2. Question no 4 : Classement des défaillances

   Les experts et professionnels certifiés doivent attester ou réfuter le classement des bogues de compilateurs parmi les erreurs non-détectées. Or, toute erreur de logiciel compte parmi les erreurs détectées. La réponse doit être négative.

Toute personne qui répond par la négative à une de ces deux questions devient susceptible de juger en connaissance de cause pour la suite du sondage. Ce deuxième filtre permet de d'éliminer cinq formulaires, pour finalement n'en considérer que 13 distribués comme suit :

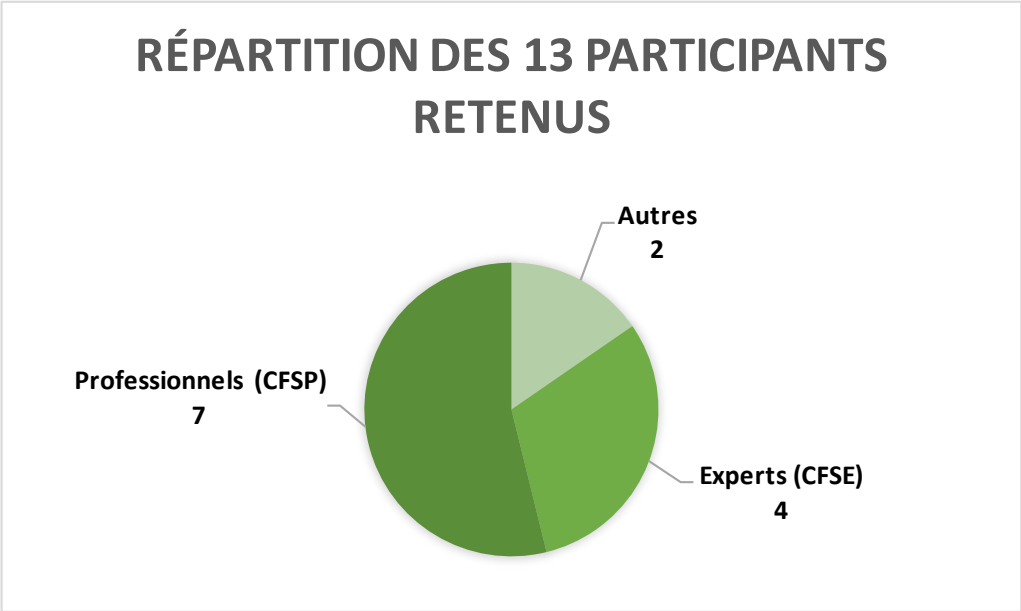


Figure 23 Distribution des formulaires retenus

L'opinion de la communauté à propos de la diversité de compilateurs découle des 13 formulaires répartis dans la Figure 23. La polarisation des réponses permet de discerner une certaine réserve de la communauté devant la suggestion d'appliquer l'indice CCF aux logiciels.

Pour les composants embarqués plus complexes de type *Soft PLC*, l'état des connaissances permet d'extraire au moins deux techniques appropriées pour une fixation objective du taux de panne :

1. Calculer la probabilité de défaillance du logiciel (λ) : $\lambda = -\frac{\ln \alpha}{t}$.

Dans l'article [1], les auteurs X Yang et *al*, démontrent le potentiel du principe de test flou appliqué au compilateur. Plutôt que de s'appliquer à trois compilateurs, la technique soumet trois *Soft PLC* issus de compilateurs différents.

Le taux de panne du logiciel (α) représente la variable clef du calcul. Le test flou permet de dénombrer les anomalies détectées à l'exécution de tests générés et vérifiés par un outil analogue à Csmith [1]. Le défi consiste à soumettre des paramètres limites ou fausses, à un grand nombre d'instances du même trio de *Soft PLC*, et dénombrer les pannes.

À titre d'exemple, supposons qu'un *Soft PLC* obtienne cinq pannes sur un banc de 100 000 ($\alpha = 5{,}0E-5$) instances exécutées pendant 604 800 secondes (70 jours, $t = 604\,800$). La probabilité de panne du *Soft PLC* en question se calcule comme suit :

$$PFH_{avg} = \lambda = -\frac{\ln\frac{5}{100\,000}}{6\,048\,000} = -\frac{\ln 5{,}0E-5}{6\,048\,000} = -\frac{-9{,}9}{6\,048\,000} = 1{,}6E-6 \approx \text{SIL1}$$

La norme 61508 déclare les pannes logiciels systématiques [23] ; le *Soft PLC* obtient alors la même capabilité systématique en mode d'exécution sur demande ou continue. Des recherches plus poussées devront éclairer la communauté à propos sur les moyens d'associer la probabilité de défaillance du logiciel à l'échelle SIL.

2. La combinaison et fusion SIL.

   Avec leur article [2], les auteurs Y. Langeron, A. Barros, A. Grall, et C. Bérenguer démontrent comment les indices SC se combinent pour calculer l'indice SIL du système. D'autres recherches devraient étudier dans quelle mesure les indices SC qui découlent du calcul décrit au point 1 peuvent se combiner.

Les propositions 1 et 2 présentent des moyens de quantifier la probabilité de panne du logiciel et l'effet de la diversité de compilateurs. La mise en œuvre du calcul décrit au point 1 présente en lui-même un défi technique et logistique. Des organismes, tels que le CRIM, le CRIQ ou le CNRC, offrent aux entrepreneurs les moyens et l'expertise pour exécuter de telles études.

Conclusion

L'hypothèse que la diversité et le choix de compilateurs puissent influencer l'indice SC du *Soft PLC* reste fausse. L'influence de ces deux variables se présente trop indirecte et difficile à mesurer pour que la technique puisse augmenter la valeur de façon notable.

Accorder une valeur mesurable à la diversité de compilateurs représente un défi de taille qui découle avant tout de la maturité des groupes de conception logicielle [60]. Les acteurs qui gravitent autour de la sécurité fonctionnelle doivent calculer les probabilités de pannes en fonction de caractéristiques contradictoires entre le matériel et le logiciel. Le premier permet une caractérisation précise et objective, qui conduit à des calculs vérifiables. Le second découle de l'esprit et dépend d'une compréhension plus subjective des spécifications. Une génération de chercheurs tels que Alena Griffiths [32], Peter B. Ladkin [24] ou T. Fujiwara, M. Kimura, Y. Satoh et *al* [33] tentent d'instaurer des mesures communes entre ces contraires. Cependant, de telles recherches restent épisodiques, si bien que les experts et professionnels certifiés en sécurité fonctionnelle réaffirment leur scepticisme.

L'approche théorique permet de constater que la diversité de compilateurs réduit le taux de panne d'un composant logiciel sans influencer à elle seule l'indice SC. Les huit simulations sur un composant homologué SC2, lorsque déployé sans redondance, et SC3, lorsque déployé en double redondance, se classe toujours SC3, si la diversité de compilateurs reste l'unique mesure de défense contre les défaillances de cause commune. Les graphiques de la simulation illustrent qu'une technique de défense doit réduire le taux CCF au-delà de 50 %, pour se répercuter de façon décisive sur l'indice SC. Les professionnels et experts en sécurité fonctionnelle reconnaissent quelques effets indirects à cette technique. Cependant, ils jugent difficile d'en quantifier l'impact sur la probabilité de panne et d'avantage pour justifier la réduction du CCF dans une proportion supérieure à 50 %. Un seul professionnel homologué sur 13 participants au sondage (environ 8%) consent à accorder un CCF significativement inférieure en présence d'une diversité de compilateurs, mais il ne s'agit que d'une estimation marginale.

Les sociétés post-modernes réglementent de plus en plus l'industrie pour assurer la sûreté de la vie, et l'environnement. Toute société qui vit une catastrophe en vient à chercher le maillon faible de l'accident. Les normes de sécurité fonctionnelles telles que CEI 61508 se développent et s'imposent rapidement comme des modèles de comparaison objectifs. Avec de tels protocoles, des règles et des critères de qualité mesurables assurent une gestion des risques dangereux associés aux opérations industrielles.

Dans un courant inverse, l'IdO s'impose en vagues déferlantes, sans aucune norme ou aucun standard précis. Ce nouveau courant inonde autant les secteurs industriels et domestiques, et exerce une pression croissante sur les coûts de conception. Le logiciel représente la clef de voûte qui permet d'exploiter l'IdO d'une manière universelle et à moindre coût. Les sociétés exigent des fabricants qu'ils maîtrisent la dangerosité de leurs opérations.

Employer des mesures de qualité objectives devient vital pour l'avenir. Il est possible d'évoquer une adaptation des formules de coûts développées par Halstead et McCabe [61] pour conduire au calcul de risques et à l'extrapolation de probabilités de pannes. Le matériel électrique ou électronique bénéficie de formules pour déterminer ces estimations, mais le génie logiciel tarde à imposer ses mesures. La communauté admet déjà la corrélation entre les couvertures et les tests par classes d'équivalences et le taux de panne du logiciel.

Liste des références

[1] X. Yang *et al.*, « Finding and understanding bugs in C compilers », dans *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation - PLDI '11*, 2011, vol. 46, nº 6, p. 283.

[2] Y. Langeron, A. Barros, A. Grall, et C. Bérenguer, « Combination of safety integrity levels (SILs): A study of IEC61508 merging rules », *J. Loss Prev. Process Ind.*, vol. 21, nº 4, p. 437- 449, 2008.

[3] E. R. Alphonsus et M. O. Abdullah, « A review on the applications of programmable logic controllers (PLCs) », *Renew. Sustain. Energy Rev.*, vol. 60, p. 1185- 1205, 2016.

[4] National Instruments, « Redundant System Basic Concepts - National Instruments », 2008. [En ligne]. Disponible à : http://www.ni.com/white-paper/6874/en/.

[5] D. John Satur, J. Kim, M. ki Bae, S. Y. Kim, et Y. Lee, « Cost effective alternative in meeting the required Safety Integrity Level (SIL) », *J. Loss Prev. Process Ind.*, vol. 40, p. 406‑418, 2016.

[6] CEI, « CEI 61508-1 », dans *Partie 1: Exigences générales*, 2.0., vol. 1, Genève, Suisse: Commission Électrotechnique Internationale, 2010.

[7] CEI, « CEI 61508-2 », dans *Partie 2: Exigences pour les systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité*, 2.0., Genève, Suisse: Commission Électrotechnique Internationale, 2010, p. 92.

[8] CEI, « CEI 61508-3 », dans *Partie 3: Exigences concernant les logiciels*, 2.0., vol. 3, Genève, Suisse: Commission Électrotechnique Internationale, 2010.

[9] CEI, « CEI 61508-4 », dans *Partie 4: Définitions et abréviations*, 2.0., vol. 4, Genève, Suisse: Commission Électrotechnique Internationale, 2010.

[10] CEI, « CEI 61508-5 », dans *Partie 5: Exemples de méthodes pour la détermination des niveaux de sécurité*, 2.0., vol. 5, Genève, Suisse: Commission Électrotechnique

Internationale, 2010, p. 41.

[11] CEI, « CEI 61508-6 », dans *Partie 6: Lignes directrices pour l'application de la CEI 61508-2 et de la CEI 61508-3*, 2.0., Genève, Suisse: Commission Électrotechnique Internationale, 2010.

[12] CEI, « CEI 61508-7 », dans *Partie 7: Présentation de techniques de mesures*, 2.0., vol. 7, Genève, Suisse: Commission Électrotechnique Internationale, 2010.

[13] S. Brown, « Overview of IEC 61508. Design of electrical/electronic/programmable electronic safety-related systems », *Comput. Control Eng. J.*, vol. 11, nº 1, p. 6, 2000.

[14] Q. Zhou *et al.*, « An embedded control system designed based on soft PLC », dans *Lecture Notes in Electrical Engineering*, 2014, vol. 308, p. 115- 120.

[15] Z. Wang, « The key technology research for embedded soft PLC control », dans *Proceedings - 2014 IEEE Workshop on Electronics, Computer and Applications, IWECA 2014*, 2014, p. 1045- 1048.

[16] Y. Ye, J. Wang, et L. Wang, « Research and implementation of embedded soft PLC system », dans *Proceedings - 5th International Conference on Intelligent Networks and Intelligent Systems, ICINIS 2012*, 2012, p. 166- 169.

[17] M. Mernik, J. Heering, et A. M. Sloane, « When and how to develop domain-specific languages », *ACM Comput. Surv.*, vol. 37, nº 4, p. 316- 344, déc. 2005.

[18] W. Bolton, *Programmable Logic Controllers*, 4e éd. Burlington, MA: Elsevier, 2006.

[19] Z. Chunjie et C. Hui, « Development of a PLC virtual machine orienting IEC 61131-3 standard », dans *2009 International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2009*, 2009, vol. 3, p. 374- 379.

[20] T. Goldschmidt, M. K. Murugaiah, C. Sonntag, B. Schlich, S. Biallas, et P. Weber, « Cloud-Based Control: A Multi-tenant, Horizontally Scalable *Soft PLC* », dans *Proceedings - 2015 IEEE 8th International Conference on Cloud Computing, CLOUD 2015*, 2015, p. 909- 916.

[21] Alfonso Velosa, J. F. Hines, H. LeHong, E. Perkins, et S. R.M, « Predicts 2015: The Internet of Things », *Gartner*, nº July 2012, p. 1- 40, 2015.

[22] Y. Lee, J. Jeong, et Y. Son, « Design and implementation of the secure compiler and virtual machine for developing secure IoT services », *Futur. Gener. Comput. Syst.*, 2016.

[23] R. Bell, « Introduction to IEC 61508 », *Conf. Res. Pract. Inf. Technol. Ser.*, vol. 55, p. 3- 12, 2005.

[24] P. B. Ladkin, « Assessing Critical SW as " Proven in Use ": Pitfalls and Possibilities », 2013.

[25] W. S. (Blacksafe C. L. . Black, « Prerequisites for IEC 61508 application », *IEE Semin. Program. Electron. Saf. Syst. Issues, Stand. Pract. Asp.*, vol. 2002, p. 4‑4, 2002.

[26] A. Mayr, R. Plösch, et M. Saft, « Towards an operational safety standard for software: Modelling IEC 61508 part 3 », dans *Proceedings - 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, ECBS 2011*, 2011, p. 97‑104.

[27] S. Y. Moon et R. Y. C. Kim, « Mapping SW Development Process with Safety Process for Safe Software », dans *2016 International Conference on Platform Technology and Service (PlatCon)*, 2016, p. 1‑5.

[28] W. S. (Blacksafe C. L. . Black, « IEC 61508 - what it doesn't tell you », *Comput. Control Eng. J.*, vol. 11, n° 1, p. 24‑27, 2000.

[29] T. Zhang, W. Long, et Y. Sato, « Availability of systems with self-diagnostic components - Applying Markov model to IEC 61508-6 », *Reliab. Eng. Syst. Saf.*, vol. 80, n° 2, p. 133‑141, 2003.

[30] H. Tang et L. Lu, « A quantitative software testing method for hardware and software integrated systems in safety critical applications », *Probabilistic Saf. Assess. Manag. PSAM*, vol. 12, n° June, 2014.

[31] Y. Xia, G. Yan, et Q. Si, « A study on the significance of software metrics in defect prediction », dans *Proceedings - 6th International Symposium on Computational Intelligence and Design, ISCID 2013*, 2013, vol. 2, p. 343‑346.

[32] A. Griffiths, « On proof-test intervals for safety functions implemented in software », dans

Conferences in Research and Practice in Information Technology Series, 2006, vol. 69, p. 23‑33.

[33] T. Fujiwara, M. Kimura, Y. Satoh, et S. Yamada, « A method of calculating safety integrity level for IEC 61508 conformity software », dans *Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing, PRDC*, 2011, p. 296‑301.

[34] M. A. Lundteigen et M. Rausand, « Common cause failures in safety instrumented systems on oil and gas installations: Implementing defense measures through function testing », *J. Loss Prev. Process Ind.*, vol. 20, nº 3, p. 218‑229, 2007.

[35] A. C. Torres-Echeverría, S. Martorell, et H. A. Thompson, « Design optimization of a safety-instrumented system based on RAMS+C addressing IEC 61508 requirements and diverse redundancy », *Reliab. Eng. Syst. Saf.*, vol. 94, nº 2, p. 162‑179, 2009.

[36] R. Wood et J. Mullens, *Taxonomy for Common- - - Cause Failure Vulnerability and Mitigation*, nº September. 2015.

[37] H. Jahanian et Q. Mahboob, « SIL determination as a utility-based decision process », *Process Saf. Environ. Prot.*, vol. 102, p. 757‑767, 2016.

[38] C. Wei, B. Xiaohong, et Z. Tingdi, « A Study on Compiler Selection in Safety-critical Redundant System based on Airworthiness Requirement », *Procedia Eng.*, vol. 17, p. 497‑504, 2011.

[39] S. S. Muchnick, *Advanced compiler design and implementation*. San Francisco, Calif: Morgan Kaufmann, 2014.

[40] M. De Sousa, « Data-type checking of IEC61131-3 ST and IL applications », dans *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2012, p. 1‑8.

[41] CEI, « CEI 61499-1 », dans *Partie 1: Architecture*, vol. 1, Genève, Suisse: Commission Électrotechnique Internationale, 2008.

[42] X. Leroy, « Formal verification of a realistic compiler », *Commun. ACM*, vol. 52, nº 7, p. 107‑115, 2009.

[43] D. Obermann et J. Börcsök, « Two-Way-Compiler: Additional data saving for generating the original source code of a binary program », *ARCS Work.*, p. 1‑4, 2012.

[44] S. Lerner, T. Millstein, et C. Chambers, « Automatically proving the correctness of compiler optimizations », *PLDI*, vol. 38, p. 220, 2003.

[45] T. Hoare, « The Verifying Compiler - A grand challenge for computing research », *J. ACM*, vol. 50, nº 1, p. 63‑69, 2003.

[46] L. Zuck, A. Pnueli, Y. Fang, et B. Goldberg, « TVOC: A translation validator for optimizing compilers », *Electron. Notes Theor. Comput. Sci.*, vol. 65, nº 2, p. 5‑21, 2002.

[47] D. J. Pavey et L. A. Winsborrow, « Demonstrating Equivalence of Source Code and PROM Contents », *Comput. J.*, vol. 36, nº 7, p. 654‑667, juill. 1993.

[48] R. Langmann et L. Rojas-Peña, « PLCs as Industry 4.0 Components in Laboratory Applications », *Int. J. Online Eng.*, vol. 12, nº 7, p. 37, juill. 2016.

[49] M. Driver, « IT Market Clock for Programming Languages », nº September, p. 1‑74, 2016.

[50] D. Crocker, « Can C++ Be Made As Safe As SPARK? », *Ada Lett.*, vol. 34, nº 3, p. 5‑12, 2014.

[51] CEI, « CEI 61131-3 », dans *Partie 3: Langages de programmation*, 3.0., Genève, Suisse: Commission Électrotechnique Internationale, 2003, p. 226.

[52] E. Eide et J. Regehr, « Volatiles are miscompiled, and what to do about it », dans *Proceedings of the 7th ACM international conference on Embedded software - EMSOFT '08*, 2008, p. 255.

[53] X. Leroy, « Compiler Verification for fun and profit », *Form. Methods Comput. Des. (FMCAD 2014)*, vol. 8044, p. 9, 2014.

[54] J. Zhao, S. Nagarakatte, M. M. K. Martin, et S. Zdancewic, « Formal Verification of SSA-based Optimizations for LLVM », *Acm Sigplan …*, vol. 48, nº 6, p. 175–186, juin 2013.

[55] B. Carré et J. Garnsworthy, « SPARK - An annotated Ada subset for safety-critical programming », *Proc. TRI-ADA*, p. 392‑402, 1990.

[56] P. Courtieu, M. V. Aponte, J. Guitton, Z. Zhang, J. Belt, et T. Jennings, « Towards The Formalization of SPARK 2014 Semantics With Explicit Run-time Checks Using Coq Categories and Subject Descriptors », dans *HILT '13*, 2014, vol. 33, nº 3, p. 21‑22.

[57] D. Crocker, « Safe Object-Oriented Software: The Verified Design-By-Contract Paradigm », dans *Practical Elements of Safety*, London: Springer London, 2004, p. 19‑41.

[58] ISO/IEC TR 24772:2013, « Guidance to avoiding vulnerabilities in programming languages through language selection and use », 2013.

[59] C. Sun, V. Le, Q. Zhang, et Z. Su, « Toward understanding compiler bugs in GCC and LLVM », dans *Proceedings of the 25th International Symposium on Software Testing and Analysis - ISSTA 2016*, 2016, p. 294‑305.

[60] K. Zahra, F. Azam, F. Ilyas, H. Faisal, N. Ambreen, et N. Gondal, « Success factors of organizational change in software process improvement: A systematic literature review », dans *ACM International Conference Proceeding Series*, 2017, vol. Part F1265, p. 155‑160.

[61] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, et C. Soubervielle-Montalvo, « Source code metrics: A systematic mapping study », *J. Syst. Softw.*, vol. 128, p. 164‑197, juin 2017.

[62] J. Gray, « What Next? A Dozen Information-Technology Research Goals », *J. ACM*, vol. 50, nº 1, p. 41‑57, janv. 1999.

[63] D. S. Touretzky, « Source vs. Object Code: A False Dichotomy », *Carnegie Mellon University, Computer Science Department*, 2000. [En ligne]. Disponible à: https://www.cs.cmu.edu/~dst/DeCSS/object-code.txt.

[64] D. Brumley, J. Lee, E. J. Schwartz, et M. Woo, « Native x86 Decompilation Using Semantics-Preserving Structural Analysis and Iterative Control-Flow Structuring », dans *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, 2013, p. 353‑368.

[65] C. Mendis *et al.*, « Helium: Lifting High-Performance Stencil Kernels from Stripped x86

Binaries to Halide DSL Code », *Proc. 36th ACM SIGPLAN Conf. Program. Lang. Des. Implement.*, vol. 50, n° 6, p. 391‑402, juin 2015.

[66] Y. Liu, S. Höglund, A. H. Khan, et I. Porres, « A feasibility study on the validation of domain specific languages using OWL 2 reasoners », dans *CEUR Workshop Proceedings*, 2010, vol. 604.

[67] « Zoos - AtlanMod ». [En ligne]. Disponible à: http://web.emn.fr/x-info/atlanmod/index.php?title=Zoos.

[68] O. Semeráth, Á. Barta, Á. Horváth, Z. Szatmári, et D. Varrós, « Formal validation of domain-specific languages with derived features and well-formedness constraints », *Softw. {&} Syst. Model.*, p. 1‑36, juill. 2015.

[69] A. Horvath, A. Hegedus, M. Bur, D. Varro, R. R. Starr, et S. Mirachi, « Hardware-software allocation specification of IMA systems for early simulation », dans *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, 2014, p. 4D31-4D315.

[70] L. Dalgaard, T. Heikkilae, et J. Koskinen, « The R3-COP Decision Support Framework for Autonomous Robotic System Design », dans *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*, 2014, p. 1‑8.

[71] P. James et M. Roggenbach, « Encapsulating Formal Methods within Domain Specific Languages: A Solution for Verifying Railway Scheme Plans », *Math. Comput. Sci.*, vol. 8, n° 1, p. 11‑38, mars 2014.

[72] P. James, « Designing Domain Specific Languages for Verification and Applications to the Railway Domain Statement », 2014.

Annexe I

Cycle de vie de la sécurité fonctionnelle

I.1 Cycle de vie de sécurité global

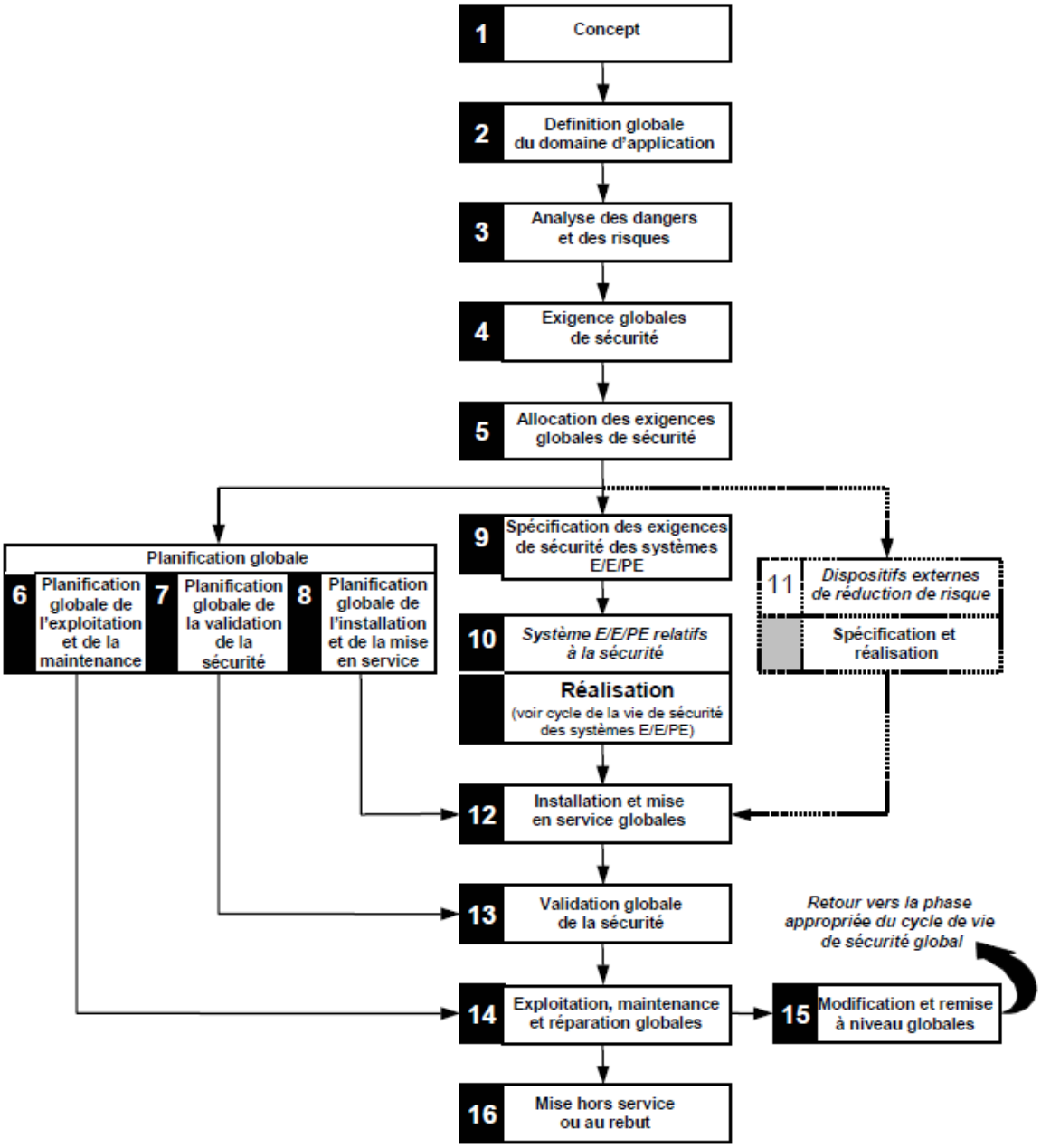


Figure 24 Cycle de vie de sécurité global ([8]**, Figure 2)**

I.2 Cycle de vie de sécurité du logiciel

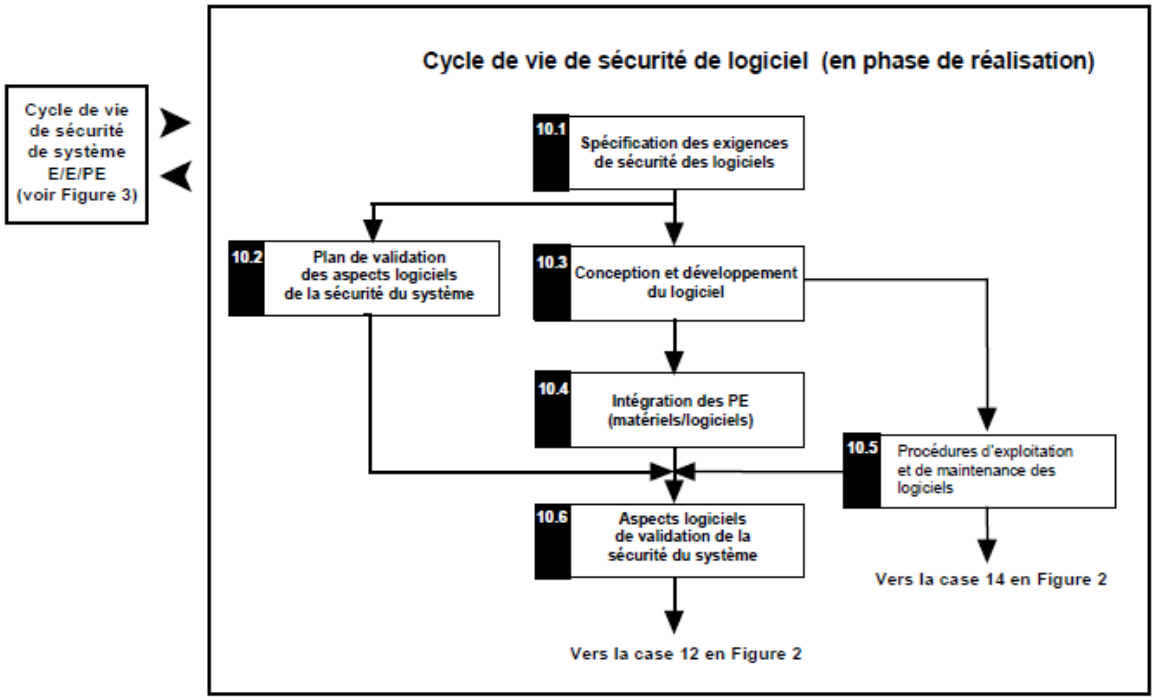


Figure 25 Cycle de vie de sécurité du logiciel ([8]**, Figure 4)**

Annexe II

Tableaux de calculs détaillés

II.1 Mode de fonctionnement en faible sollicitation

La norme propose des tableaux de PFD_{avg}, calculés en fonction de six taux de défaillance et six architectures et quatre intervalles d'essais [11, Chap. B.3.2.3].

II.1.1 Intervalle entre essais de six mois et MTTR de huit heures

Tableau 11 SIL en faible sollicitation, en fonction d'un DC de 0 %, d'essais périodiques de six mois et MTTR de huit heures

Architecture	DC	λ_D = 0,5E-07			λ_D = 2,5E-07			λ_D = 0,5E-06		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
		β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %
1oo1 (voir la Note 2)	0 %	1,1E-04			5,5E-04			1,1E-03		
	60 %	4,4E-05			2,2E-04			4,4E-04		
	90 %	1,1E-05			5,7E-05			1,1E-04		
	99 %	1,5E-06			7,5E-06			1,5E-05		
1oo2	0 %	2,2E-06	1,1E-05	2,2E-05	1,1E-05	5,5E-05	1,1E-04	2,4E-05	1,1E-04	2,2E-04
	60 %	8,8E-07	4,4E-06	8,8E-06	4,5E-06	2,2E-05	4,4E-05	9,1E-06	4,4E-05	8,8E-05
	90 %	2,2E-07	1,1E-06	2,2E-06	1,1E-06	5,6E-06	1,1E-05	2,3E-06	1,1E-05	2,2E-05
	99 %	2,6E-08	1,3E-07	2,6E-07	1,3E-07	6,5E-07	1,3E-06	2,6E-07	1,3E-06	2,6E-06
2oo2 (voir la Note 2)	0 %	2,2E-04			1,1E-03			2,2E-03		
	60 %	8,8E-05			4,4E-04			8,8E-04		
	90 %	2,3E-05			1,1E-04			2,3E-04		
	99 %	3,0E-06			1,5E-05			3,0E-05		
1oo2D (voir la Note 3)	0 %	2,2E-06	1,1E-05	2,2E-05	1,1E-05	5,5E-05	1,1E-04	2,4E-05	1,1E-04	2,2E-04
	60 %	1,4E-06	4,9E-06	9,3E-06	7,1E-06	2,5E-05	4,7E-05	1,4E-05	5,0E-05	9,3E-05
	90 %	4,3E-07	1,3E-06	2,4E-06	2,2E-06	6,6E-06	1,2E-05	4,3E-06	1,3E-05	2,4E-05
	99 %	6,0E-08	1,5E-07	2,6E-07	3,0E-07	7,4E-07	1,3E-06	6,0E-07	1,5E-06	2,6E-06
2oo3	0 %	2,2E-06	1,1E-05	2,2E-05	1,2E-05	5,6E-05	1,1E-04	2,7E-05	1,1E-04	2,2E-04
	60 %	8,9E-07	4,4E-06	8,8E-06	4,6E-06	2,2E-05	4,4E-05	9,6E-06	4,5E-05	8,9E-05
	90 %	2,2E-07	1,1E-06	2,2E-06	1,1E-06	5,6E-06	1,1E-05	2,3E-06	1,1E-05	2,2E-05
	99 %	2,6E-08	1,3E-07	2,6E-07	1,3E-07	6,5E-07	1,3E-06	2,6E-07	1,3E-06	2,6E-06
1oo3	0 %	2,2E-06	1,1E-05	2,2E-05	1,1E-05	5,5E-05	1,1E-04	2,2E-05	1,1E-04	2,2E-04
	60 %	8,8E-07	4,4E-06	8,8E-06	4,4E-06	2,2E-05	4,4E-05	8,8E-06	4,4E-05	8,8E-05
	90 %	2,2E-07	1,1E-06	2,2E-06	1,1E-06	5,6E-06	1,1E-05	2,2E-06	1,1E-05	2,2E-05
	99 %	2,6E-08	1,3E-07	2,6E-07	1,3E-07	6,5E-07	1,3E-06	2,6E-07	1,3E-06	2,6E-06

Architecture	DC	λ_D = 2,5E-06			λ_D = 0,5E-05			λ_D = 2,5E-05		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
		β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %
1oo1 (voir la Note 2)	0 %	5,5E-03			1,1E-02			5,5E-02		
	60 %	2,2E-03			4,4E-03			2,2E-02		
	90 %	5,7E-04			1,1E-03			5,7E-03		
	99 %	7,5E-05			1,5E-04			7,5E-04		
1oo2	0 %	1,5E-04	5,8E-04	1,1E-03	3,7E-04	1,2E-03	2,3E-03	5,0E-03	8,8E-03	1,4E-02
	60 %	5,0E-05	2,3E-04	4,5E-04	1,1E-04	4,6E-04	9,0E-04	1,1E-03	2,8E-03	4,9E-03
	90 %	1,2E-05	5,6E-05	1,1E-04	2,4E-05	1,1E-04	2,2E-04	1,5E-04	6,0E-04	1,2E-03
	99 %	1,3E-06	6,5E-06	1,3E-05	2,6E-06	1,3E-05	2,6E-05	1,4E-05	6,6E-05	1,3E-04
2oo2 (voir la Note 2)	0 %	1,1E-02			2,2E-02			>1E-01		
	60 %	4,4E-03			8,8E-03			4,4E-02		
	90 %	1,1E-03			2,3E-03			1,1E-02		
	99 %	1,5E-04			3,0E-04			1,5E-03		
1oo2D (voir la Note 3)	0 %	1,5E-04	5,8E-04	1,1E-03	3,8E-04	1,2E-03	2,3E-03	5,0E-03	9,0E-03	1,4E-02
	60 %	7,7E-05	2,5E-04	4,7E-04	1,7E-04	5,2E-04	9,5E-04	1,3E-03	3,0E-03	5,1E-03
	90 %	2,2E-05	6,6E-05	1,2E-04	4,5E-05	1,3E-04	2,4E-04	2,6E-04	6,9E-04	1,2E-03
	99 %	3,0E-06	7,4E-06	1,3E-05	6,0E-06	1,5E-05	2,6E-05	3,0E-05	7,4E-05	1,3E-04
2oo3	0 %	2,3E-04	6,5E-04	1,2E-03	6,8E-04	1,5E-03	2,5E-03	1,3E-02	1,5E-02	1,9E-02
	60 %	6,3E-05	2,4E-04	4,6E-04	1,6E-04	5,1E-04	9,4E-04	2,3E-03	3,9E-03	5,9E-03
	90 %	1,2E-05	5,7E-05	1,1E-04	2,7E-05	1,2E-04	2,3E-04	2,4E-04	6,8E-04	1,2E-03
	99 %	1,3E-06	6,5E-06	1,3E-05	2,7E-06	1,3E-05	2,6E-05	1,5E-05	6,7E-05	1,3E-04
1oo3	0 %	1,1E-04	5,5E-04	1,1E-03	2,2E-04	1,1E-03	2,2E-03	1,4E-03	5,7E-03	1,1E-02
	60 %	4,4E-05	2,2E-04	4,4E-04	8,8E-05	4,4E-04	8,8E-04	4,6E-04	2,2E-03	4,4E-03
	90 %	1,1E-05	5,6E-05	1,1E-04	2,2E-05	1,1E-04	2,2E-04	1,1E-04	5,6E-04	1,1E-03
	99 %	1,3E-06	6,5E-06	1,3E-05	2,6E-06	1,3E-05	2,6E-05	1,3E-05	6,5E-05	1,3E-04

NOTE 1 Ce tableau donne des exemples de valeurs de PFD_G calculées en appliquant les équations données en B.3.2 et en fonction des hypothèses énoncées en B.3.1. Si le sous-système capteur, logique ou élément final comprend seulement un groupe de canaux à logique majoritaire, alors PFD_G est équivalent respectivement à PFD_S, PFD_L ou PFD_{FE} (voir B.3.2.1).

NOTE 2 Le tableau suppose que $\beta = 2 \times \beta_D$. Pour les architectures 1oo1 et 2oo2, les valeurs de β et de β_D n'affectent pas la probabilité moyenne de défaillance.

NOTE 3 Le taux de défaillance en sécurité est supposé égal au taux de défaillance dangereuse et K = 0,98.

II.1.2 Intervalle entre essais d'un an et MTTR de huit heures

Tableau 12 SIL en faible sollicitation, en fonction d'un DC de 0 %, d'essais périodiques d'un an et MTTR de huit heures

Architecture	DC	λ_D = 0,5E-07			λ_D = 2,5E-07			λ_D = 0,5E-06		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
		β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %
1oo1 (voir la Note 2)	0 %	2,2E-04			1,1E-03			2,2E-03		
	60 %	8,8E-05			4,4E-04			8,8E-04		
	90 %	2,2E-05			1,1E-04			2,2E-04		
	99 %	2,6E-06			1,3E-05			2,6E-05		
1oo2	0 %	4,4E-06	2,2E-05	4,4E-05	2,3E-05	1,1E-04	2,2E-04	5,0E-05	2,2E-04	4,4E-04
	60 %	1,8E-06	8,8E-06	1,8E-05	9,0E-06	4,4E-05	8,8E-05	1,9E-05	8,9E-05	1,8E-04
	90 %	4,4E-07	2,2E-06	4,4E-06	2,2E-06	1,1E-05	2,2E-05	4,5E-06	2,2E-05	4,4E-05
	99 %	4,8E-08	2,4E-07	4,8E-07	2,4E-07	1,2E-06	2,4E-06	4,8E-07	2,4E-06	4,8E-06
2oo2 (voir la Note 2)	0 %	4,4E-04			2,2E-03			4,4E-03		
	60 %	1,8E-04			8,8E-04			1,8E-03		
	90 %	4,5E-05			2,2E-04			4,5E-04		
	99 %	5,2E-06			2,6E-05			5,2E-05		
1oo2D (voir la Note 3)	0 %	4,5E-06	2,2E-05	4,4E-05	2,4E-05	1,1E-04	2,2E-04	5,0E-05	2,2E-04	4,4E-04
	60 %	2,8E-06	9,8E-06	1,9E-05	1,4E-05	4,9E-05	9,3E-05	2,9E-05	9,9E-05	1,9E-04
	90 %	8,5E-07	2,6E-06	4,8E-06	4,3E-06	1,3E-05	2,4E-05	8,5E-06	2,6E-05	4,8E-05
	99 %	1,0E-07	2,8E-07	5,0E-07	5,2E-07	1,4E-06	2,5E-06	1,0E-06	2,8E-06	5,0E-06
2oo3	0 %	4,6E-06	2,2E-05	4,4E-05	2,7E-05	1,1E-04	2,2E-04	6,2E-05	2,4E-04	4,5E-04
	60 %	1,8E-06	8,8E-06	1,8E-05	9,5E-06	4,5E-05	8,8E-05	2,1E-05	9,1E-05	1,8E-04
	90 %	4,4E-07	2,2E-06	4,4E-06	2,3E-06	1,1E-05	2,2E-05	4,6E-06	2,2E-05	4,4E-05
	99 %	4,8E-08	2,4E-07	4,8E-07	2,4E-07	1,2E-06	2,4E-06	4,8E-07	2,4E-06	4,8E-06
1oo3	0 %	4,4E-06	2,2E-05	4,4E-05	2,2E-05	1,1E-04	2,2E-04	4,4E-05	2,2E-04	4,4E-04
	60 %	1,8E-06	8,8E-06	1,8E-05	8,8E-06	4,4E-05	8,8E-05	1,8E-05	8,8E-05	1,8E-04
	90 %	4,4E-07	2,2E-06	4,4E-06	2,2E-06	1,1E-05	2,2E-05	4,4E-06	2,2E-05	4,4E-05
	99 %	4,8E-08	2,4E-07	4,8E-07	2,4E-07	1,2E-06	2,4E-06	4,8E-07	2,4E-06	4,8E-06

Architecture	DC	λ_D = 2,5E-06			λ_D = 0,5E-05			λ_D = 2,5E-05		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
		β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %
1oo1 (voir la Note 2)	0 %	1,1E-02			2,2E-02			>1E-01		
	60 %	4,4E-03			8,8E-03			4,4E-02		
	90 %	1,1E-03			2,2E-03			1,1E-02		
	99 %	1,3E-04			2,6E-04			1,3E-03		
1oo2	0 %	3,7E-04	1,2E-03	2,3E-03	1,1E-03	2,7E-03	4,8E-03	1,8E-02	2,4E-02	3,2E-02
	60 %	1,1E-04	4,6E-04	9,0E-04	2,8E-04	9,7E-04	1,8E-03	3,4E-03	6,6E-03	1,1E-02
	90 %	2,4E-05	1,1E-04	2,2E-04	5,1E-05	2,3E-04	4,5E-04	3,8E-04	1,3E-03	2,3E-03
	99 %	2,4E-06	1,2E-05	2,4E-05	4,9E-06	2,4E-05	4,8E-05	2,6E-05	1,2E-04	2,4E-04
2oo2 (voir la Note 2)	0 %	2,2E-02			4,4E-02			>1E-01		
	60 %	8,8E-03			1,8E-02			8,8E-02		
	90 %	2,2E-03			4,5E-03			2,2E-02		
	99 %	2,6E-04			5,2E-04			2,6E-03		
1oo2D (voir la Note 3)	0 %	3,8E-04	1,2E-03	2,3E-03	1,1E-03	2,7E-03	4,9E-03	1,8E-02	2,5E-02	3,4E-02
	60 %	1,7E-04	5,1E-04	9,5E-04	3,8E-04	1,1E-03	1,9E-03	3,9E-03	7,1E-03	1,1E-02
	90 %	4,4E-05	1,3E-04	2,4E-04	9,1E-05	2,7E-04	4,8E-04	5,8E-04	1,4E-03	2,5E-03
	99 %	5,2E-06	1,4E-05	2,5E-05	1,0E-05	2,8E-05	5,0E-05	5,4E-05	1,4E-04	2,5E-04
2oo3	0 %	6,8E-04	1,5E-03	2,5E-03	2,3E-03	3,8E-03	5,6E-03	4,8E-02	5,0E-02	5,3E-02
	60 %	1,6E-04	5,1E-04	9,4E-04	4,8E-04	1,1E-03	2,0E-03	8,4E-03	1,1E-02	1,5E-02
	90 %	2,7E-05	1,2E-04	2,3E-04	6,4E-05	2,4E-04	4,6E-04	7,1E-04	1,6E-03	2,6E-03
	99 %	2,5E-06	1,2E-05	2,4E-05	5,1E-06	2,4E-05	4,8E-05	3,1E-05	1,3E-04	2,5E-04
1oo3	0 %	2,2E-04	1,1E-03	2,2E-03	4,6E-04	2,2E-03	4,4E-03	4,7E-03	1,3E-02	2,3E-02
	60 %	8,8E-05	4,4E-04	8,8E-04	1,8E-04	8,8E-04	1,8E-03	1,0E-03	4,5E-03	8,9E-03
	90 %	2,2E-05	1,1E-04	2,2E-04	4,4E-05	2,2E-04	4,4E-04	2,2E-04	1,1E-03	2,2E-03
	99 %	2,4E-06	1,2E-05	2,4E-05	4,8E-06	2,4E-05	4,8E-05	2,4E-05	1,2E-04	2,4E-04

NOTE 1 Ce tableau donne des exemples de valeurs de PFD_G calculées en appliquant les équations données en B.3.2 et en fonction des hypothèses énoncées en B.3.1. Si le sous-système capteur, logique ou élément final comprend seulement un groupe de canaux à logique majoritaire, alors PFD_G est équivalent respectivement à PFD_S, PFD_L ou PFD_{FE} (voir B.3.2.1).

NOTE 2 Le tableau suppose que $\beta = 2 \times \beta_D$. Pour les architectures 1oo1 et 2oo2, les valeurs de β et de β_D n'affectent pas la probabilité moyenne de défaillance.

NOTE 3 Le taux de défaillance en sécurité est supposé égal au taux de défaillance dangereuse et K = 0,98.

II.1.3 Intervalle entre essais de deux ans et MTTR de huit heures

Tableau 13 SIL en faible sollicitation, en fonction d'un DC de 0 %, d'essais périodiques de deux ans et MTTR de huit heures

Architecture	DC	λ_D = 0,5E-07			λ_D = 2,5E-07			λ_D = 0,5E-06		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
		β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %
1oo1 (voir la Note 2)	0 %	4,4E-04			2,2E-03			4,4E-03		
	60 %	1,8E-04			8,8E-04			1,8E-03		
	90 %	4,4E-05			2,2E-04			4,4E-04		
	99 %	4,8E-06			2,4E-05			4,8E-05		
1oo2	0 %	9,0E-06	4,4E-05	8,8E-05	5,0E-05	2,2E-04	4,4E-04	1,1E-04	4,6E-04	8,9E-04
	60 %	3,5E-06	1,8E-05	3,5E-05	1,9E-05	8,9E-05	1,8E-04	3,9E-05	1,8E-04	3,5E-04
	90 %	8,8E-07	4,4E-06	8,8E-06	4,5E-06	2,2E-05	4,4E-05	9,1E-06	4,4E-05	8,8E-05
	99 %	9,2E-08	4,6E-07	9,2E-07	4,6E-07	2,3E-06	4,6E-06	9,2E-07	4,6E-06	9,2E-06
2oo2 (voir la Note 2)	0 %	8,8E-04			4,4E-03			8,8E-03		
	60 %	3,5E-04			1,8E-03			3,5E-03		
	90 %	8,8E-05			4,4E-04			8,8E-04		
	99 %	9,6E-06			4,8E-05			9,6E-05		
1oo2D (voir la Note 3)	0 %	9,0E-06	4,4E-05	8,8E-05	5,0E-05	2,2E-04	4,4E-04	1,1E-04	4,6E-04	9,0E-04
	60 %	5,7E-06	2,0E-05	3,7E-05	2,9E-05	9,9E-05	1,9E-04	6,0E-05	2,0E-04	3,7E-04
	90 %	1,7E-06	5,2E-06	9,6E-06	8,5E-06	2,6E-05	4,8E-05	1,7E-05	5,2E-05	9,6E-05
	99 %	1,9E-07	5,4E-07	9,8E-07	9,5E-07	2,7E-06	4,9E-06	1,9E-06	5,4E-06	9,8E-06
2oo3	0 %	9,5E-06	4,4E-05	8,8E-05	6,2E-05	2,3E-04	4,5E-04	1,6E-04	5,0E-04	9,3E-04
	60 %	3,6E-06	1,8E-05	3,5E-05	2,1E-05	9,0E-05	1,8E-04	4,7E-05	1,9E-04	3,6E-04
	90 %	8,9E-07	4,4E-06	8,8E-06	4,6E-06	2,2E-05	4,4E-05	9,6E-06	4,5E-05	8,9E-05
	99 %	9,2E-08	4,6E-07	9,2E-07	4,6E-07	2,3E-06	4,6E-06	9,3E-07	4,6E-06	9,2E-06
1oo3	0 %	8,8E-06	4,4E-05	8,8E-05	4,4E-05	2,2E-04	4,4E-04	8,8E-05	4,4E-04	8,8E-04
	60 %	3,5E-06	1,8E-05	3,5E-05	1,8E-05	8,8E-05	1,8E-04	3,5E-05	1,8E-04	3,5E-04
	90 %	8,8E-07	4,4E-06	8,8E-06	4,4E-06	2,2E-05	4,4E-05	8,8E-06	4,4E-05	8,8E-05
	99 %	9,2E-08	4,6E-07	9,2E-07	4,6E-07	2,3E-06	4,6E-06	9,2E-07	4,6E-06	9,2E-06

Architecture	DC	λ_D =2,5E-06			λ_D = 0,5E-05			λ_D = 2,5E-05		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
		β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %
1oo1 (voir la Note 2)	0 %	2,2E-02			4,4E-02			>1E-01		
	60 %	8,8E-03			1,8E-02			8,8E-02		
	90 %	2,2E-03			4,4E-03			2,2E-02		
	99 %	2,4E-04			4,8E-04			2,4E-03		
1oo2	0 %	1,1E-03	2,7E-03	4,8E-03	3,3E-03	6,5E-03	1,0E-02	6,6E-02	7,4E-02	8,5E-02
	60 %	2,8E-04	9,7E-04	1,8E-03	7,5E-04	2,1E-03	3,8E-03	1,2E-02	1,8E-02	2,5E-02
	90 %	5,0E-05	2,3E-04	4,5E-04	1,1E-04	4,6E-04	9,0E-04	1,1E-03	2,8E-03	4,9E-03
	99 %	4,7E-06	2,3E-05	4,6E-05	9,5E-06	4,6E-05	9,2E-05	5,4E-05	2,4E-04	4,6E-04
2oo2 (voir la Note 2)	0 %	4,4E-02			8,8E-02			>1E-01		
	60 %	1,8E-02			3,5E-02			>1E-01		
	90 %	4,4E-03			8,8E-03			4,4E-02		
	99 %	4,8E-04			9,6E-04			4,8E-03		
1oo2D (voir la Note 3)	0 %	1,1E-03	2,7E-03	4,8E-03	3,4E-03	6,6E-03	1,1E-02	6,7E-02	7,7E-02	9,0E-02
	60 %	3,8E-04	1,1E-03	1,9E-03	9,6E-04	2,3E-03	4,0E-03	1,3E-02	1,9E-02	2,6E-02
	90 %	9,0E-05	2,6E-04	4,8E-04	1,9E-04	5,4E-04	9,8E-04	1,5E-03	3,2E-03	5,3E-03
	99 %	9,6E-06	2,7E-05	4,9E-05	1,9E-05	5,4E-05	9,8E-05	1,0E-04	2,8E-04	5,0E-04
2oo3	0 %	2,3E-03	3,7E-03	5,6E-03	8,3E-03	1,1E-02	1,4E-02	1,9E-01	1,8E-01	1,7E-01
	60 %	4,8E-04	1,1E-03	2,0E-03	1,6E-03	2,8E-03	4,4E-03	3,2E-02	3,5E-02	4,0E-02
	90 %	6,3E-05	2,4E-04	4,6E-04	1,6E-04	5,1E-04	9,4E-04	2,4E-03	4,0E-03	6,0E-03
	99 %	4,8E-06	2,3E-05	4,6E-05	1,0E-05	4,7E-05	9,2E-05	6,9E-05	2,5E-04	4,8E-04
1oo3	0 %	4,6E-04	2,2E-03	4,4E-03	1,0E-03	4,5E-03	8,9E-03	2,4E-02	3,7E-02	5,5E-02
	60 %	1,8E-04	8,8E-04	1,8E-03	3,6E-04	1,8E-03	3,5E-03	3,1E-03	9,9E-03	1,8E-02
	90 %	4,4E-05	2,2E-04	4,4E-04	8,8E-05	4,4E-04	8,8E-04	4,6E-04	2,2E-03	4,4E-03
	99 %	4,6E-06	2,3E-05	4,6E-05	9,2E-06	4,6E-05	9,2E-05	4,6E-05	2,3E-04	4,6E-04

NOTE 1 Ce tableau donne des exemples de valeurs de PFD_G calculées en appliquant les équations données en B.3.2 et en fonction des hypothèses énoncées en B.3.1. Si le sous-système capteur, logique ou élément final comprend seulement un groupe de canaux à logique majoritaire, alors PFD_G est équivalent respectivement à PFD_S, PFD_L ou PFD_{FE} (voir B.3.2.1).

NOTE 2 Le tableau suppose que $\beta = 2 \times \beta_D$. Pour les architectures 1oo1 et 2oo2, les valeurs de β et de β_D n'affectent pas la probabilité moyenne de défaillance.

NOTE 3 Le taux de défaillance en sécurité est supposé égal au taux de défaillance dangereuse et K = 0,98.

II.1.4 Intervalle entre essais de dix ans et MTTR de huit heures

Tableau 14 SIL en faible sollicitation, en fonction d'un DC de 0 %, d'essais périodiques de dix ans et MTTR de huit heures

Architecture	DC	λ_D = 0,5E-07			λ_D = 2,5E-07			λ_D = 0,5E-06		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
		β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %
1oo1 (voir la Note 2)	0 %	2,2E-03			1,1E-02			2,2E-02		
	60 %	8,8E-04			4,4E-03			8,8E-03		
	90 %	2,2E-04			1,1E-03			2,2E-03		
	99 %	2,2E-05			1,1E-04			2,2E-04		
1oo2	0 %	5,0E-05	2,2E-04	4,4E-04	3,7E-04	1,2E-03	2,3E-03	1,1E-03	2,7E-03	4,8E-03
	60 %	1,9E-05	8,9E-05	1,8E-04	1,1E-04	4,6E-04	9,0E-04	2,7E-04	9,6E-04	1,8E-03
	90 %	4,4E-06	2,2E-05	4,4E-05	2,3E-05	1,1E-04	2,2E-04	5,0E-05	2,2E-04	4,4E-04
	99 %	4,4E-07	2,2E-06	4,4E-06	2,2E-06	1,1E-05	2,2E-05	4,5E-06	2,2E-05	4,4E-05
2oo2 (voir la Note 2)	0 %	4,4E-03			2,2E-02			4,4E-02		
	60 %	1,8E-03			8,8E-03			1,8E-02		
	90 %	4,4E-04			2,2E-03			4,4E-03		
	99 %	4,5E-05			2,2E-04			4,5E-04		
1oo2D (voir la Note 3)	0 %	5,0E-05	2,2E-04	4,4E-04	3,7E-04	1,2E-03	2,3E-03	1,1E-03	2,7E-03	4,8E-03
	60 %	2,9E-05	9,9E-05	1,9E-04	1,7E-04	5,1E-04	9,5E-04	3,8E-04	1,1E-03	1,9E-03
	90 %	8,4E-06	2,6E-05	4,8E-05	4,3E-05	1,3E-04	2,4E-04	9,0E-05	2,6E-04	4,8E-04
	99 %	8,9E-07	2,6E-06	4,8E-06	4,5E-06	1,3E-05	2,4E-05	8,9E-06	2,6E-05	4,8E-05
2oo3	0 %	6,2E-05	2,3E-04	4,5E-04	6,8E-04	1,5E-03	2,5E-03	2,3E-03	3,7E-03	5,6E-03
	60 %	2,1E-05	9,0E-05	1,8E-04	1,6E-04	5,0E-04	9,3E-04	4,7E-04	1,1E-03	2,0E-03
	90 %	4,6E-06	2,2E-05	4,4E-05	2,7E-05	1,1E-04	2,2E-04	6,3E-05	2,4E-04	4,5E-04
	99 %	4,4E-07	2,2E-06	4,4E-06	2,3E-06	1,1E-05	2,2E-05	4,6E-06	2,2E-05	4,4E-05
1oo3	0 %	4,4E-05	2,2E-04	4,4E-04	2,2E-04	1,1E-03	2,2E-03	4,6E-04	2,2E-03	4,4E-03
	60 %	1,8E-05	8,8E-05	1,8E-04	8,8E-05	4,4E-04	8,8E-04	1,8E-04	8,8E-04	1,8E-03
	90 %	4,4E-06	2,2E-05	4,4E-05	2,2E-05	1,1E-04	2,2E-04	4,4E-05	2,2E-04	4,4E-04
	99 %	4,4E-07	2,2E-06	4,4E-06	2,2E-06	1,1E-05	2,2E-05	4,4E-06	2,2E-05	4,4E-05

Architecture	DC	λ_D =2,5E-06			λ_D = 0,5E-05			λ_D = 2,5E-05		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
		β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %
1oo1 (voir la Note 2)	0 %	>1E-01			>1E-01			>1E-01		
	60 %	4,4E-02			8,8E-02			>1E-01		
	90 %	1,1E-02			2,2E-02			>1E-01		
	99 %	1,1E-03			2,2E-03			1,1E-02		
1oo2	0 %	1,8E-02	2,4E-02	3,2E-02	6,6E-02	7,4E-02	8,5E-02	>1E-01	>1E-01	>1E-01
	60 %	3,4E-03	6,6E-03	1,1E-02	1,2E-02	1,8E-02	2,5E-02	>1E-01	>1E-01	>1E-01
	90 %	3,8E-04	1,2E-03	2,3E-03	1,1E-03	2,8E-03	4,9E-03	1,8E-02	2,5E-02	3,5E-02
	99 %	2,4E-05	1,1E-04	2,2E-04	5,1E-05	2,3E-04	4,5E-04	3,8E-04	1,3E-03	2,3E-03
2oo2 (voir la Note 2)	0 %	>1E-01			>1E-01			>1E-01		
	60 %	8,8E-02			>1E-01			>1E-01		
	90 %	2,2E-02			4,4E-02			>1E-01		
	99 %	2,2E-03			4,5E-03			2,2E-02		
1oo2D (voir la Note 3)	0 %	1,8E-02	2,5E-02	3,3E-02	6,6E-02	7,7E-02	9,0E-02	1,6E+00	1,5E+00	1,4E+00
	60 %	3,9E-03	7,1E-03	1,1E-02	1,3E-02	1,9E-02	2,6E-02	2,6E-01	2,7E-01	2,8E-01
	90 %	5,7E-04	1,4E-03	2,5E-03	1,5E-03	3,1E-03	5,2E-03	2,0E-02	2,7E-02	3,5E-02
	99 %	4,6E-05	1,3E-04	2,4E-04	9,5E-05	2,7E-04	4,9E-04	6,0E-04	1,5E-03	2,5E-03
2oo3	0 %	4,8E-02	5,0E-02	5,3E-02	1,9E-01	1,8E-01	1,7E-01	4,6E+00	4,0E+00	3,3E+00
	60 %	8,3E-03	1,1E-02	1,4E-02	3,2E-02	3,5E-02	4,0E-02	7,6E-01	7,1E-01	6,6E-01
	90 %	6,9E-04	1,5E-03	2,6E-03	2,3E-03	3,9E-03	5,9E-03	4,9E-02	5,4E-02	6,0E-02
	99 %	2,7E-05	1,2E-04	2,3E-04	6,4E-05	2,4E-04	4,6E-04	7,1E-04	1,6E-03	2,6E-03
1oo3	0 %	4,7E-03	1,3E-02	2,3E-02	2,4E-02	3,7E-02	5,5E-02	2,5E+00	2,0E+00	1,6E+00
	60 %	1,0E-03	4,5E-03	8,9E-03	3,0E-03	9,8E-03	1,8E-02	1,7E-01	1,8E-01	1,9E-01
	90 %	2,2E-04	1,1E-03	2,2E-03	4,6E-04	2,2E-03	4,4E-03	4,8E-03	1,3E-02	2,4E-02
	99 %	2,2E-05	1,1E-04	2,2E-04	4,4E-05	2,2E-04	4,4E-04	2,2E-04	1,1E-03	2,2E-03

NOTE 1 Ce tableau donne des exemples de valeurs de PFD_G calculées en appliquant les équations données en B.3.2 et en fonction des hypothèses énoncées en B.3.1. Si le sous-système capteur, logique ou élément final comprend seulement un groupe de canaux à logique majoritaire, alors PFD_G est équivalent respectivement à PFD_S, PFD_L ou PFD_{FE} (voir B.3.2.1).

NOTE 2 Le tableau suppose que $\beta = 2 \times \beta_D$. Pour les architectures 1oo1 et 2oo2, les valeurs de β et de β_D n'affectent pas la probabilité moyenne de défaillance.

NOTE 3 Le taux de défaillance en sécurité est supposé égal au taux de défaillance dangereuse et K = 0,98.

II.2 Mode de fonctionnement en sollicitation élevée ou continue

La norme propose des tableaux de PFH_{avg}, calculés en fonction de six taux de défaillance et six architectures et quatre intervalles d'essais [11, Chap. B.3.3.3].

II.2.1 Intervalle entre essais d'un mois et MTTR de huit heures

Tableau 15 SIL en sollicitation élevée, en fonction d'un DC de 0 %, d'essais périodiques d'un mois et MTTR de huit heures

Architecture	DC	λ_D = 0,5E-07			λ_D = 2,5E-07			λ_D = 0,5E-06		
		β = 2 % β_D = 1 %	β = 10 % β_D = 5 %	β = 20 % β_D = 10 %	β = 2 % β_D = 1 %	β = 10 % β_D = 5 %	β = 20 % β_D = 10 %	β = 2 % β_D = 1 %	β = 10 % β_D = 5 %	β = 20 % β_D = 10 %
1oo1 (voir la Note 2)	0 %	5.0E-08			2.5E-07			5.0E-07		
	60 %	2.0E-08			1.0E-07			2.0E-07		
	90 %	5.0E-09			2.5E-08			5.0E-08		
	99 %	5.0E-10			2.5E-09			5.0E-09		
1oo2	0 %	1.0E-09	5.0E-09	1.0E-08	5.0E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07
	60 %	4.0E-10	2.0E-09	4.0E-09	2.0E-09	1.0E-08	2.0E-08	4.0E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09
2oo2 (voir la Note 2)	0 %	1.0E-07			5.0E-07			1.0E-06		
	60 %	4.0E-08			2.0E-07			4.0E-07		
	90 %	1.0E-08			5.0E-08			1.0E-07		
	99 %	1.0E-09			5.0E-09			1.0E-08		
1oo2D (voir la Note 3)	0 %	1.0E-09	5.0E-09	1.0E-08	5.0E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07
	60 %	1.6E-09	3.2E-09	5.2E-09	8.0E-09	1.6E-08	2.6E-08	1.6E-08	3.2E-08	5.2E-08
	90 %	1.9E-09	2.3E-09	2.8E-09	9.5E-09	1.2E-08	1.4E-08	1.9E-08	2.3E-08	2.8E-08
	99 %	2.0E-09	2.0E-09	2.1E-09	1.0E-08	1.0E-08	1.0E-08	2.0E-08	2.0E-08	2.1E-08
2oo3	0 %	1.0E-09	5.0E-09	1.0E-08	5.1E-09	2.5E-08	5.0E-08	1.1E-08	5.0E-08	1.0E-07
	60 %	4.0E-10	2.0E-09	4.0E-09	2.0E-09	1.0E-08	2.0E-08	4.1E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09
1oo3	0 %	1.0E-09	5.0E-09	1.0E-08	5.0E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07
	60 %	4.0E-10	2.0E-09	4.0E-09	2.0E-09	1.0E-08	2.0E-08	4.0E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09

Architecture	DC	λ_D =2,5E-06			λ_D = 0,5E-05			λ_D = 2,5E-05		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
1oo1 (voir la Note 2)	0 %	2.5E-06			5.0E-06			2.5E-05		
	60 %	1.0E-06			2.0E-06			1.0E-05		
	90 %	2.5E-07			5.0E-07			2.5E-06		
	99 %	2.5E-08			5.0E-08			2.5E-07		
1oo2	0 %	5.4E-08	2.5E-07	5.0E-07	1.2E-07	5.2E-07	1.0E-06	9.5E-07	2.9E-06	5.3E-06
	60 %	2.1E-08	1.0E-07	2.0E-07	4.3E-08	2.0E-07	4.0E-07	2.7E-07	1.1E-06	2.1E-06
	90 %	5.1E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07	5.5E-08	2.5E-07	5.0E-07
	99 %	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	5.1E-09	2.5E-08	5.0E-08
2oo2 (voir la Note 2)	0 %	5.0E-06			1.0E-05			5.0E-05		
	60 %	2.0E-06			4.0E-06			2.0E-05		
	90 %	5.0E-07			1.0E-06			5.0E-06		
	99 %	5.0E-08			1.0E-07			5.0E-07		
1oo2D (voir la Note 3)	0 %	5.4E-08	2.5E-07	5.0E-07	1.2E-07	5.2E-07	1.0E-06	9.5E-07	2.9E-06	5.3E-06
	60 %	8.1E-08	1.6E-07	2.6E-07	1.6E-07	3.2E-07	5.2E-07	8.7E-07	1.7E-06	2.7E-06
	90 %	9.5E-08	1.2E-07	1.4E-07	1.9E-07	2.3E-07	2.8E-07	9.6E-07	1.2E-06	1.4E-06
	99 %	1.0E-07	1.0E-07	1.0E-07	2.0E-07	2.0E-07	2.1E-07	1.0E-06	1.0E-06	1.0E-06
2oo3	0 %	6.3E-08	2.6E-07	5.1E-07	1.5E-07	5.5E-07	1.0E-06	1.8E-06	3.6E-06	5.9E-06
	60 %	2.2E-08	1.0E-07	2.0E-07	4.9E-08	2.1E-07	4.1E-07	4.2E-07	1.2E-06	2.2E-06
	90 %	5.2E-09	2.5E-08	5.0E-08	1.1E-08	5.1E-08	1.0E-07	6.6E-08	2.6E-07	5.1E-07
	99 %	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	5.4E-09	2.5E-08	5.0E-08
1oo3	0 %	5.0E-08	2.5E-07	5.0E-07	1.0E-07	5.0E-07	1.0E-06	5.1E-07	2.5E-06	5.0E-06
	60 %	2.0E-08	1.0E-07	2.0E-07	4.0E-08	2.0E-07	4.0E-07	2.0E-07	1.0E-06	2.0E-06
	90 %	5.0E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07	5.0E-08	2.5E-07	5.0E-07
	99 %	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	5.0E-09	2.5E-08	5.0E-08

NOTE 1 Ce tableau donne des exemples de valeurs de PFH_G calculées en appliquant les équations données en B.3.3 et en fonction des hypothèses énoncées en B.3.1. Si le sous-système capteur, logique ou élément final comprend seulement un groupe de canaux à logique majoritaire, alors PFH_G est équivalent respectivement à PFH_S, PFH_L ou PFH_{FE} (voir B.3.3.1).

NOTE 2 Le tableau suppose que $\beta = 2 \times \beta_D$. Pour les architectures 1oo1 et 2oo2, les valeurs de β et de β_D n'affectent pas la fréquence moyenne d'une défaillance dangereuse.

NOTE 3 Le taux de défaillance en sécurité est supposé égal au taux de défaillance dangereuse et K = 0,98.

II.2.2 Intervalle entre essais de trois mois et MTTR de huit heures

Tableau 16 SIL en sollicitation élevée, en fonction d'un DC de 0 %, d'essais périodiques de trois mois et MTTR de huit heures

Architecture	DC	λ_D = 0,5E-07			λ_D = 2,5E-07			λ_D = 0,5E-06		
		β = 2 % β_D = 1 %	β = 10 % β_D = 5 %	β = 20 % β_D = 10 %	β = 2 % β_D = 1 %	β = 10 % β_D = 5 %	β = 20 % β_D = 10 %	β = 2 % β_D = 1 %	β = 10 % β_D = 5 %	β = 20 % β_D = 10 %
1oo1 (voir la Note 2)	0 %	5.0E-08			2.5E-07			5.0E-07		
	60 %	2.0E-08			1.0E-07			2.0E-07		
	90 %	5.0E-09			2.5E-08			5.0E-08		
	99 %	5.0E-10			2.5E-09			5.0E-09		
1oo2	0 %	1.0E-09	5.0E-09	1.0E-08	5.1E-09	2.5E-08	5.0E-08	1.1E-08	5.0E-08	1.0E-07
	60 %	4.0E-10	2.0E-09	4.0E-09	2.0E-09	1.0E-08	2.0E-08	4.1E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09
2oo2 (voir la Note 2)	0 %	1.0E-07			5.0E-07			1.0E-06		
	60 %	4.0E-08			2.0E-07			4.0E-07		
	90 %	1.0E-08			5.0E-08			1.0E-07		
	99 %	1.0E-09			5.0E-09			1.0E-08		
1oo2D (voir la Note 3)	0 %	1.0E-09	5.0E-09	1.0E-08	5.1E-09	2.5E-08	5.0E-08	1.1E-08	5.0E-08	1.0E-07
	60 %	1.6E-09	3.2E-09	5.2E-09	8.0E-09	1.6E-08	2.6E-08	1.6E-08	3.2E-08	5.2E-08
	90 %	1.9E-09	2.3E-09	2.8E-09	9.5E-09	1.2E-08	1.4E-08	1.9E-08	2.3E-08	2.8E-08
	99 %	2.0E-09	2.0E-09	2.1E-09	1.0E-08	1.0E-08	1.0E-08	2.0E-08	2.0E-08	2.1E-08
2oo3	0 %	1.0E-09	5.0E-09	1.0E-08	5.4E-09	2.5E-08	5.0E-08	1.2E-08	5.1E-08	1.0E-07
	60 %	4.0E-10	2.0E-09	4.0E-09	2.1E-09	1.0E-08	2.0E-08	4.3E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09
1oo3	0 %	1.0E-09	5.0E-09	1.0E-08	5.0E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07
	60 %	4.0E-10	2.0E-09	4.0E-09	2.0E-09	1.0E-08	2.0E-08	4.0E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09

Architecture	DC	λ_D = 2,5E-06			λ_D = 0,5E-05			λ_D = 2,5E-05		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
1oo1 (voir la Note 2)	0 %	2.5E-06			5.0E-06			2.5E-05		
	60 %	1.0E-06			2.0E-06			1.0E-05		
	90 %	2.5E-07			5.0E-07			2.5E-06		
	99 %	2.5E-08			5.0E-08			2.5E-07		
1oo2	0 %	6.3E-08	2.6E-07	5.1E-07	1.5E-07	5.4E-07	1.0E-06	1.8E-06	3.6E-06	5.9E-06
	60 %	2.2E-08	1.0E-07	2.0E-07	4.9E-08	2.1E-07	4.1E-07	4.2E-07	1.2E-06	2.2E-06
	90 %	5.1E-09	2.5E-08	5.0E-08	1.1E-08	5.0E-08	1.0E-07	6.4E-08	2.6E-07	5.1E-07
	99 %	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	5.2E-09	2.5E-08	5.0E-08
2oo2 (voir la Note 2)	0 %	5.0E-06			1.0E-05			5.0E-05		
	60 %	2.0E-06			4.0E-06			2.0E-05		
	90 %	5.0E-07			1.0E-06			5.0E-06		
	99 %	5.0E-08			1.0E-07			5.0E-07		
1oo2D (voir la Note 3)	0 %	6.3E-08	2.6E-07	5.1E-07	1.5E-07	5.4E-07	1.0E-06	1.8E-06	3.6E-06	5.9E-06
	60 %	8.2E-08	1.6E-07	2.6E-07	1.7E-07	3.3E-07	5.3E-07	1.0E-06	1.8E-06	2.8E-06
	90 %	9.5E-08	1.2E-07	1.4E-07	1.9E-07	2.3E-07	2.8E-07	9.6E-07	1.2E-06	1.4E-06
	99 %	1.0E-07	1.0E-07	1.0E-07	2.0E-07	2.0E-07	2.1E-07	1.0E-06	1.0E-06	1.0E-06
2oo3	0 %	9.0E-08	2.8E-07	5.3E-07	2.6E-07	6.3E-07	1.1E-06	4.5E-06	5.9E-06	7.6E-06
	60 %	2.6E-08	1.1E-07	2.0E-07	6.6E-08	2.2E-07	4.2E-07	8.5E-07	1.6E-06	2.5E-06
	90 %	5.4E-09	2.5E-08	5.0E-08	1.2E-08	5.1E-08	1.0E-07	9.3E-08	2.9E-07	5.3E-07
	99 %	5.1E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	5.7E-09	2.6E-08	5.1E-08
1oo3	0 %	5.0E-08	2.5E-07	5.0E-07	1.0E-07	5.0E-07	1.0E-06	5.5E-07	2.5E-06	5.0E-06
	60 %	2.0E-08	1.0E-07	2.0E-07	4.0E-08	2.0E-07	4.0E-07	2.0E-07	1.0E-06	2.0E-06
	90 %	5.0E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07	5.0E-08	2.5E-07	5.0E-07
	99 %	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	5.0E-09	2.5E-08	5.0E-08

NOTE 1 Ce tableau donne des exemples de valeurs de PFH_G calculées en appliquant les équations données en B.3.3 et en fonction des hypothèses énoncées en B.3.1. Si le sous-système capteur, logique ou élément final comprend seulement un groupe de canaux à logique majoritaire, alors PFH_G est équivalent respectivement à PFH_S, PFH_L ou PFH_{FE} (voir B.3.3.1).

NOTE 2 Le tableau suppose que $\beta = 2 \times \beta_D$. Pour les architectures 1oo1 et 2oo2, les valeurs de β et de β_D n'affectent pas la fréquence moyenne d'une défaillance dangereuse.

NOTE 3 Le taux de défaillance en sécurité est supposé égal au taux de défaillance dangereuse et K = 0,98.

II.2.3 Intervalle entre essais de six mois et MTTR de huit heures

Tableau 17 SIL en sollicitation élevée, en fonction d'un DC de 0 %, d'essais périodiques de six mois et MTTR de huit heures

Architecture	DC	λ_D = 0,5E-07			λ_D = 2,5E-07			λ_D = 0,5E-06		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
		β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %
1oo1 (voir la Note 2)	0 %	5.0E-08			2.5E-07			5.0E-07		
	60 %	2.0E-08			1.0E-07			2.0E-07		
	90 %	5.0E-09			2.5E-08			5.0E-08		
	99 %	5.0E-10			2.5E-09			5.0E-09		
1oo2	0 %	1.0E-09	5.0E-09	1.0E-08	5.3E-09	2.5E-08	5.0E-08	1.1E-08	5.1E-08	1.0E-07
	60 %	4.0E-10	2.0E-09	4.0E-09	2.0E-09	1.0E-08	2.0E-08	4.2E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09
2oo2 (voir la Note 2)	0 %	1.0E-07			5.0E-07			1.0E-06		
	60 %	4.0E-08			2.0E-07			4.0E-07		
	90 %	1.0E-08			5.0E-08			1.0E-07		
	99 %	1.0E-09			5.0E-09			1.0E-08		
1oo2D (voir la Note 3)	0 %	1.0E-09	5.0E-09	1.0E-08	5.3E-09	2.5E-08	5.0E-08	1.1E-08	5.1E-08	1.0E-07
	60 %	1.6E-09	3.2E-09	5.2E-09	8.0E-09	1.6E-08	2.6E-08	1.6E-08	3.2E-08	5.2E-08
	90 %	1.9E-09	2.3E-09	2.8E-09	9.5E-09	1.2E-08	1.4E-08	1.9E-08	2.3E-08	2.8E-08
	99 %	2.0E-09	2.0E-09	2.1E-09	1.0E-08	1.0E-08	1.0E-08	2.0E-08	2.0E-08	2.1E-08
2oo3	0 %	1.0E-09	5.0E-09	1.0E-08	5.8E-09	2.6E-08	5.1E-08	1.3E-08	5.3E-08	1.0E-07
	60 %	4.1E-10	2.0E-09	4.0E-09	2.1E-09	1.0E-08	2.0E-08	4.5E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.1E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09
1oo3	0 %	1.0E-09	5.0E-09	1.0E-08	5.0E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07
	60 %	4.0E-10	2.0E-09	4.0E-09	2.0E-09	1.0E-08	2.0E-08	4.0E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09

Architecture	DC	λ_D = 2,5E-06			λ_D = 0,5E-05			λ_D = 2,5E-05		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
1oo1 (voir la Note 2)	0 %	2.5E-06			5.0E-06			2.5E-05		
	60 %	1.0E-06			2.0E-06			1.0E-05		
	90 %	2.5E-07			5.0E-07			2.5E-06		
	99 %	2.5E-08			5.0E-08			2.5E-07		
1oo2	0 %	7.6E-08	2.7E-07	5.2E-07	2.1E-07	5.9E-07	1.1E-06	3.1E-06	4.7E-06	6.8E-06
	60 %	2.4E-08	1.0E-07	2.0E-07	5.7E-08	2.1E-07	4.1E-07	6.3E-07	1.4E-06	2.3E-06
	90 %	5.3E-09	2.5E-08	5.0E-08	1.1E-08	5.1E-08	1.0E-07	7.8E-08	2.7E-07	5.2E-07
	99 %	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	5.4E-09	2.5E-08	5.0E-08
2oo2 (voir la Note 2)	0 %	5.0E-06			1.0E-05			5.0E-05		
	60 %	2.0E-06			4.0E-06			2.0E-05		
	90 %	5.0E-07			1.0E-06			5.0E-06		
	99 %	5.0E-08			1.0E-07			5.0E-07		
1oo2D (voir la Note 3)	0 %	7.6E-08	2.7E-07	5.2E-07	2.1E-07	5.9E-07	1.1E-06	3.1E-06	4.7E-06	6.8E-06
	60 %	8.4E-08	1.6E-07	2.6E-07	1.8E-07	3.3E-07	5.3E-07	1.2E-06	2.0E-06	2.9E-06
	90 %	9.5E-08	1.2E-07	1.4E-07	1.9E-07	2.3E-07	2.8E-07	9.8E-07	1.2E-06	1.4E-06
	99 %	1.0E-07	1.0E-07	1.0E-07	2.0E-07	2.0E-07	2.1E-07	1.0E-06	1.0E-06	1.0E-06
2oo3	0 %	1.3E-07	3.2E-07	5.5E-07	4.2E-07	7.7E-07	1.2E-06	8.4E-06	9.2E-06	1.0E-05
	60 %	3.3E-08	1.1E-07	2.1E-07	9.1E-08	2.4E-07	4.4E-07	1.5E-06	2.1E-06	2.9E-06
	90 %	5.8E-09	2.6E-08	5.1E-08	1.3E-08	5.3E-08	1.0E-07	1.3E-07	3.2E-07	5.6E-07
	99 %	5.1E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	6.1E-09	2.6E-08	5.1E-08
1oo3	0 %	5.0E-08	2.5E-07	5.0E-07	1.0E-07	5.0E-07	1.0E-06	7.1E-07	2.7E-06	5.1E-06
	60 %	2.0E-08	1.0E-07	2.0E-07	4.0E-08	2.0E-07	4.0E-07	2.1E-07	1.0E-06	2.0E-06
	90 %	5.0E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07	5.0E-08	2.5E-07	5.0E-07
	99 %	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	5.0E-09	2.5E-08	5.0E-08

NOTE 1 Ce tableau donne des exemples de valeurs de PFH_G calculées en appliquant les équations données en B.3.3 et en fonction des hypothèses énoncées en B.3.1. Si le sous-système capteur, logique ou élément final comprend seulement un groupe de canaux à logique majoritaire, alors PFH_G est équivalent respectivement à PFH_S, PFH_L ou PFH_{FE} (voir B.3.3.1).

NOTE 2 Le tableau suppose que $\beta = 2 \times \beta_D$. Pour les architectures 1oo1 et 2oo2, les valeurs de β et de β_D n'affectent pas la fréquence moyenne d'une défaillance dangereuse.

NOTE 3 Le taux de défaillance en sécurité est supposé égal au taux de défaillance dangereuse et K = 0,98.

II.2.4 Intervalle entre essais d'un an et MTTR de huit heures

Tableau 18 SIL en sollicitation élevée, en fonction d'un DC de 0 %, d'essais périodiques d'un an et MTTR de huit heures

Architecture	DC	λ_D = 0,5E-07			λ_D = 2,5E-07			λ_D = 0,5E-06		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
		β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %	β_D = 1 %	β_D = 5 %	β_D = 10 %
1oo1 (voir la Note 2)	0 %	5.0E-08			2.5E-07			5.0E-07		
	60 %	2.0E-08			1.0E-07			2.0E-07		
	90 %	5.0E-09			2.5E-08			5.0E-08		
	99 %	5.0E-10			2.5E-09			5.0E-09		
1oo2	0 %	1.0E-09	5.0E-09	1.0E-08	5.5E-09	2.5E-08	5.0E-08	1.2E-08	5.2E-08	1.0E-07
	60 %	4.0E-10	2.0E-09	4.0E-09	2.1E-09	1.0E-08	2.0E-08	4.3E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.1E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09
2oo2 (voir la Note 2)	0 %	1.0E-07			5.0E-07			1.0E-06		
	60 %	4.0E-08			2.0E-07			4.0E-07		
	90 %	1.0E-08			5.0E-08			1.0E-07		
	99 %	1.0E-09			5.0E-09			1.0E-08		
1oo2D (voir la Note 3)	0 %	1.0E-09	5.0E-09	1.0E-08	5.5E-09	2.5E-08	5.0E-08	1.2E-08	5.2E-08	1.0E-07
	60 %	1.6E-09	3.2E-09	5.2E-09	8.1E-09	1.6E-08	2.6E-08	1.6E-08	3.2E-08	5.2E-08
	90 %	1.9E-09	2.3E-09	2.8E-09	9.5E-09	1.2E-08	1.4E-08	1.9E-08	2.3E-08	2.8E-08
	99 %	2.0E-09	2.0E-09	2.1E-09	1.0E-08	1.0E-08	1.0E-08	2.0E-08	2.0E-08	2.1E-08
2oo3	0 %	1.1E-09	5.1E-09	1.0E-08	6.6E-09	2.6E-08	5.1E-08	1.6E-08	5.5E-08	1.0E-07
	60 %	4.1E-10	2.0E-09	4.0E-09	2.3E-09	1.0E-08	2.0E-08	5.0E-09	2.1E-08	4.1E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.2E-10	2.5E-09	5.0E-09	1.1E-09	5.1E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09
1oo3	0 %	1.0E-09	5.0E-09	1.0E-08	5.0E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07
	60 %	4.0E-10	2.0E-09	4.0E-09	2.0E-09	1.0E-08	2.0E-08	4.0E-09	2.0E-08	4.0E-08
	90 %	1.0E-10	5.0E-10	1.0E-09	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08
	99 %	1.0E-11	5.0E-11	1.0E-10	5.0E-11	2.5E-10	5.0E-10	1.0E-10	5.0E-10	1.0E-09

Architecture	DC	λ_D = 2,5E-06			λ_D = 0,5E-05			λ_D = 2,5E-05		
		β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %	β = 2 %	β = 10 %	β = 20 %
1oo1 (voir la Note 2)	0 %	2.5E-06			5.0E-06			2.5E-05		
	60 %	1.0E-06			2.0E-06			1.0E-05		
	90 %	2.5E-07			5.0E-07			2.5E-06		
	99 %	2.5E-08			5.0E-08			2.5E-07		
1oo2	0 %	1.0E-07	2.9E-07	5.4E-07	3.1E-07	6.8E-07	1.1E-06	5.8E-06	6.9E-06	8.5E-06
	60 %	2.9E-08	1.1E-07	2.1E-07	7.4E-08	2.3E-07	4.2E-07	1.1E-06	1.7E-06	2.6E-06
	90 %	5.5E-09	2.5E-08	5.0E-08	1.2E-08	5.2E-08	1.0E-07	1.0E-07	3.0E-07	5.4E-07
	99 %	5.1E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	5.6E-09	2.6E-08	5.0E-08
2oo2 (voir la Note 2)	0 %	5.0E-06			1.0E-05			5.0E-05		
	60 %	2.0E-06			4.0E-06			2.0E-05		
	90 %	5.0E-07			1.0E-06			5.0E-06		
	99 %	5.0E-08			1.0E-07			5.0E-07		
1oo2D (voir la Note 3)	0 %	1.0E-07	2.9E-07	5.4E-07	3.1E-07	6.8E-07	1.1E-06	5.8E-06	6.9E-06	8.5E-06
	60 %	8.9E-08	1.7E-07	2.7E-07	1.9E-07	3.5E-07	5.4E-07	1.7E-06	2.3E-06	3.2E-06
	90 %	9.6E-08	1.2E-07	1.4E-07	1.9E-07	2.3E-07	2.8E-07	1.0E-06	1.2E-06	1.4E-06
	99 %	1.0E-07	1.0E-07	1.0E-07	2.0E-07	2.0E-07	2.1E-07	1.0E-06	1.0E-06	1.0E-06
2oo3	0 %	2.1E-07	3.8E-07	6.1E-07	7.3E-07	1.0E-06	1.4E-06	1.6E-05	1.6E-05	1.6E-05
	60 %	4.6E-08	1.2E-07	2.2E-07	1.4E-07	2.9E-07	4.7E-07	2.8E-06	3.2E-06	3.8E-06
	90 %	6.6E-09	2.6E-08	5.1E-08	1.6E-08	5.6E-08	1.0E-07	2.1E-07	3.9E-07	6.2E-07
	99 %	5.2E-10	2.5E-09	5.0E-09	1.1E-09	5.1E-09	1.0E-08	6.9E-09	2.7E-08	5.1E-08
1oo3	0 %	5.1E-08	2.5E-07	5.0E-07	1.1E-07	5.1E-07	1.0E-06	1.4E-06	3.2E-06	5.5E-06
	60 %	2.0E-08	1.0E-07	2.0E-07	4.0E-08	2.0E-07	4.0E-07	2.6E-07	1.0E-06	2.0E-06
	90 %	5.0E-09	2.5E-08	5.0E-08	1.0E-08	5.0E-08	1.0E-07	5.1E-08	2.5E-07	5.0E-07
	99 %	5.0E-10	2.5E-09	5.0E-09	1.0E-09	5.0E-09	1.0E-08	5.0E-09	2.5E-08	5.0E-08

NOTE 1 Ce tableau donne des exemples de valeurs de PFH_G calculées en appliquant les équations données en B.3.3 et en fonction des hypothèses énoncées en B.3.1. Si le sous-système capteur, logique ou élément final comprend seulement un groupe de canaux à logique majoritaire, alors PFH_G est équivalent respectivement à PFH_S, PFH_L ou PFH_{FE} (voir B.3.3.1).

NOTE 2 Le tableau suppose que $\beta = 2 \times \beta_D$. Pour les architectures 1oo1 et 2oo2, les valeurs de β et de β_D n'affectent pas la fréquence moyenne d'une défaillance dangereuse.

NOTE 3 Le taux de défaillance en sécurité est supposé égal au taux de défaillance dangereuse et K = 0,98.

Annexe III

Questionnaire de sondage

Le questionnaire se destine aux experts et professionnels certifiés en sécurité fonctionnelle. Il se présente en anglais pour assurer un plus grand nombre de réponses.

III.1 Page d'accueil

La page d'accueil offre une mise en contexte choc, pour frapper l'imagination et solliciter un auditoire plus large d'experts impliqués en développement de systèmes critiques. Une question de principe est posée en vue d'assurer une porte de sortie lorsque le lecteur estime avoir répondu au questionnaire. Il n'y a qu'une seule réponse possible.

> *Industrial Internet of Things (IIoT: "https://en.wikipedia.org/wiki/Internet_of_things#Manufacturing") greatly disrupts industrial IT solutions. Profit margins are further reduced by the effervescence of new devices combined with a lack of standards, which leads to an increased pressure to reduce costs. Even Safety Instrumented Systems (SIS) are victims of this cost reductions influence. In wake of this revelation, compiler diversity delivers unexpected value as a cost-effective SIS defensive measure against Common Cause Failures (CCF).*
>
> *In a master's thesis, I advocate a quantifiable impact of compiler diversity on industrial safety functions, and especially in the presence of safety certified compilers. Using the following 10 questions, you are invited to confirm or refute, using less than one minute per answer, the hypothesis that using different compilers with redundant systems provides a significant leverage effect on a Soft PLC Systematic Capability (SC) level.*
>
> *Do you wish to participate in the study?*
>
> *Done Quit* *First Question*

III.1.1 Répartition des personnes sondées

La première question permet de classifier chaque personne d'après son degré d'homologation en sécurité fonctionnelle. Cette catégorisation sert au dénombrement du public sondé et permet de pondérer la qualité des réponses. Il n'y a qu'une seule réponse possible.

> *First, we need to classify the level of involvement in functional safety.*
>
> *I hold:*
>
> *a) A functional safety professional certificate (CFSP)*
>
> *b) A functional safety expert certificate (CFSE)*
>
> *c) Other (please, specify)*
>
> *d) Do not know / refuse to answer*

III.1.2 Équivalence du concept FIT, SFF et DC pour les logiciels

La seconde question détermine si l'expert ou le professionnel certifié atteste que l'analyse dynamique par classes d'équivalences permet de départager les erreurs détectées non-détectées du logiciel, de telle sorte que des équivalents du FIT, SFF et DC puissent s'en dégager.

Une seule réponse possible.

> *While normally used for hardware, some researchers perform a direct correlation between the software branch coverage and FIT, SFF and DC rates.*
>
> *Do you agree that equivalence classes and input partition testing, combined with boundary and error values are producing credible FIT, SFF and DC rates for the software?*
>
> *YES* *NO*

III.1.3 Équivalence entre la couverture de test et DC pour les logiciels

La troisième question tente de produire une corrélation plus directe entre la couverture de test logiciel et l'indice DC équivalente. Elle demande à la communauté l'estimation d'un DC plausible, en fonction de tests d'injection de fautes.

Une seule réponse possible.

> *While normally used for hardware, some researchers perform a direct correlation between the software branch coverage and FIT, SFF and DC rates.*
>
> *Let's suppose that a dynamic testing tool declares a 95% branch coverage, resulting from its analysis of an embedded software executed with equivalence classes and input partition testing, combined with boundary and error values.*
>
> *Given your experience, which plausible DC level can arise from this 95% branch coverage scenario?*
>
> a) *99% DC*
> b) *90% DC*
> c) *60% DC*
> d) *0% DC*
> e) *Another DC (specify the rate)*

III.1.4 Classement des défaillances

La question suivante demande aux experts et professionnels certifiés d'attester que les bogues de compilateurs se classent parmi les défaillances non-détectées.

> *GCC registered 39,890 bugs and 22,946 bug fixes between August 1999 and October 2015. In the case of LLVM, the community registered 12,842 bugs and 8,452 fixes between October 2003 and October 2015. Looking at these numbers, we can generalize that all compilers enclose defects.*

Given your experience, would you say that compiler bugs systematically fall under the category of "undetected" defects.

YES *NO*

III.1.5 Vulnérabilité aux CCF qui découle des compilateurs

La question suivante demande aux experts et professionnels certifiés d'attester que deux logiciels redondants augmentent leurs vulnérabilités aux CCF lorsqu'un même compilateur les produit tous les deux.

GCC registered 39,890 bugs and 22,946 bug fixes between August 1999 and October 2015. In the case of LLVM, the community registered 12,842 bugs and 8,452 bug fixes between October 2003 and October 2015. Looking at these numbers, we can generalize that all compilers enclose defects.

Given your experience, in a situation where the software of both channels is built from the same source code and uses a single compiler, does the resulting 1oo2 SIS have a higher CCF rate?

YES *NO*

III.1.6 Estimation du CCF pour deux canaux produits par un seul compilateur

La question suivante demande aux experts et professionnels certifiés d'estimer le taux du CCF pour deux logiciels redondants lorsqu'un seul compilateur les produit tous les deux à partir d'une même base de code source.

GCC registered 39,890 bugs and 22,946 bug fixes between August 1999 and October 2015. In the case of LLVM, the community registered 12,842 bugs and 8,452 bug fixes between October 2003 and October 2015.

Looking at these numbers, we can generalize that all compilers enclose defects.

Again, given your experience, what would be a plausible CCF rate for a 1oo2 SIS when the software of both channels is built from the same source code and uses a single compiler?

a) *2% CCF*

b) *10% CCF*

c) *20% CCF*

d) *Greater than 20% CCF (specify the rate)*

III.1.7 Diversité de compilateurs comme mesure de défense contre les CCF

La question suivante demande aux experts et professionnels certifiés d'attester que deux logiciels redondants diminuent leurs vulnérabilités aux CCF lorsque chacun d'eux se produit par un compilateur différent.

In general, we can admit that all compilers enclose defects. However, it remains that each compiler*, with or without a standard library, remains designed, produced and sustained by separate groups of individuals. Competition between compilers tends to distribute bugs differently from group to group.*

Based on your experience, does a 1oo2 SIS have a lower CCF rate when the software of both channels is built from the same source code, but using distinct compilers?

YES *NO*

III.1.8 Estimation du CCF pour deux canaux produits par différents compilateurs

La question suivante demande aux experts et professionnels certifiés d'estimer le taux du CCF pour deux logiciels redondants produits avec deux compilateurs distincts.

> *In general, we can admit that all compilers enclose defects. However, it remains that* each compiler*, with or without a standard library, remains designed, produced and sustained by separate groups of individuals. Competition between compilers tends to distribute bugs differently from group to group.*
>
> *Given your experience, what would be a plausible CCF rate for a 1oo2 SIS, where the software of both channels is built from the same source code, but using different compilers?*
>
> *a) 2% CCF*
>
> *b) 10% CCF*
>
> *c) 20% CCF*
>
> *d) Greater than 20% CCF (specify the rate).*

III.1.9 Effet levier d'un compilateur homologué sur le CCF

Plusieurs compilateurs se présentent avec une homologation SIL4. Cependant, le présent essai se garde de promouvoir un fabricant en particulier.

La question suivante demande aux experts et professionnels certifiés d'attester qu'un compilateur approuvé SIL4 procure un effet de levier sur l'abaissement des vulnérabilités aux CCF.

> *Certified SIL4 compilers perform extensive integrity analysis and instrumentation on translated source code. In general, these compilers receive exhaustive testing which reduces the quantity of bugs susceptible of corrupting the generated software. However, using SIL4 compilers*

does not imply that the produced software will systematically reach any SC layer.

Based on your experience, does a 1oo2 SIS have a lower CCF rate when the software of both channels is built from the same source code, but using distinct compilers, where one of these compilers has a SIL4 certification?

YES *NO*

III.1.10 Estimation du CCF en présence d'un compilateur homologué

La question suivante demande aux experts et professionnels certifiés d'estimer le taux du CCF pour deux logiciels redondants produits avec deux compilateurs distincts.

Certified SIL4 compilers perform extensive integrity analysis and instrumentation on translated source code. In general, these compilers receive exhaustive testing which reduces the quantity of bugs susceptible of corrupting the generated software. However, using SIL4 compilers does not guarantee that the produced software will systematically reach any SIL layer.

Given your experience, what would be a plausible CCF rate for a 1oo2 SIS, where the software of both channels is built from the same source code, but using different compilers, where one of those compilers has a SIL4 certification?

a) *2% CCF*
b) *10% CCF*
c) *20% CCF*
d) *Greater than 20% CCF (specify the rate)*

III.2 Conclusion

Remerciement de conclusion.

Thank you for your attention and for your support.

III.3 Commentaires libres

Chaque question offre un espace de commentaire libre. Voici comment la demande d'opinion se présente :

Please share your thoughts about this question.

À la dernière page, la personne peut également présenter son opinion sur l'ensemble du questionnaire.

Please share your thoughts about this survey.

Annexe IV

Lectures supplémentaires

IV.1 Vérification de compilateurs

Dans son article [45], Tony Hoare promeut la recherche en vérification de compilateurs comme un des grands défis scientifiques de l'humanité. Selon Hoare, ce surpassement se compare à la conquête spatiale dans les années 1960 ou à la cartographie du génome humain. L'auteur appuie son analyse sur un ensemble de critères tirés d'un texte de Jim Gray [62], classifiant la notoriété des domaines de recherches en fonction du degré de maturité requise pour anticiper les progrès futurs, et de l'intérêt de la communauté scientifique long terme.

Les problématiques varient selon la classification d'un langage. La technique de vérification des compilateurs est teintée par ces particularités.

IV.1.1 Problématiques des langages dédiés

Dans leur article [17], Mernik, Heering et Sloane invoquent l'intérêt envers ces langages, pour leur spécialisation autour d'un domaine de connaissance. Cette catégorie offre typiquement moins de fonctionnalités que les langages généralistes, mais procure un mode d'opération fortement aligné sur un champ d'expertise. Selon ces trois auteurs, les langages dédiés rassemblent les caractéristiques suivantes [17] :

- Conformité — aux termes, aux règles, aux symboles et à la sémantique du domaine de compétence.
- Construction et abstraction — des concepts reliés au domaine de compétence.
- Exécutable, ou non — bien que souvent équipé d'une sémantique d'exécution, le langage dédié peut servir de moyen de configuration ou d'intermédiaire de traduction vers un langage généraliste. La caractéristique non exécutable, et spécialement la génération automatique par traducteurs, confèrent aux langages dédiés un potentiel de réutilisation unique.

À l'inverse, dans son article à propos de la norme CEI 61131-3 [40], Mario de Sousa dresse certaines limites. En considérant que cette catégorie de langage porte sur un domaine d'expertise précis [17], les communautés de chercheurs actifs sont moins nombreuses. Les standards qui en découlent comportent souvent des contradictions ou des sémantiques

implicites qui les rendent difficilement vérifiables par une preuve formelle [40]. Chaque fournisseur interprète ces norme et s'en éloigne pour former un contexte de vérification spécifique [40] à sa gamme de produits. De telles contraintes orientent vers d'autres techniques de vérification, moins formelles, liées au contexte issu d'interprétations du standard, plutôt qu'aux spécifications normatives [40].

IV.1.2 Problématiques des langages généralistes

Par analogie avec les langages dédiés, dans leur article [17], Mernik, Heering et Sloane évoquent l'intérêt envers les langages généralistes pour leur d'adaptation à tous les domaines de connaissances. Ils transcendent les domaines de connaissances et forment la base fondamentale des langages dédiés. Avec l'apport de bibliothèques spécialisées, les langages généralistes peuvent même s'y substituer. Selon ces trois auteurs, les langages généralistes rassemblent les caractéristiques suivantes [17] :

- Conformité — à un standard formalisé. L'étude de Gartner [49] permet de comprendre que la maturité des langages généralistes est plutôt inégale, mais ces derniers découlent de comités publics de standardisation ou de normalisation.
- Construction et abstraction — des structures définies par l'utilisateur et pour conduire à l'abstraction du jeu d'instructions attendu par l'unité de traitement ciblée, plutôt qu'un domaine de compétence.
- Exécutable uniquement — pour traduire la spécification de l'utilisateur en instructions optimisées pour l'unité de traitement.

Toujours dans l'article [40], Mario de Sousa explique comment les langages généralistes servent de base fondamentale à la gamme des langages dédiés. D'après l'article [49] de Gartner, les langages généralistes se conforment à des standards ou normes qui se situent à des niveaux de maturités inégales. En fin de compte, la vérification devient un enjeu central pour les deux gammes de langages.

IV.1.3 Vérification de traçabilité

Le standard CEI 61508 exige une traçabilité entre les niveaux de spécifications [51]. La comparaison du programme source et du code exécutable représente un moyen d'assurer une traçabilité ascendante, prouvant la cohérence entre la spécification et le code généré par le compilateur.

Qu'un langage soit dédié ou générique, le succès de la comparaison dépend de la qualité des rétro-compilations. Le code de l'utilisateur et celui produit par la décompilation sont comparés par le moyen de comparaison de graphes de contrôle. La thèse de Touretzky [63] invite plutôt à la prudence, et suggère qu'une reconstruction du code source est impossible sans préserver le code source original, mais l'auteur n'a pu achever son étude ou proposer d'alternatives.

La littérature contient peu d'exemples concrets de rétro-compilation en vue d'une inspection de traçabilité. Les exemples trouvés parmi les articles [43], [64], [65] sont motivés par la restauration du code source, pour des fins d'optimisations ou d'analyse de logiciels malveillants. Seul l'article [47] en fait la démonstration complète.

Langages dédiés

La comparaison entre le code source et le programme exécutable présente l'avantage de s'ajuster au contexte de l'outil et à son interprétation des règles du langage. Que les langages dédiés puissent appliquer des standards incomplets ou présentant un risque d'interprétation [40], n'exclut pas les moyens de vérifications susceptibles d'apporter l'homologation de sécurité [8], [12].

Une des rares démonstrations d'une comparaison du code binaire avec la spécification, est décrite dans l'article [65], produit par les chercheurs Mendis, Bosboom et *al*. Les auteurs ont produit une rétro-compilation en traduisant vers le langage Helium, un programme exécutable de filtres et stencils Photoshop optimisés pour les processeurs x86. Helium fait partie de la gamme des langages dédiés. Il permet de produire des algorithmes de filtres graphiques. Les auteurs reproduisent la source en vue d'améliorer les filtres Photoshop.

Langages généralistes

Selon la thèse de [63], les compilateurs pour langages généralistes produisent du code binaire optimisé difficilement transposable en langage de haut niveau. Dans leur recherche [43], les auteurs Obermann & Börcsök font la même déduction, mais proposent l'injection de métadonnées pour faciliter le processus de décompilation. Ces derniers admettent que le processus de compilation engendre une perte d'information, ce qui bloque toute possibilité de reconstruction du code source propre à une comparaison. La perte d'information doit être compensée par l'ajout de métadonnées utiles à une rétro-compilation de qualité. Leur article [43] ne présente cependant pas d'expérimentation concrète permettant de corroborer leurs propos.

À cet effet, les volumes de la norme CEI 61508 font référence à une démonstration faite en 1993 par Pavey et Winsborrow [47], d'une technique qui consiste à comparer le programme source écrit en PL/M-86 avec le code exécutable binaire extrait d'une puce PROM [12, Chap. C.4.4.1]. L'expérimentation met en œuvre la décompilation d'un programme exécutable vers le langage PL/M-86. Ce langage n'apparaît pas dans l'échelle de maturité de Gartner [49] et son histoire le présente comme un dérivé du PL/1, lui-même considéré en fin de vie [49]. L'expérimentation reste toutefois valide, même s'il s'agit d'un langage généraliste en fin de vie.

IV.1.4 Vérification par preuve formelle

Les techniques formelles restent toutefois les approches privilégiées pour obtenir une homologation de sécurité fonctionnelle de niveau supérieure (SIL4). Les deux gammes de langages présentent toutefois des besoins spécifiques.

Langages dédiés

La vérification formelle des compilateurs pour langages dédiés reste accessible. Cependant, les démonstrations consistent le plus souvent à prouver la cohérence du langage lui-même, en vue de pallier les inconvénients décrits par Mario de Sousa [40].

Dans leur article [66], les auteurs Liu, Höglund, Khan, et Porres démontrent une technique de vérification formelle en vue de prouver la cohérence d'un langage dédié. Ils se penchent sur l'intégrité sémantique du langage, plutôt que sur le programme exécutable produit. Ils

traduisent des diagrammes UML en ontologies qu'ils soumettent ensuite à des analyseurs de théorèmes OWL2 nommés « Pellet » et « HermiT ». L'emploi de deux analyseurs permet aux auteurs de valider leurs résultats. Pour la démonstration, ils appliquent leur technique sur 286 modèles tirés de la banque de données « Atlantic Metamodel Zoo » [67] et réussissent à prouver la cohérence de 279 modèles.

Avec leur article [68], l'équipe Semeráth, Barta, Horváth, Szatmári, et Varrós poussent la technique plus loin en combinant les règles OCL (*Object Constraint Language*) aux ontologies soumises aux analyseurs de théorèmes. Ils exécutent l'expérimentation avec « Z3 », un analyseur de type SMT (*Satisfiability modulo theories*), et avec « Alloy », un analyseur de type SAT (*Boolean satisfiability problem*). Les auteurs proposent une technique plus complexe que celle décrite dans [66], mais peut s'appliquer sur des langages comportant des points d'extensibilités. Ils ont appliqué leurs techniques à plusieurs langages employés en aérospatiale [69], [70].

Les auteurs James et Roggenbach abordent également la vérification de modèle avec leur article [71]. L'expérimentation porte sur la vérification d'un langage dédié à la modélisation de plans pour chemins de fers. Cette équipe cherche à élaborer une technique accessible aux ingénieurs de terrain ; leur approche se veut moins théorique que [66] et [68]. Leur technique consiste à combiner un modèle UML avec des spécifications décrites en langage naturel, et les soumettre à un analyseur de théorèmes nommé « CASL ». L'expérimentation porte sur plusieurs projets en collaboration avec la société australienne « Invensys Rail », pour l'implémentation du langage dédiés proposé dans la thèse de James [72].

Langages généralistes

Les auteurs Jianzhou Zhao et *al* [54] proposent une stratégie de vérification formelle pour l'infrastructure de compilation LLVM. La technique consiste à insérer des appels aux services de vérification dans certains fils de génération SSA (« *single static assignment* ») qui servent dans plusieurs stratégies d'optimisation. Un peu comme le propose Leroy dans son article [42], chaque service implémente des preuves formelles sous la forme d'invariants réalisés en langage Coq. Le générateur conserve les optimisations vérifiées et neutralise les autres.

Dans [55], les auteurs Carré et Garnsworthy présentent ADA comme un des langages conçus pour le développement de systèmes critiques. Le sous-ensemble SPARK renferme des types et fonctions de sécurité ainsi qu'un langage d'annotations qui assure la prédictibilité par vérification formelle des programmes ADA. Dernièrement, l'article de Courtieu, Aponte et *al* [56] décrit plus en détail le fonctionnement des annotations permettant l'ajout de prédicats et leur interaction avec le programme à l'exécution. Le programme peut s'auto-vérifier constamment.

Dans [50], Croker s'engage dans une comparaison, entre un compilateur C et C++ conçu pour la production de logiciels vérifiés par méthodes formelles, et l'offre SPARK intégrée au langage ADA. Contrairement aux techniques présentés dans les articles [42], [54], Croker étudie un dérivé de C++ affermi par des annotations pour exercer la vérification sémantique à la manière du paradigme « Verified Design-by-contract » [57]. L'expérimentation porte sur le compilateur MISRA-C : 2012, enrichi avec des règles qui prohibent l'usage des constructions du langage C plus vulnérables [58], ainsi que d'une notation de règles contractuelles, semblable à celle offerte avec SPARK. Un analyseur de théorèmes vérifie la cohérence des contrats. Le code généré peut se valider constamment en cours d'exécution. Ces règles deviennent inactives pour assurer la compatibilité avec tout compilateur C standard.