GROUPE SERMA TECHNOLOGIES

SEMINAIRE CAPTRONIC - 24 mai 2012

Présentation du groupe Serma Technologies

SERMA INGENIERIE

GROUPE SERMA TECHNOLOGIES

SERMA TECHNOLOGIES | SERMA INGENIERIE | SERMA GmbH | ID MOS

Bordeaux

Toulouse

Grenoble

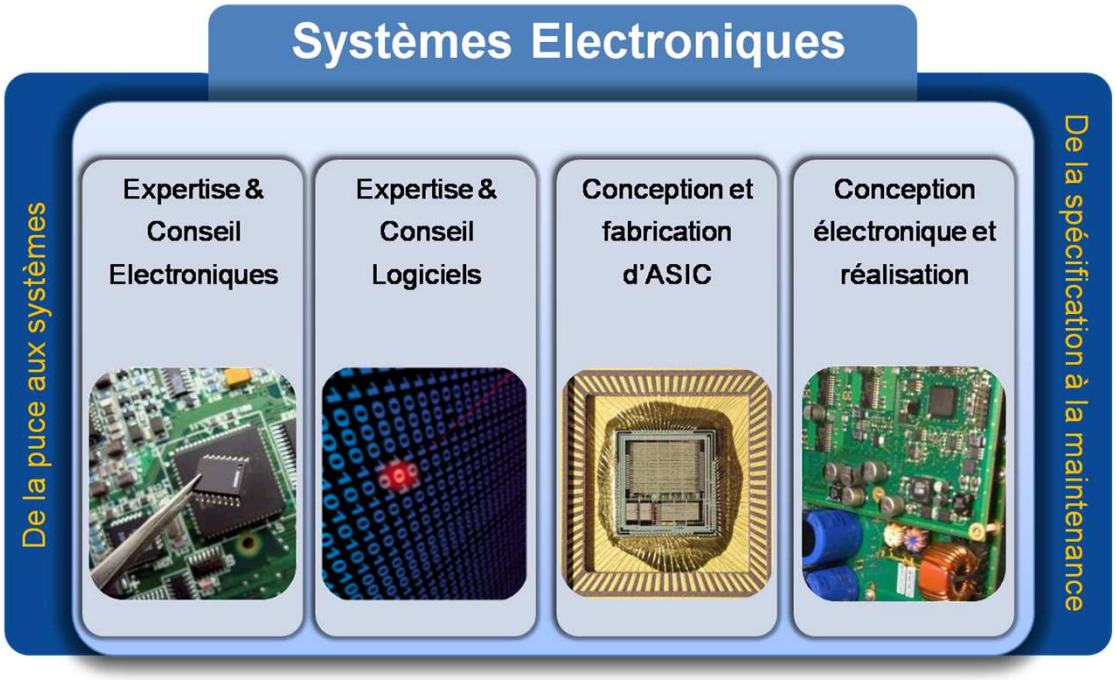
Paris

Nüremberg

Berlin

650 pers.

66 M€ CA



ISO 9001 V2008

EN 9100

PART 21G

PART 145

FAR 145

Membre de 6 “Pôles de compétitivité” dont Aerospace Valley, Minalogic, Mov’eo, System@tic

Nos activités Sûreté de Fonctionnement Système, Matériel , Logiciel

- **Une équipe de près de 20 personnes**
- **Maîtrise des normes applicables à différents domaines :**

CEI 61508, CEI 62061, CEI 61511, ISO 26262, EN 50126, EN 50128, EN 50129, CEI 60880, CEI 61513, CEI 62304, DO 178B,…

- **Des prestations adaptées :**

- **Des experts reconnus (CERTIFER, TÜV Rheinland)**

- **Maîtrise des méthodes et outils de la SdF**
  - ✓ Analyse fonctionnelle (APTE, SaDT, SaRT…),
  - ✓ Analyse Préliminaire des Risques,
  - ✓ Analyse par Arbres de Défaillances
  - ✓ Calcul de fiabilité prévisionnelle (RDF 2000, FIDES, MIL HDBK 217…)
  - ✓ AMDEC (Analyse des Modes de Défaillances, de leurs Effets et leur Criticité)
  - ✓ Critical Process (analyse physique d'une carte ou d'un système qui permet d'avoir une vue globale sur la maîtrise des procédés de l'assembleur)
  - ✓ Estimation / allocation des niveaux de SIL (Safety Integrity Level)
  - ✓ Lecture critique de code (C, C++, JAVA, VHDL)
  - ✓ Traçabilité code / spécification
  - ✓ Analyse des règles de codage, d'architecture et des données
  - ✓ AMDE de niveau logiciel (AEEL)

Un environnement industriel exigeant

SERMA INGENIERIE

Aéronautique

Ferroviaire

Industrie

Spatial

Automobile

Médical

Défense

Objectifs du séminaire

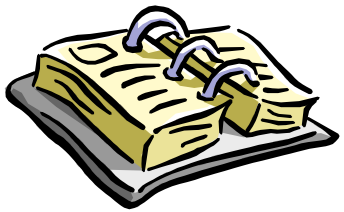
- L'objectif de ce séminaire est de répondre aux interrogations des concepteurs, développeurs, chefs de projet, directeur technique souhaitant :
  - Intégrer du logiciel dans des applications « critiques » d'un point de vue sécurité, disponibilité…
  - Améliorer le niveau de « fiabilité » de vos logiciels

- En vous présentant :
  - Un tour d'horizon des normes existantes dans ce domaine
  - Des éléments utiles pour spécifier efficacement et définir l'architecture des systèmes dits « Électriques / Électroniques / Électroniques Programmables (E/E/EP) » et des logiciels associés
  - Les méthodes et techniques à mettre en œuvre pour concevoir et apporter l'assurance de la sûreté de fonctionnement des logiciels

- Ce séminaire est basé sur notre retour d'expérience de plus de 15 ans dans l'évaluation et l'expertise de systèmes critiques à forte composante logicielle

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP
- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel
- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)
- Conclusion



SERMA INGENIERIE

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP

- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel

- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)

- Conclusion

Quelques définitions

- Définition de la Sûreté de Fonctionnement :

«Aptitude d'une entité à satisfaire une ou plusieurs fonctions requises dans des conditions données» - (A. Villemeur)

- Définitions des attributs de la SdF :
  - La **F**iabilité (Reliability) : aptitude d'un système à accomplir sa mission dans les conditions données d'utilisation
  - La **M**aintenabilité (Maintenability) : aptitude d'un système à être entretenu ou remis en marche
  - La **D**isponibilité (Availability) : aptitude d'un système à fonctionner lorsqu'on le sollicite
  - La **S**écurité (Safety) : aptitude d'un système à respecter l'utilisateur et son environnement

➔ FMDS (en français) - RAMS (en anglais)

Quelles applications ?

- **Quelques exemples d'application :**

  – Système de contrôle/commande d'une centrale nucléaire
  – Système de gestion de la signalisation ferroviaire
  – Régulateur de vitesse de voiture
  – Système de sécurité incendie de tunnel ferroviaire

  Mais aussi …

  – Système de régulation de débit sanguin d'un cœur artificielle
  – Capteur de position
  – Système de traction dans un train
  – …

Pourquoi prendre en compte la SdF dans les développements des logiciels ?

- Pour réaliser un Produit sain c'est-à-dire fiable, maintenable et sûr de fonctionnement dans le temps
- Pour accéder à de nouveaux marchés
- Pour répondre à l'existence de marchés où la certification / évaluation de conformité est obligatoire

☛ Évaluation / Certification SdF = indicateur pour le client de la « qualité » du produit

Similitudes et Différences entre le matériel et le logiciel (1/2)

Aspects similaires avec le matériel	
• Analogies dans l'**architecture** (modularité, redondance, …)	• Exigences de **sûreté** transposables
• **Traitements de tolérance aux fautes** équivalents	• Allocation des niveaux de **criticité**
• Composants **réutilisables, paramétrables**	• **Modes de défaillance, modes communs de défaillance**
• Méthodes de **modélisation**	• Tests de **robustesse**, tests par **injection de fautes**
• **Défaillances systématiques** : conséquence d'erreurs commises durant le développement du système, du matériel, du logiciel.	

Similitudes et Différences entre le matériel et le logiciel (2/2)

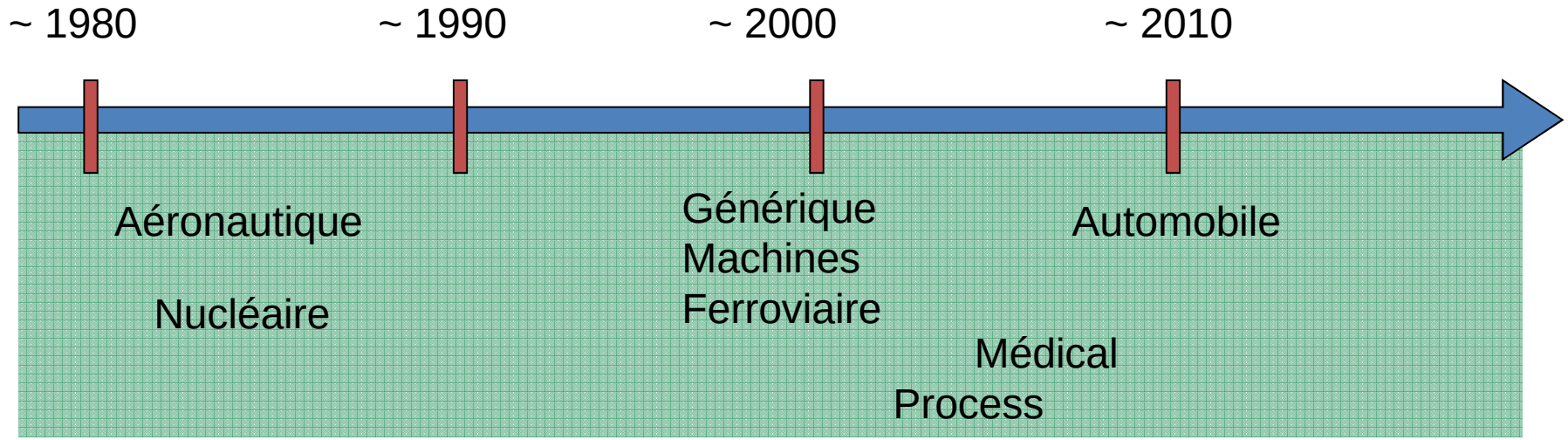
SERMA INGENIERIE

Aspects différents avec le matériel	
• **Pas de vieillissement** du logiciel	• **Cycle de développement en V** plus complet
• **Maintenance évolutive** plus aisée, moins coûteuse avec le logiciel	• Peu de retours d'expérience, de statistiques, de bases de données de fiabilité, …
• Capacité à réaliser des traitements plus "**évolués**", plus "**complexes**" avec le logiciel	• Démarche d'**évaluation** spécifique
• **Pas de défaillances aléatoires** du logiciel	• **Documentation** technique particulière

Des outils pour faciliter ces développements : les normes

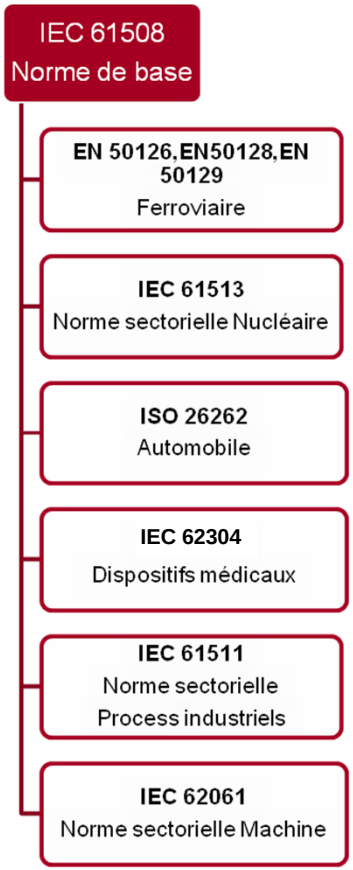
- Utilisation de plus en plus importante du logiciel pour assurer ou participer à des fonctions critiques pouvant impacter la sécurité

➔ Développement de référentiels intégrant la composante logicielle dans les différents secteurs industriels



La normalisation

- La normalisation



- La norme CEI 61 508 présente une approche générique de toutes les activités liées au cycle de vie de sécurité des systèmes E/E/EP
- Objectif principal : permettre l'élaboration de Normes par secteur d'application pour les systèmes E/E/EP relatifs à la sécurité
- Objectif induit : permettre le développement de systèmes E/E/EP relatifs à la sécurité en l'absence de Norme
- D'autres normes existent dans le domaine aéronautique (DO178)

Exemple de norme de Sdf Logiciel (1/3)

SERMA INGENIERIE

Domaine Aéronautique

- Norme DO 178C : "Considérations sur le logiciel en vue de la certification des systèmes et équipements bord "(mai 2012)

  – s'adresse à des logiciels destinés à des systèmes et équipements embarqués à bord des aéronefs civils
  – présente les processus du cycle de vie du logiciel (spécification, conception, codage, intégration, vérification, assurance qualité, gestion de configuration) en terme d'objectifs, d'activités pour les atteindre (documentation, tests), de preuves pour montrer qu'ils sont atteints
  – ne constitue pas un guide de développement, mais un guide de certification.

Exemple de norme de Sdf Logiciel (2/3)

SERMA INGENIERIE

Tous les domaines (norme générique)

- Norme CEI 61508 (parties 1 à 7) : Sécurité fonctionnelle des systèmes électriques / électroniques / électroniques programmables (E/E/EP) relatifs à la sécurité (Janvier 2011)
  - Composée de 7 parties, le logiciel est couvert par les parties 3 et 7.
  - Décrit les différentes étapes du cycle de vie du logiciel :
    1. Spécification des Exigences de Sécurité du Logiciel
    2. Planification de la validation du logiciel
    3. Conception et développement du logiciel
    4. Intégration Électronique programmable (matériel/logiciel)
    5. Exploitation et maintenance du logiciel
    6. Validation de la sécurité du logiciel
    7. Modification du logiciel
    8. Vérification du logiciel
    9. Évaluation de la sécurité fonctionnelle

Exemple de norme de Sdf Logiciel (3/3)

Domaine Ferroviaire (norme sectorielle, fille de la CEI 61508)

- Norme CENELEC 50128 : "Systèmes de signalisation, de télécommunication et de traitement - Logiciels pour systèmes de commande et de protection ferroviaire" (2011)

  – s'adresse à des logiciels de **systèmes ferroviaires** de **protection** et de **contrôle-commande**
  – décrit les différentes étapes du cycle de vie du logiciel :
    - Spécification de Besoin Logiciel et Architecture du Logiciel
    - Conception, développement et tests du logiciel
    - Intégration du logiciel sur le matériel cible
    - Validation du logiciel
    - Intégration du logiciel dans le système programmable global
    - Maintenance du logiciel.

Normes et applications

- **Domaine d'application de ces normes**

  - Pour tout système E/E/EP dit « de sécurité » si une défaillance de ce système peut conduire à une situation où des **personnes** ou **l'environnement** pourraient être exposés à un **danger**

  - Pour tout système E/E/EP de **commande**, de **protection** ou de **surveillance**

  - Plus globalement pour mettre en œuvre une **démarche structurée** destinée à « **fiabiliser** » les logiciels

Les fonctions de sécurité

Les fonctions de sécurité sont des fonctions conçues pour la **protection de la sécurité** de **l'environnement** (donc des personnes et des biens), pour la mise dans un état sûr et la remontée d'informations.

Exemple : fonction de freinage d'urgence en cas de survitesse sur un train.

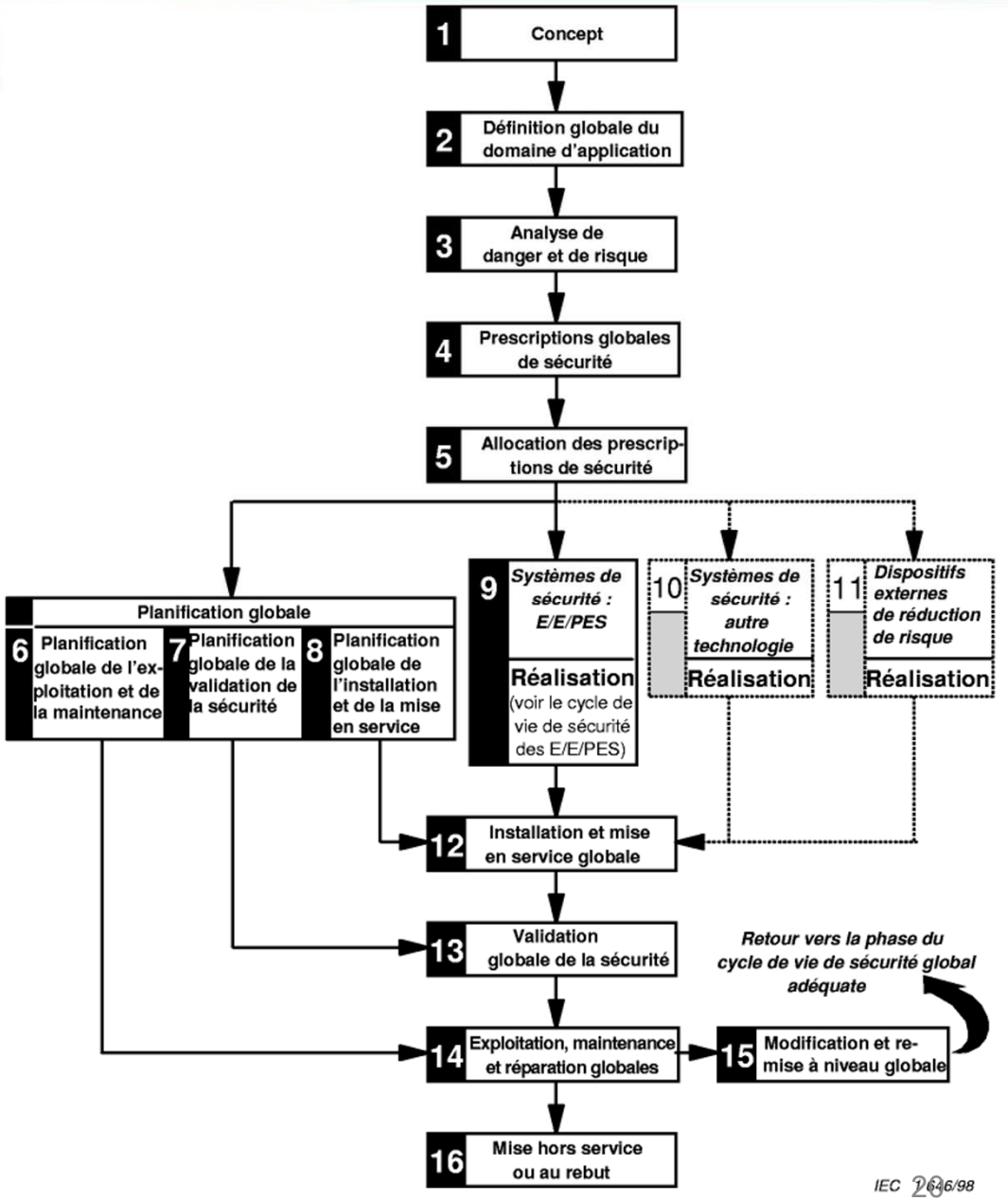
Ces fonctions sont réalisées par des **systèmes** qui **peuvent défaillir**
=> Fonctions participant à la sûreté ou de tolérance aux fautes

- Fonctions de **détection des fautes** du système (équipements, capteurs, actionneurs...), du matériel supportant le logiciel (réseau, microprocesseur, mémoires, disque...), du logiciel (spécifications, code, séquencement, logiciel standard...)
- Fonctions **de contrôle** (des limites des données, des relations d'ordre entre les données, des erreurs de transmission, des erreurs humaines...)
- Fonctions **de suivi** et de **diagnostic des erreurs** détectées (gestionnaire d'alarmes, historiques des données, tampons de fautes...).

Concepts portés par la norme générique CEI 61 508

- **Le cycle de vie de la sécurité**

  Cadre pour structurer les activités concernant la spécification, la conception, l'intégration, la validation, la mise en service, l'exploitation, la maintenance et la mise hors service des systèmes de sécurité E/E/EP.



Le concept de SIL (1/2) de la norme CEI 61508

- **Ce concept représente un niveau de criticité :**

- **Il définit un système de classement des performances attendues en fonction du niveau des exigences émises**

  Niveau d'intégrité de la sécurité : SIL de 1 à 4 « probabilité qu'un système relatif à la sécurité exécute de manière satisfaisante les fonctions de sécurité requises ».

- **Prescription de méthodes et outils à employer en fonction du niveau de SIL**

Cas d'autres normes :

- La norme CENELEC EN 50128 :
  Niveau d'intégrité de sécurité du logiciel : déterminé d'après le niveau de risque associé à l'utilisation du logiciel dans le système et le SIL du système.
  Du plus faible au plus élevé : *SIL 0 à 4 (notion SSIL pour le logiciel)*

- La norme DO 178 B :
  Niveau logiciel : fondé sur la contribution du logiciel à d'éventuelles conditions de panne.
  Du plus faible au plus élevé : *Niveau E à A*

Le concept de SIL (2/2)

La norme CEI 61508 propose 4 niveaux d'intégrité de sécurité (**S**afety **I**ntegrity **L**evel).

Intervalle de criticité associé à chaque niveau :

Niveau d'intégrité de sécurité	Probabilité de défaillance à l'exécution d'une fonction spécifique **sur demande**	Probabilité de défaillance dangereuse par heure **en service continu ou en demande élevée**
4	$\geq 10^{-5}$ à $<10^{-4}$	$\geq 10^{-9}$ à $<10^{-8}$
3	$\geq 10^{-4}$ à $<10^{-3}$	$\geq 10^{-8}$ à $<10^{-7}$
2	$\geq 10^{-3}$ à $<10^{-2}$	$\geq 10^{-7}$ à $<10^{-6}$ *
1	$\geq 10^{-2}$ à $<10^{-1}$	$\geq 10^{-6}$ à $<10^{-5}$

10^{-6*} = 1 défaillance dangereuse tous les 1 000 000 heures (114 ans)

Les acteurs de la SdF du logiciel (1/2)

SERMA INGENIERIE

- **Le concepteur/réalisateur** :
  est responsable des activités de conception du logiciel.
- **Le chargé de vérification** :
  est responsable des activités de vérification des activités de développement (revues, traces, LCC…) et de tests (unitaires, d'intégration)
- **Le chargé de validation** :
  est responsable des activités de validation (audits des processus, tests des exigences, analyses)
- **Le chargé d'évaluation** :
  évalue la sûreté fonctionnelle du logiciel à travers des relectures critiques de documents et des analyses de sûreté (AMDEC, arbre des causes…)
- **L'autorité de sûreté / tutelle** :
  a pour rôle de certifier que le logiciel est apte au service, à partir des analyses et des résultats de tests qui lui sont fournis, et par rapport aux exigences réglementaires internationales

Les acteurs de la SdF du logiciel (2/2)

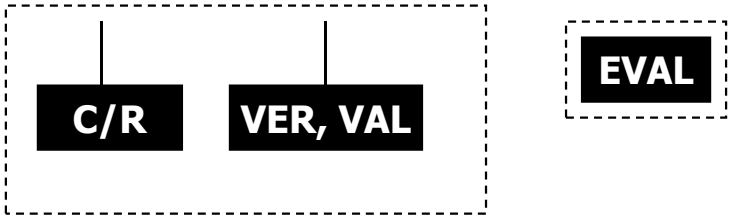
SERMA INGENIERIE

➢ **Le principe d'indépendance** (schémas de la CENELEC 50128)

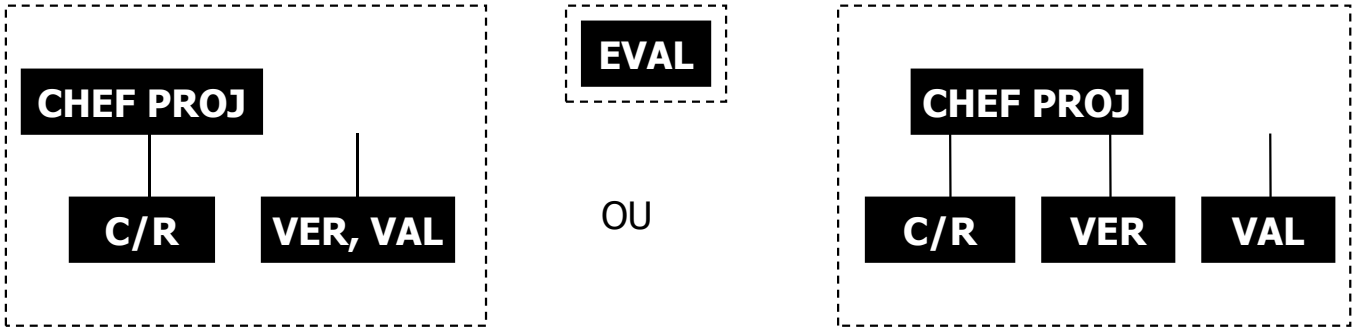
C/R : Concepteur/ Réalisateur
VER : Chargé de Vérification
VAL : Chargé de Validation
EVAL : Chargé d'évaluation
CHEF PROJ : Chef de Projet

SIL0

SIL1&2



SIL3&4



AVOIR UNE APPROCHE SYSTEME

- Déduire, des exigences de sûreté système, les exigences de sûreté du logiciel :
  - Définition globale du domaine d'application :
    - l'autorité de sûreté définit les règles de sécurité
    - l'exploitant établit les objectifs de sûreté en terme de disponibilité, maintenabilité, robustesse, …
  - Analyse de Dangers et de Risques du système :
    - identification des dangers, les événements redoutés
    - déduction des fonctions de sûreté et leur criticité (prescriptions globales de sécurité)
    - répartition des fonctions de sûreté dans les différents équipements (allocation)
    - définition de l'architecture de chaque équipement (E/E/PES)
  - Analyse de Dangers et de Risques de chaque équipement :
    - répartition des fonctions de sûreté entre le matériel et le logiciel
    - définition du niveau d'intégrité de sûreté du logiciel (SIL)
    - détermination des fonctions logicielles de sûreté
    - liste des événements redoutés du logiciel

➔ EXIGENCES DU LOGICIEL

Mise en œuvre de la Sûreté de Fonctionnement du logiciel

Modèle de cycle de vie du logiciel

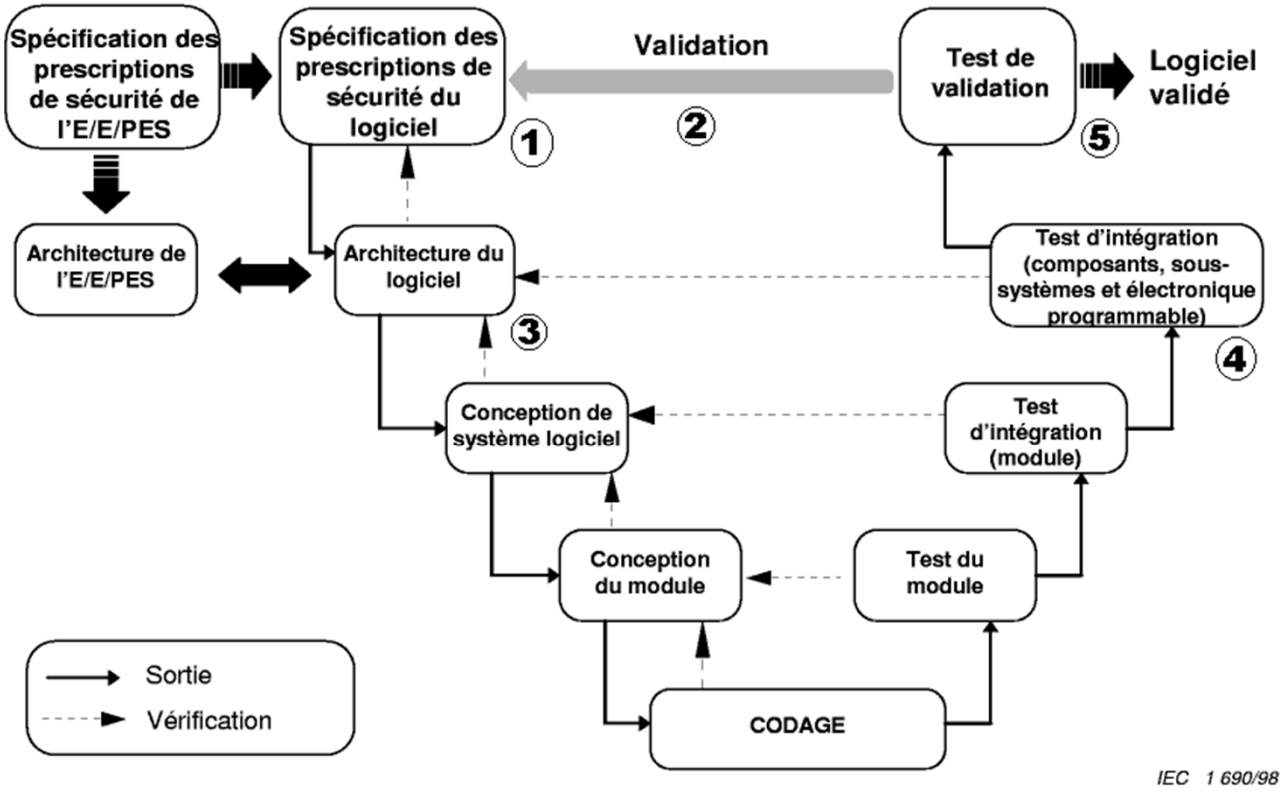


Figure 5 – Intégrité de sécurité du logiciel et cycle de vie de développement (modèle en V)

ET CONCRETEMENT QUE FAIT-ON ?

Quel que soit le niveau d'intégrité de sûreté du logiciel, il faut un système de gestion de la qualité du logiciel.

1. Planification
2. Conception
   - la spécification
   - l'architecture
   - les outils et langage de programmation
   - la conception détaillée
3. Vérification et Validation (V&V)
   - la vérification
   - les tests
   - la validation
4. Évaluation

Plan Qualité Logiciel

Objectifs :

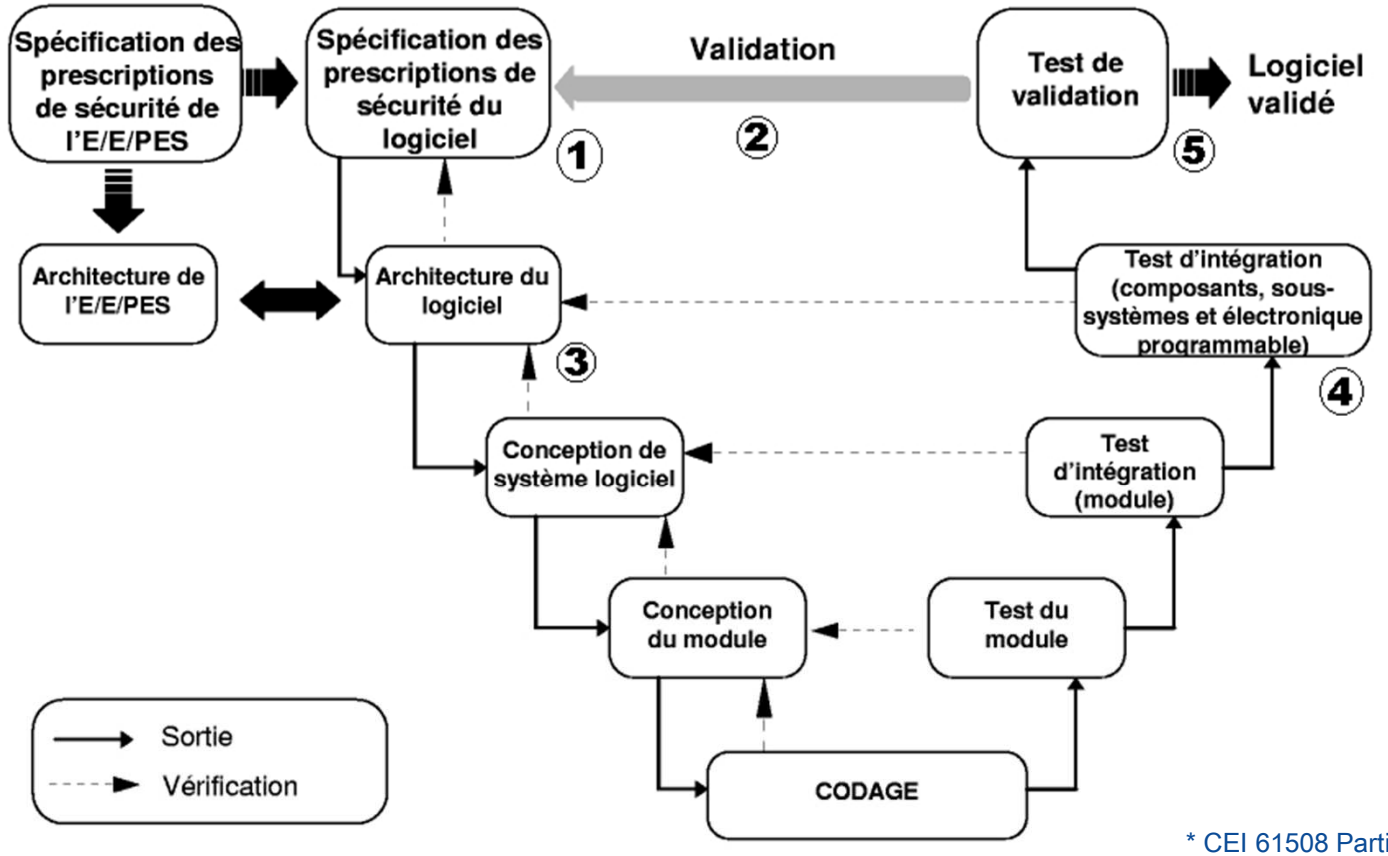
- Présenter les informations relatives à
  - **l'organisation**
  - **la méthodologie**
  - **l'assurance qualité** du logiciel développé (description des phases)
  - **La documentation, la gestion de configuration**
  - **Les activités de vérification du logiciel**
  - La sous-traitance
  - Les **normes de codage** à utiliser
  - Les **méthodes et outils** mis en place pour chaque phase et la **justification du choix** de ces outils
  - …
- C'est un document de référence pour l'équipe de développement logiciel.

SERMA INGENIERIE

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP
- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel
- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)
- Conclusion

Cycle de vie de développement*



* CEI 61508 Partie 3

Figure 5 – Intégrité de sécurité du logiciel et cycle de vie de développement (modèle en V)

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP
- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel
- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)
- Conclusion

La spécification du système E/E/EP* 1/2

SERMA INGENIERIE

Objectifs :

- Prescriptions globales de sécurité :
  - Spécifier les fonctions de commande, de protection ou de service de l'EUC
  - Spécifier les fonctions de sécurité qui permettent de réduire le risque de l'EUC jusqu'à un niveau acceptable
  - Faire les analyses fonctionnelles et dysfonctionnelles du système E/E/EP (APR, AF, AMDEC, AdD, …) -> Spécifier les fonctions participant à la sécurité de l'E/E/EP (diagnostic, autotests, tests en ligne, détection de panne des équipements en interface, …).
- Définir les modes d'exploitation (init, réglage, paramétrage, démarrage, régime établi, ...)

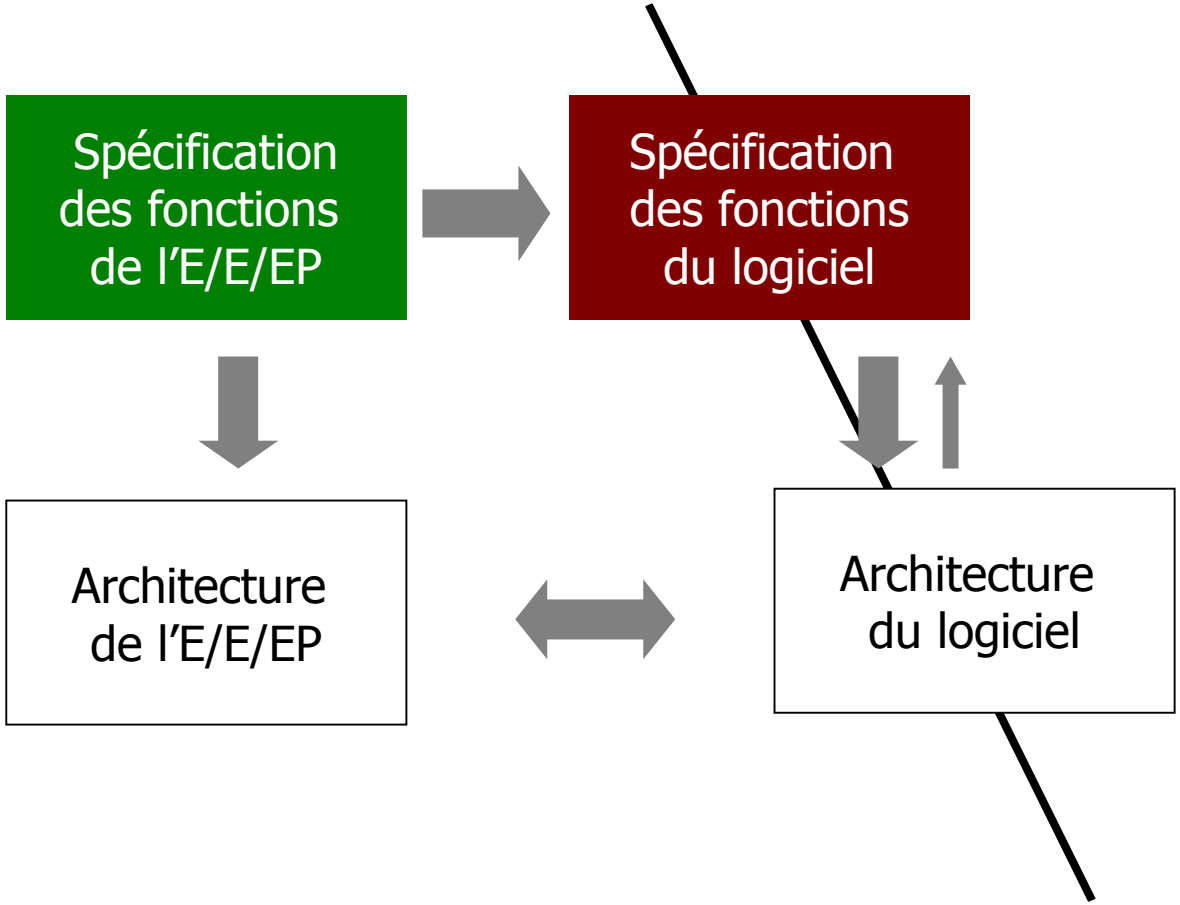
* (CEI 61 508-1 articles 7.5 et 7.6)

La spécification du système E/E/EP* 2/2

Objectifs :

- Définir les exigences techniques (performance, maintenabilité, disponibilité, sécurité, ...)
- Définir les Évènements Redoutés
- Allocation des Prescriptions de sécurité : Allouer les SIL aux fonctions définies
- Description des interfaces (opérateur, externe, interne au niveau système)

… à partir des prescriptions des fonctions du Système EP



Objectifs de la spécification du logiciel (1/2)

- Le document de *Spécification du Logiciel* définit l'ensemble complet **des exigences pour le logiciel respectant les exigences du système**. Il sert de document **destiné aux Développeurs** qui ainsi **n'ont pas besoin des autres documents système**.

- La Spécification des Exigences du Logiciel permet de spécifier en détail les **exigences fonctionnelles, opérationnelles, d'interface** d'un produit logiciel sans préjuger de la solution informatique susceptible de répondre à ces exigences.

Objectifs de la spécification du logiciel (2/2)

- La Spécification des Exigences du Logiciel doit être auto-suffisante pour **permettre de spécifier les tests de validation** de type :
  - tests des valeurs aux limites (informatique)
  - tests de classes d'équivalence et partition d'entrée
  - tests de performance
  - tests des temps de réponse
  - tests des contraintes de place mémoire

Les grandes parties de la spécification du logiciel

- La spécification du logiciel est composé d' :
  - Une **présentation générale du logiciel** faisant le lien avec le système qui l'entoure (configuration, contrainte)
  - Une **description des différents modes de fonctionnement** du logiciel (init, mode nominal, mode dégradé, veille)
  - Une **architecture fonctionnelle du logiciel** qui définit ce que le logiciel doit faire d'un point de vue fonctionnel
  - Une **identification et une description des fonctions de sécurité** ou y participant (détection, vérification, protection, ...)
  - Une **traçabilité des exigences** définies avec les documents amonts

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP
- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel
- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)
- Conclusion

L'architecture du système E/E/EP*

Objectifs :

- Prescriptions d'intégrité de sécurité du matériel :
  - Prescriptions vis-à-vis des défaillances aléatoires
    - Contraintes architecturales, SIL du matériel et couverture des diagnostics

Tableau 3 – Intégrité de sécurité du matériel: contraintes architecturales sur les sous-systèmes relatifs à la sécurité de type B

Proportion de défaillances en sécurité	Tolérance aux anomalies matérielles (voir note 2)		
	0	1	2
< 60 %	Non autorisé	SIL1	SIL2
60 % – < 90 %	SIL1	SIL2	SIL3
90 % – < 99 %	SIL2	SIL3	SIL4
≥ 99 %	SIL3	SIL4	SIL4

NOTE 1 Voir 7.4.3.1.1 à 7.4.3.1.4 pour plus de détails quant à l'interprétation de ce tableau.

NOTE 2 Une tolérance aux anomalies du matériel N signifie que N + 1 anomalies sont susceptibles de provoquer la perte de la fonction de sécurité.

NOTE 3 Voir l'annexe C pour les détails concernant le calcul de la proportion de défaillances en sécurité.

  - Prescriptions vis-à-vis des défaillances systématiques
  - Prescriptions sur le comportement du système sur détection d'un défaut

* CEI 61 508-2 article 7.4

Architecture matérielle :

- **Redondance**
  - des lignes de communication : informations transférées plusieurs fois en séquence (efficace contre les défaillances transitoires)
  - des données de sécurité
    - Mirroring dans un système décentralisé
    - Double RAM
    - Test RAM (redondance à un bit : test de parité)

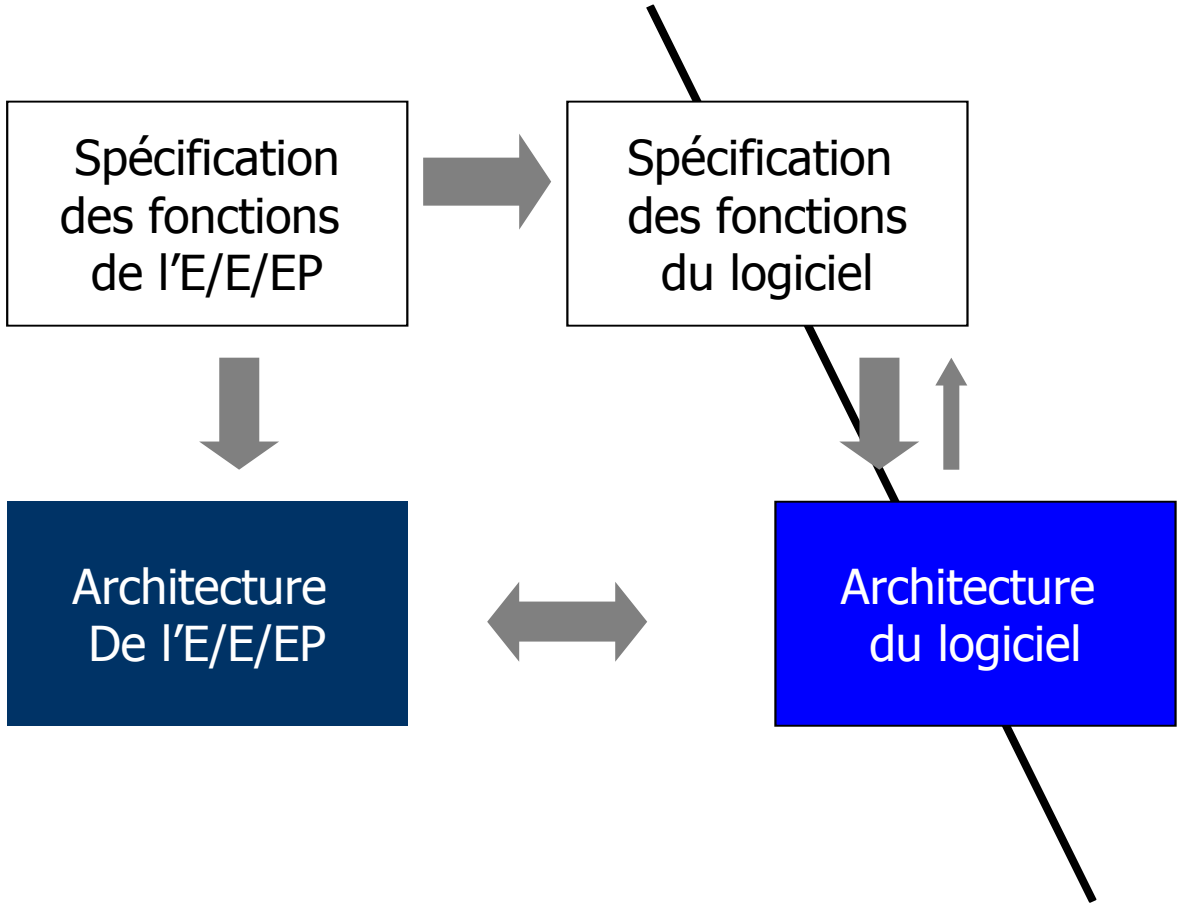
- **Cloisonnement** des sous-systèmes pour réduire les interactions possibles et réduire la possibilité de propagation d'une panne

- **Ségrégation** des sous-systèmes de niveaux de criticité différents (éviter la "pollution" des composants critiques par les composants moins critiques)

Gestion de la mémoire
Memory Management Unit (MMU) et ECC

- MMU : Interface matérielle qui gère les accès mémoire.
- Intégrée au processeur ou bien située entre le processeur et la RAM.
- Fonctions gérées par la MMU:
  - Segmentation
    - Transcription des adresses logiques en adresses linéaires (permet à chaque programme d'avoir une plage d'adresse qui commence à 0).
  - Pagination
    - Mémoire est divisée en « pages » dont les zones sont accessible via des adresses « physiques ». La pagination est la transcription (par la MMU) d'une adresse linéaire en adresse physique
  - Protection mémoire
    - Empêche un programme d'accéder à une plage qui ne lui est pas attribuée.
    - Si accès mémoire non autorisé, la MMU lève une exception récupérée par l'OS, puis l'OS stoppe le programme qui violait l'adresse mémoire.
- ECC (Error Correction Coding) : mémoire possédant plusieurs bits dédiés à la détection et à la correction d'erreurs (appelés bits de contrôle).

…à partir des prescriptions de l'architecture du Système EP



Objectifs de l'Architecture du Logiciel (1/2)

- Le document *d'Architecture du Logiciel* définit l'ensemble complet **des exigences du logiciel respectant les exigences d'utilisation du matériel** EP. Il sert de **document destiné aux Développeurs** qui ainsi **n'ont pas besoin des autres documents système.**

- Le document d'Architecture du Logiciel permet de présenter en détail les **interactions matériel / logiciel**, les **contraintes d'implémentation** du logiciel dans l'EP, les stratégies de **tolérance aux fautes.**

Objectifs de l'Architecture du Logiciel (2/2)

- Le document d'Architecture du Logiciel doit être auto-suffisant pour **permettre de spécifier les tests d'intégration matériel/logiciel** de type :
  - tests des valeurs aux limites (informatique) des données d'interface
  - tests des classes d'équivalence et partition des entrées et des sorties des composants ou sous-systèmes du logiciel

Les grandes parties de l'architecture du logiciel

- L'architecture du logiciel est composé d' :
  - Une description des **principaux composants et sous-systèmes du logiciel** (système d'exploitation, langage, contrainte temporelle / mémoire, ...)
  - Une description des **interfaces avec le matériel** et/ou avec **d'autres logiciels**
  - Une description des **stratégies de tolérances aux fautes** (programmation défensive, redondance, test mémoire, protection de données critiques)
  - Une **traçabilité des exigences** définies avec le document amont

SERMA INGENIERIE

Les Bonnes Pratiques de l'Architecture du Logiciel

Séminaire CapTronic
24 mai 2012

Erreurs d'Architecture

- Le non **respect de contraintes** technologiques (utilisation de ressources, partage de ressources…)
- la mauvaise **gestion des tâches** et de **l'ordonnancement** (non respect des échéances à l'exécution) et des **données partagées** entre tâches
- la non-robustesse vis-à-vis de **l'environnement EP** (charge, microcoupures, environnement électromagnétique…)
- la **chaîne de production de l'exécutable** (compilateur, linkeur, run time…) non "validée" et non "certifiée"

Règles d'Architecture du logiciel (1/3)

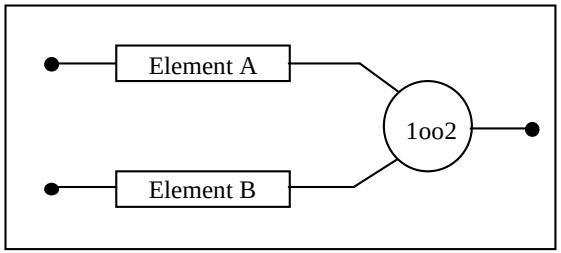
Architecture du logiciel, **issue de contraintes et de choix sur l'architecture matérielle** :

- **Redondance** des chaînes de traitement (une « chaîne de traitement» correspond à une carte électronique avec son logiciel - unité maître/esclave)
- **Synchronisation** des chaînes de traitement
- **Programmation diversifiée** (des fonctions, de l'équipe de développement, des compilateurs, des jeux d'instructions…)
- **Ségrégation** des constituants logiciels de niveaux de criticité différents (éviter la "pollution" des composants critiques par les composants moins critiques)
- **Interfaces limitées** entre le système de sûreté et les systèmes non relatifs à la sûreté (contrôle-commande…).

Règles d'Architecture du logiciel (2/3)

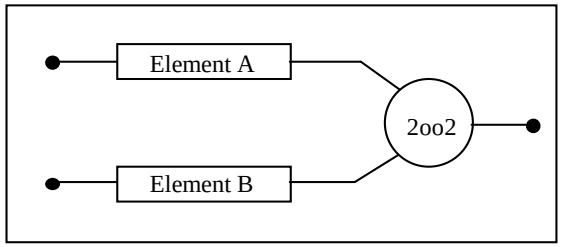
Exemples d'architectures redondantes (2 chaînes de traitement) :

1/2 de disponibilité (CEI 61508 partie 6, chapitre B.2.4.2) :



- architecture composée de 2 éléments connectés en parallèle, telle que l'un ou l'autre de ces éléments peut commander la fermeture de la sortie
- seule une défaillance importante dans les deux éléments peut entraîner une défaillance globale

2/2 de sécurité (CEI 61508 partie 6, chapitre B.2.4.3) :

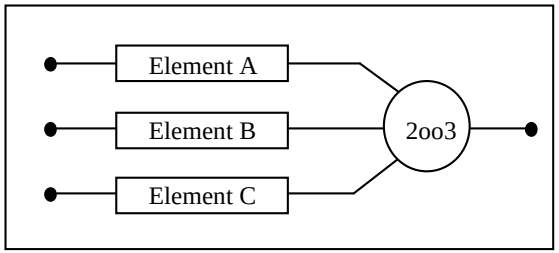


- architecture composée de 2 éléments connectés en parallèle, telle que les deux éléments sont nécessaires pour commander la fermeture de la sortie

Règles d'Architecture du logiciel (3/3)

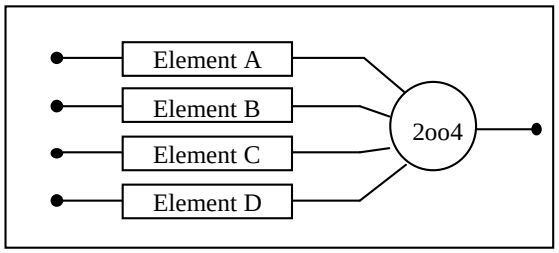
Exemples d'architectures redondantes :

2/3 de sécurité/disponibilité (CEI 61508 partie 6, chapitre B.2.4.5) :



- architecture composée de 3 éléments connectés en parallèle avec vote majoritaire pour le signal de sortie, telle que l'état de la sortie est inchangé si seulement une chaîne donne un résultat différent non en accord avec les deux autres chaînes

2/4 de sécurité/disponibilité/testabilité :



- architecture composée de 4 éléments connectés en parallèle, telle que :
. 1 chaîne en tests en ligne périodiques
(changement régulier de la chaîne testée)
. les 3 autres fonctionnent en 2/3

SERMA INGENIERIE

Exemple de techniques de programmation défensive

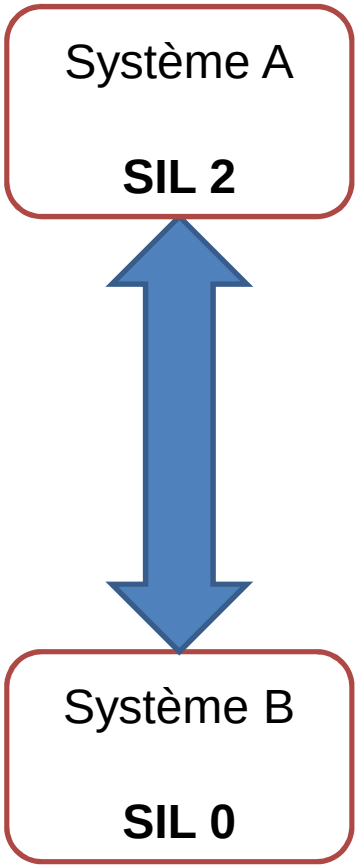
Principes :

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

Techniques de programmation défensive :

- Contrôle des **limites des données**, **des relations entre les données** (e.g. traitements de pré et de post conditions des composants logiciels)
- Contrôle de la **séquentialité des traitements,** des **temps de réponse**
- **Codes détecteurs d'erreurs** (CRC), checksum
- **Autotests** pour protéger le système contre les conditions de pannes spécifiques de l'EP (watchdog, contrôle de vraisemblance, comparaison entre voies…)
- Fonctions de suivi et de **diagnostic** des erreurs détectées en exploitation (le gestionnaire d'alarmes, le gestionnaire d'historiques des données, les tampons de fautes…)
- Fonctions de **mise dans un état sûr du système** et de **modes dégradés**

Problématiques relatives à l'architecture : cohabitation SIL 0 / SIL 2



Pour gérer la cohabitation de différents niveau de sécurité, il faut assurer l'intégrité du plus haut niveau (SIL 2 dans l'exemple).

Pour cela :

- **Contrôle du flot de données** (une donnée erronée du système B ne doit pas perturber A. **A vérifie l'intégrité** de chaque données entrantes)

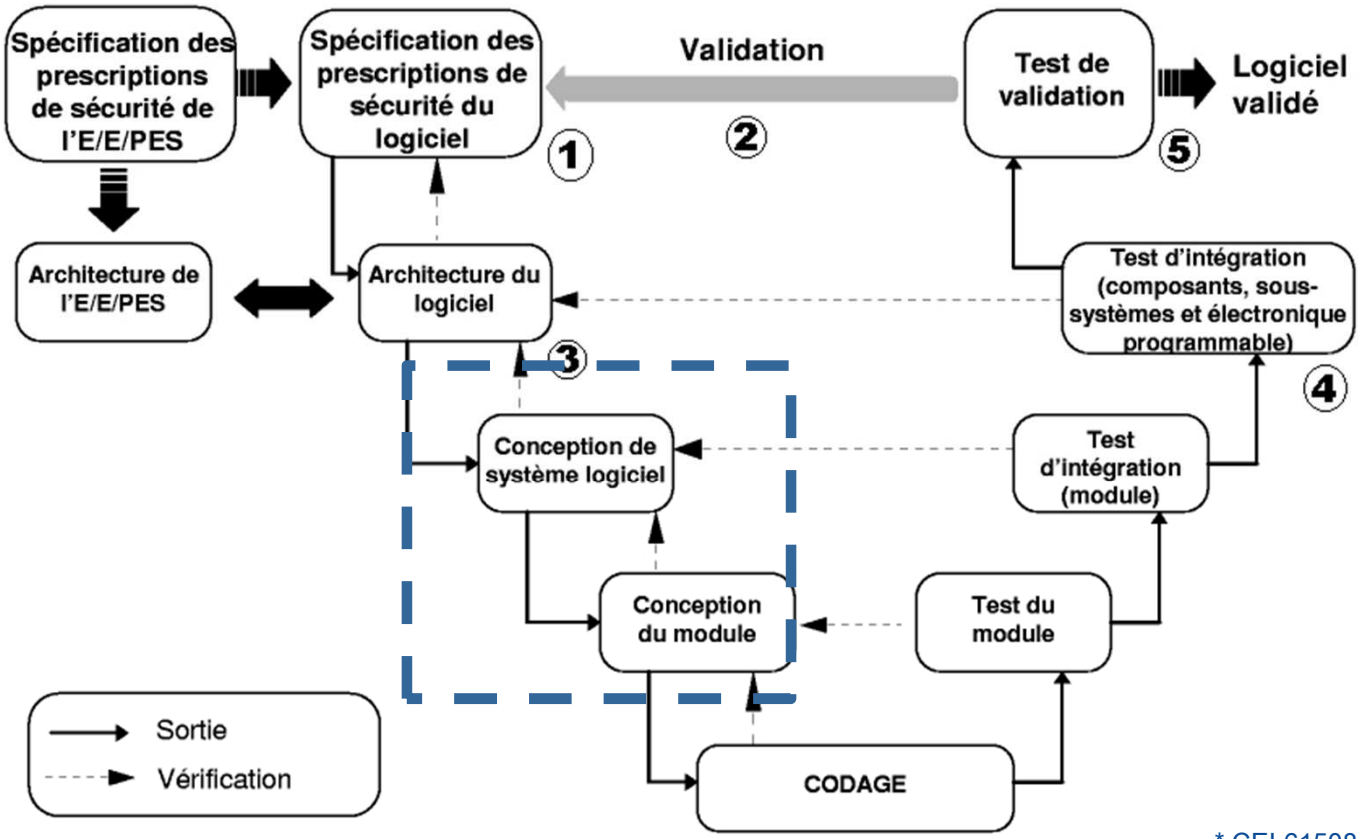
- Protection des données critiques (recopie, ...)

- **Protection de la RAM**
  - tests périodiques de l'intégrité de la RAM
  - Checksum

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP
- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel
- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)
- Conclusion

Cycle de vie de développement*



* CEI 61508 Partie 3

Figure 5 – Intégrité de sécurité du logiciel et cycle de vie de développement (modèle en V)

Objectifs de la Conception du Logiciel (1/2)

- Le document de *Conception du Logiciel* fournit la description fonctionnelle de l'ensemble complet des **modules de haut niveau** du logiciel leurs **séquencements**, leurs **données d'interface** (données entre modules, données d'environnement).

- La description des **modules bas niveau** composant chaque module haut niveau doit être fournie dans un document de *Conception des Modules du Logiciel* (ou incluse, suivant la complexité du logiciel, dans le document de *Conception du Logiciel*).

Objectifs de la Conception du Logiciel (2/2)

- La *Conception du Logiciel* doit être auto-suffisante pour **permettre de spécifier les tests unitaires** et les **tests d'intégration Logiciel/Logiciel** de type :
  - tests des valeurs aux limites (informatique) des données d'interface
  - tests des classes d'équivalence et partition des entrées et des sorties des modules

Les grandes parties du document de conception détaillée

- La conception logiciel est composée d' :
  - Une **description globale du séquencement du logiciel** (tâches, interruptions, interfaces entre les tâches)
  - Une description des **structures de données** (données partagées)
  - Une description des **allocations des ressources informatiques** (taille mémoire, temps d'exécution)
  - Une description des **modules et des fonctions du logiciel** (entrées, sorties, traitements, appels, organigramme)
  - Une **traçabilité des exigences** définies avec le document amont

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP
- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel
- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)
- Conclusion

Quelques définitions

TESTS = VÉRIFICATIONS DYNAMIQUES

"Les tests sont des contrôles consistant à s'assurer, au moyen de l'exécution du programme, que son comportement est conforme à des données préétablies " (Définition AFNOR Z61-102, GAM-T17).

V&V = VÉRIFICATION & VALIDATION

Validation : Construisons-nous le bon produit ?

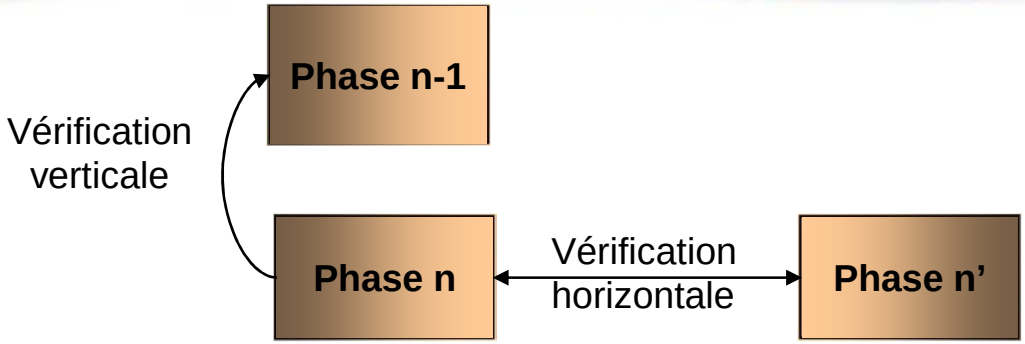
Vérification : Le construisons-nous bien ?

Tests Unitaires (TU) : tests des modules du logiciel.

Tests d'Intégration (TI) : tests des interfaces du logiciel et du logiciel avec les autres éléments du système

Tests de Validation (TV) : tests du comportement du logiciel dans son environnement.

Schéma de la vérification



La phase **n-1** représente la phase amont de la phase **n** dans la partie descendante du cycle. La phase **n'** représente la phase en vis-à-vis de la phase **n** dans la partie remontante du cycle de développement.

- **Les deux types de vérification concernent les aspects suivants :**

  - **Vérification verticale** : contrôle de conformité des produits de la phase **n** (spécifications, plans de tests, code) par rapport à la phase **n-1** (spécifications).

  - **Vérification horizontale** : contrôle de conformité des tests effectués lors de la phase **n'** par rapport au plan de tests de la phase **n**.

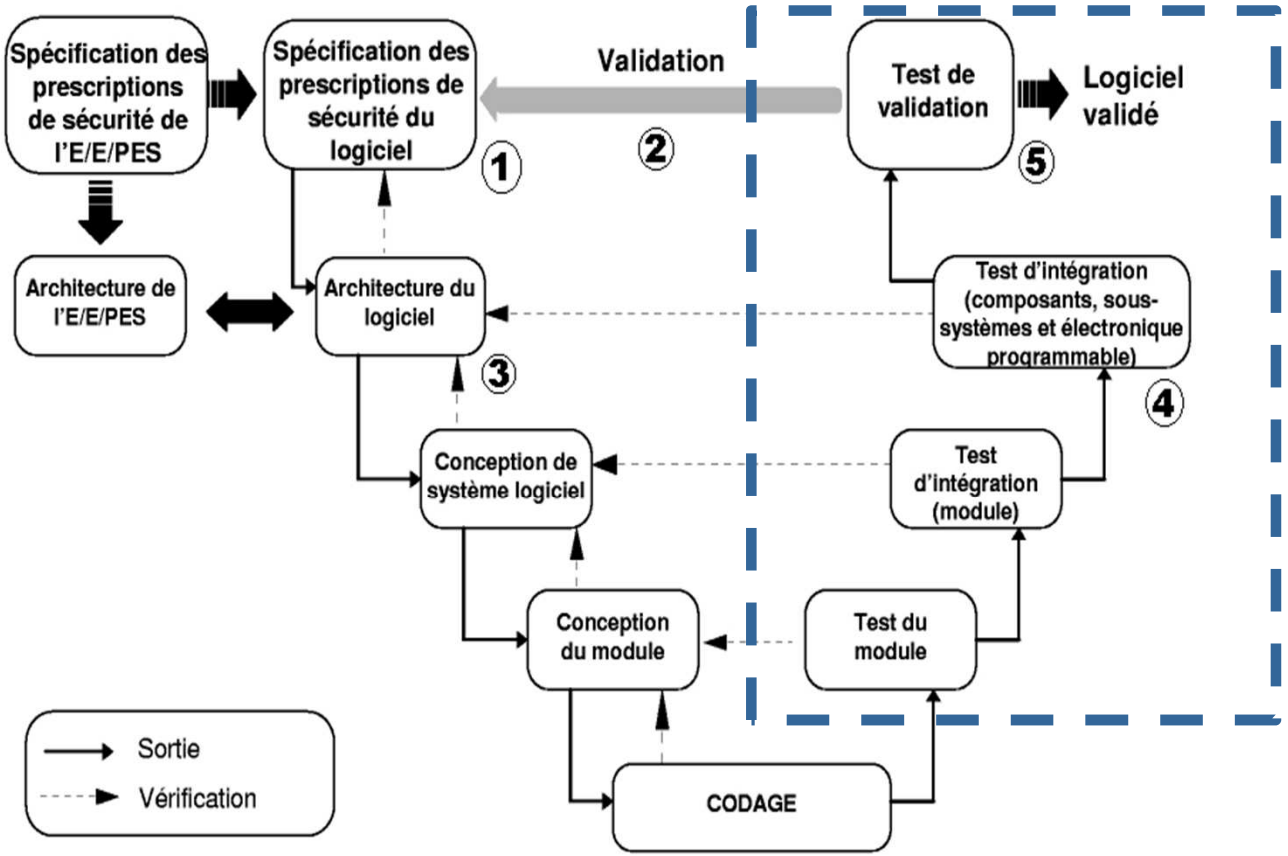
Objectifs des tests

- OBJECTIF : Détecter les défauts du logiciel
  - montrer que le produit logiciel correspond aux besoins (correction)
  - mettre en évidence les défauts latents (fiabilité).

- LIMITE : La complétude des exigences
  - difficulté d'exprimer complètement, sans ambiguïté, sans incohérence les besoins
  - complexité au niveau fonctionnelle, temporelle, séquentielle du logiciel.

- LIMITE : La complétude des moyens de tests
  - difficulté de disposer de l'EUC représentative ou de simuler le comportement fonctionnel et temporel de l'EUC
  - difficulté de simuler les défaillances aléatoires de l'électronique, « d'injecter » des fautes.

Les tests dans le cycle de développement



* CEI 61508 Partie 3

Figure 5 – Intégrité de sécurité du logiciel et cycle de vie de développement (modèle en V)

Les Tests Unitaires (TU)

Activité ayant pour but de vérifier, pour chaque module du logiciel **pris isolément**, que les résultats sont conformes au Document de Conception du Logiciel.

- **Couverture fonctionelle :**
  - couverture des classes d'équivalence du module
  - couverture des valeurs aux limites des entrées et des sorties
- **Couverture structurelle :**
  - couverture des instructions
  - couverture des branches
  - couverture des conditions
  - …
- **Traçabilité des tests** avec le Document de Conception du Logiciel.

Les Tests d'Intégration Logiciel/Logiciel (TI L/L)

Activité ayant pour but de vérifier, pour chaque module du logiciel **intégré avec les autres modules**, que les résultats sont conformes au Document de Conception du Logiciel.

- **Couverture fonctionnelle** :
  - couverture des classes d'équivalence des entrées et des sorties des modules en interface
  - couverture des valeurs aux limites des entrées et des sorties des modules en interface
- **Couverture structurelle** :
  - couverture des modules appelés
- **Traçabilité des tests** avec le Document de Conception du Logiciel.

Les Tests d'Intégration Logiciel/Matériel (TI L/M)

Activité ayant pour but de vérifier pour chaque **interaction matériel/logiciel** et chaque **interface des constituants** du logiciel (logiciel de base/logiciel applicatif, OS/Noyau Temps Réel/Tâches, bibliothèques), que les résultats sont conformes au Document d'Architecture du Logiciel.

- **Couverture fonctionnelle** :
  - couverture des valeurs fonctionnelles et aux limites de chaque interface avec l'environnement
  - couverture des valeurs fonctionnelles et aux limites de chaque interface avec les constituants
  - couverture des traitements de robustesse par injection de fautes (défaillances du matériel en interface avec le logiciel, défaillance du matériel supportant le logiciel)
- **Traçabilité des tests** avec le Document d'Architecture du Logiciel.

Les Tests de Validation du Logiciel (TV)

Activité ayant pour but de vérifier, pour **chaque fonction décrite dans la Spécification** du Logiciel, que les résultats sont conformes au Document de Spécification du Logiciel.

- **Couverture fonctionnelle** :
  - tests des classes d'équivalence des entrées et des sorties des fonctions de la spécification
  - tests aux limites des entrées et des sorties des fonctions de la spécification
  - tests des modes de fonctionnement et des changements de modes de fonctionnement
  - tests de performance : temps, mémoire…
  - tests de robustesse : charge, stress/avalanche
- **Traçabilité des exigences** avec le Document de Spécification du Logiciel

Utilité des tests / coût global

Coût de détection des défauts

- en tests unitaires	1
- en tests d'intégration	10
- en tests validation / système	100
- en exploitation	1000

(extraits de "Software Engineering Economics" Boehm)

Coût de correction des défauts du logiciel

- Spécification	1
- Architecture	2
- Conception	5
- Codage	10
- Tests unitaires	15
- Tests d'intégration	22
- Tests validation / système	50
- Exploitation	100

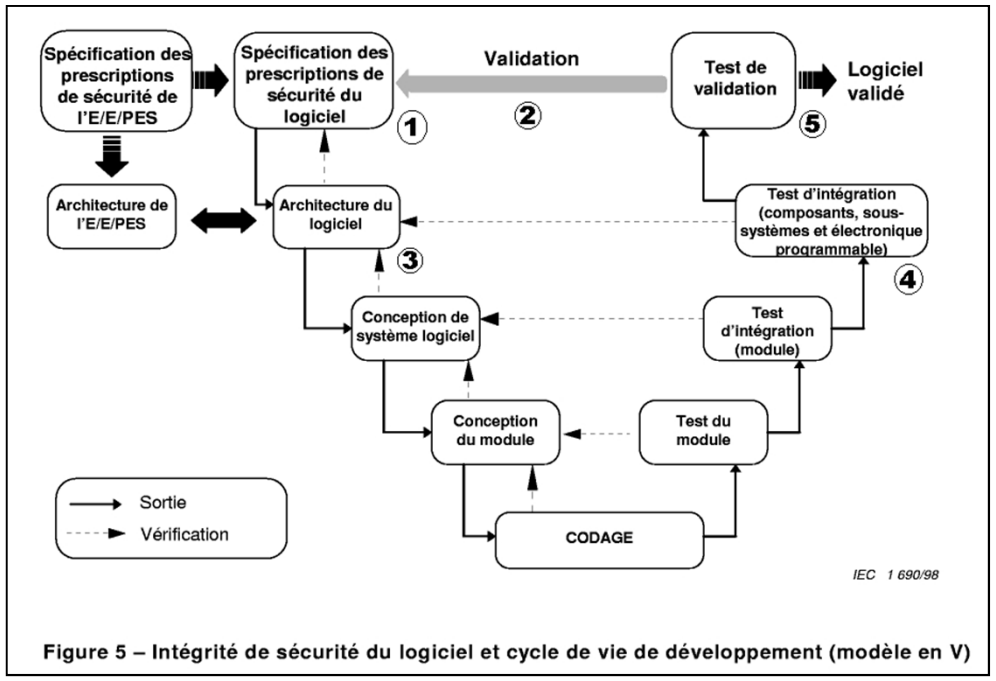


Figure 5 – Intégrité de sécurité du logiciel et cycle de vie de développement (modèle en V)

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP
- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel
- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)
- Conclusion

Activités de relecture critique

- Lecture Critique de Code (LCC)
- Revue de la documentation
- Revue des tests

Définition LCC

- Vérification de l'aspect fonctionnel du logiciel
- Vérification que chaque fonction est conforme à ses descriptions
- La LCC est une lecture comparée
  - entre les exigences de différents documents
  - entre les exigences et le code qui en découle
- Elle permet de vérifier la traçabilité des exigences

Méthode de LCC

- Elle se déroule en plusieurs étapes :
  - Vérification de la traçabilité et du raffinement des exigences entre la Spécification Système et la Spécification Logiciel.
  - Vérification de la traçabilité et du raffinement des exigences entre la Spécification Logiciel et l'Architecture Logiciel.
  - Vérification de la traçabilité et du raffinement des exigences entre l'Architecture Logiciel et la Conception Logiciel (préliminaire et détaillée).
  - Vérification de la traçabilité des exigences entre la conception et le code et de sa bonne implémentation.

Exemple LCC 1/3

- Considérons un système de régulation de vitesse d'un train

- L'une des exigences (niveau Spécification du logiciel) est :

SwRS.004.0 – Limitation de la vitesse :

« La vitesse du train ne doit pas excéder 200 km/h. »

Exemple LCC 2/3

```
137  BOOLEAN verifier_vitesse (int temps , int distance)
138  {
139
140      signed int vitesse = 0 ;
141  // le temps est exprimé en heures et la distance en km
142      vitesse = calculer_vitesse ( int temps , int distance);
143
144      if (vitesse > 200)
145      {
146          reset_logiciel();
147      }
148      ret = TRUE ;
149  return ret ;
150  }
```

Exemple LCC 3/3

Composants	Exigences	Commentaires
Vitesse.verifier_vitesse	**SwRS.004.0 – Limitation de la vitesse :** « La vitesse du train ne doit pas excéder 200 km/h »	La fonction ‘verifier_vitesse’ teste bien que la limite de vitesse n’est pas atteinte **Conforme**
…	….	…
….		

Revue de la documentation

- Vérification la conformité vis-à-vis de la norme
- Vérification de la cohérence globale, interne.
- Vérification de l'homogénéité (termes, sigles, ...)
- Effectuer sous forme de fiche de relecture :
  - Remarques initiales,
  - Réponses Équipe de développement,
  - États des points (Clos, Ouvert)

Revue des tests

- Vérification de la cohérence des plans de tests avec les dossiers et résultats de tests
- Vérification de la complétude de la couverture des exigences (Ex : Spécification du logiciel pour les Test de Validation)
- Vérification de la pertinence des tests
- Vérification des écarts de couverture (justification)
- Vérification de l'impact sur la sécurité de chacune des anomalies identifiées

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP
- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel
- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)
- Conclusion

L'analyse statique des règles FMDS du code

- Objectif : Augmenter la Fiabilité - Maintenabilité - Robustesse - Portabilité - Testabilité du code du logiciel.

Ce contrôle des règles de programmation peut se faire à la main. Cependant, on utilise en général un logiciel d'analyse statique de code.

Exemple : les règles de programmation du langage C

- Différents types de règles :
  - Liées aux fonctions
  - Liées aux instructions
  - Liées aux données

Quelques règles de Programmation C liées aux fonctions

- Tout bloc inutilisé doit être supprimé (code mort) :
  - Bloc inatteignable,
  - Fonctions C non appelées,
  - Plusieurs écritures consécutives sur une même variable

- Il faut utiliser les valeurs de retour des fonctions. En effet, un code retour retourne généralement les erreurs. Ne pas utiliser ce retour revient à ignorer l'erreur détectée (traitement de robustesse manquant).

Quelques règles de Programmation C liées aux instructions

- La division par zéro est interdite. Le risque du diviseur à zéro doit être contrôlé (traitement de robustesse). Ces instructions peuvent mener à des plantages du logiciel.

- Les boucles doivent être à limite constante ou il doit y avoir un test sur le nombre maximum d'itérations (traitement de robustesse). Ceci peut mener à des boucles infinies pouvant bloquer le logiciel.

Quelques règles de Programmation C liées aux données

- Les données de paramétrage (de configuration de l'application) doivent être protégées par checksum (traitement de robustesse).

- L'accès aux données partagées entre traitements parallèles doit être protégé (protection des données, section critique).

SERMA INGENIERIE

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP
- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel
- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)
- Conclusion

AEEL – Définitions

- Vocabulaire :
  - AEEL : Analyse des Effets des Erreurs des Logiciels
- Problématique :
  - Augmentation du nombre et de la complexité des systèmes informatique
  - Augmentation de leurs fonctionnalités

  ⇒ Augmentation des défaillances
- Définition :
  - L'AEEL est une application de l'AMDEC sur un logiciel critique
  - C'est une méthode d'analyse utilisée lors de la phase de développement du logiciel

AEEL – Hypothèse / Principe / Support

- **Hypothèse :**
  - Existence de défaillances logicielles ou matérielles
- **Principe :**
  - Propager des défaillances dans le logiciel afin d'identifier une éventuelle atteinte des évènements redoutés (ER).
- **Support :**
  - Fonctionnalité du Logiciel,
  - Graphes d'Appels,
  - Graphes de Flux de données,
  - Code.

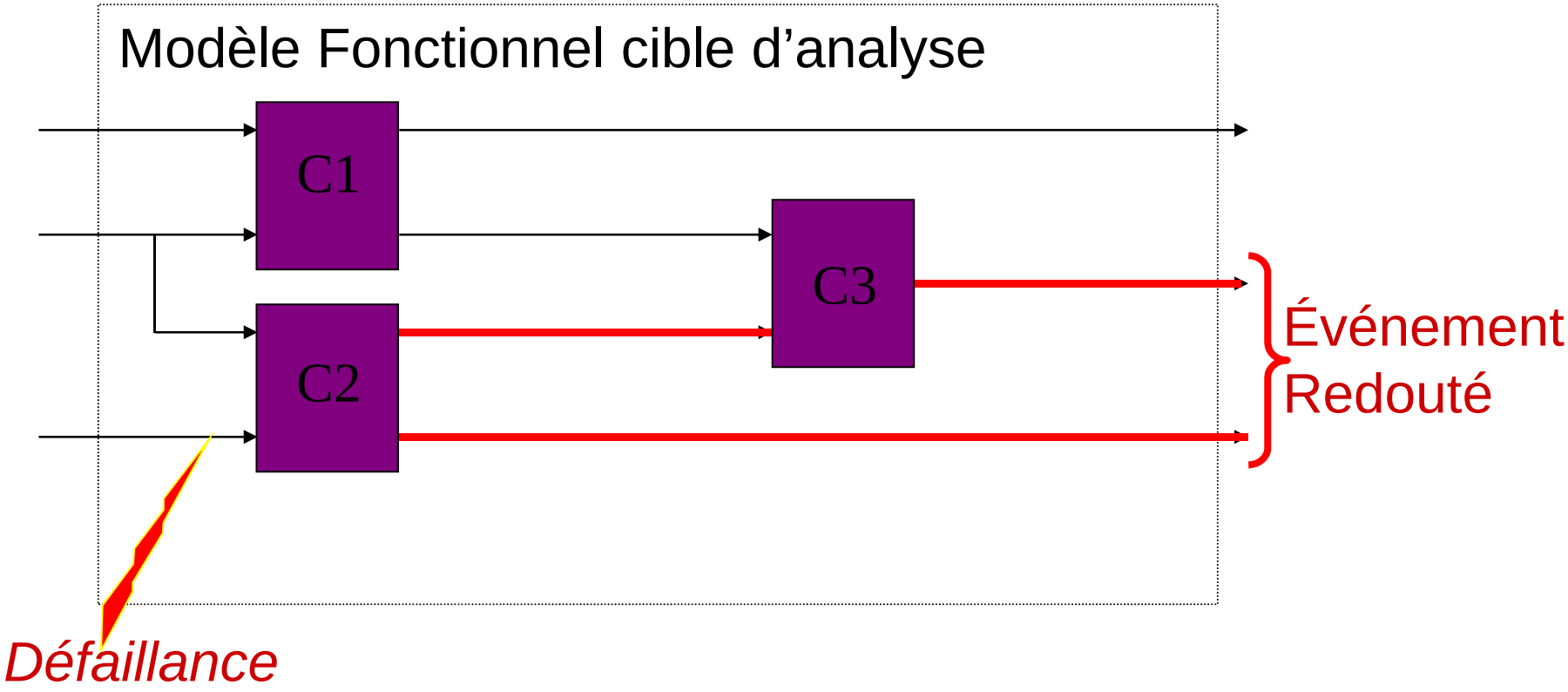
AEEL – Objectifs

- S'assurer de la non atteinte des Évènements Redoutés critiques
- Vérifier la présence de mise en position de repli avant l'atteinte des Évènements Redoutés critiques
- Vérifier la suffisance des protections présentes dans le système/logiciel

AEEL –Travail préliminaire

- **Mode de défaillance** (Perte de la fonction, Fonction erronée, Fonction active à tort, Fonction retardée)
- **Niveaux de gravité** (de 1 à 4 : du plus grave au moins grave)
- **Liste des Évènements Redoutés**
- **Niveaux et type de détection** (protection implémentée)
- **Niveaux de criticité** (fonction des barrières de sécurité)

AEEL – Évènement Redouté



Exemple Code

```
137  BOOLEAN verifier_vitesse (int temps , int distance)
138  {
139
140      signed int vitesse = 0 ;
141  // le temps est exprimé en heures et la distance en km
142      vitesse = calculer_vitesse ( int temps , int distance);
143
144      if (vitesse > 200)
145      {
146          reset_logiciel();
147      }
148      ret = TRUE ;
149  return ret ;
150  }
```

Tableaux d'AEEL correspondant

N°	Hypothèse de défaillance Module	Mode de défaillance	Effets sur le module	sur la cible d'analyse	sur le système	Id. ER	G	Détection Protections / Remarques	C
1	calculer_vitesse de verifier_vitesse	Défaillance hors intervalle	Retour erroné en dehors du domaine (>200)	**verifier_vitesse :** La variable "vitesse" est erronée à >200 : La fonction reset_logiciel est appelée.	Le logiciel est en position de repli (reset du logiciel).	-	4	-	4
2		Défaillance hors intervalle	Retour erroné en dehors du domaine (<0)	**verifier_vitesse :** La variable "vitesse" est erronée à <0 : La fonction reset_logiciel n'est pas appelée.	La vitesse est incohérente sans que le logiciel ne détecte l'erreur.	ER_vit	1	-	1
3		Défaillance dans l'intervalle	Retour erroné dans dans le domaine	**verifier_vitesse :** L'erreur sur la variable "vitesse" n'est pas detectée.	La vitesse est erronée sans que le logiciel ne le détecte.	ER_vit	1	-	1
3	verifier_vitesse	...							

Points identifiés en AEEL

- L'AEEL a permis d'identifier des comportements dysfonctionnels non détectés par le logiciel.

- La spécification du logiciel est imprécise :

SwRS.004.0 – Limitation de la vitesse : « La vitesse du train ne doit pas excéder 200 km/h. »

- Corrections à apporter :

SwRS.004.0 v2 – Limitation de la vitesse :

- « La vitesse du train doit être comprise entre 0 et 200 km/h »
- Un test de cohérence temporelle sur la vitesse doit être défini (exemple : comparaison de la vitesse du cycle précédent avec la vitesse actuelle).

Plan du séminaire

- Introduction à la SdF des Systèmes E/E/EP
- Intégration de la sécurité dans les grandes étapes du cycle en « V » :
  - La Spécification du système au logiciel : Les bonnes pratiques de spécification des exigences
  - L'architecture de l'électronique au logiciel : Les bonnes pratiques de l'architecture du Logiciel
  - La Conception du Logiciel : Les bonnes pratiques de codage
  - Les Tests du logiciel
- Méthodes et outils d'évaluation des logiciels critiques :
  - Lecture Critique de Code (LCC)
  - Analyse statique des règles FMDS du code
  - AMDEC du logiciel (AEEL)
- Conclusion

Conclusions

La Sûreté de Fonctionnement du Logiciel c'est :

- Une maîtrise des erreurs systématiques par :
  - la mise en place d'un processus de développement structurée et d'une organisation adaptée aux enjeux
  - une traçabilité assurée du début à la fin du projet
- Une parfaite gestion des modifications et des évolutions
- Des référentiels existants qui peuvent servir de guide pour « fiabiliser » le développement des logiciels

Conclusions

La Sûreté de Fonctionnement du Logiciel :

- Peut favoriser l'accès à de nouveaux marchés pour lesquels la réglementation exige la mise en œuvre de telles démarches
- Peut être un critère « marketing » différenciateur - exemple SIL
- Est un facteur d'économie si elle est prise en compte très tôt dans le développement d'un logiciel

Vos contacts

SERMA INGENIERIE

Boulevard des Chênes
5 parc Ariane
78280 Guyancourt
Tél : 01 30 05 36 00

Julien CANCELIER
Portable : +33 (0)6 86 17 76 06

Email: j.cancelier@serma.com

Michel DUFRESNE
Portable : +33 (0)6 72 77 59 06

Email: m.dufresne@serma.com