12/04/12 - ENSI de Bourges / Cap'Tronic

Sûreté de fonctionnement des logiciels et Analyses de risques

ALL4TEC
ETUDES & CONSEIL

Ingénierie

Sûreté

Fiabilité

Test

Maturité

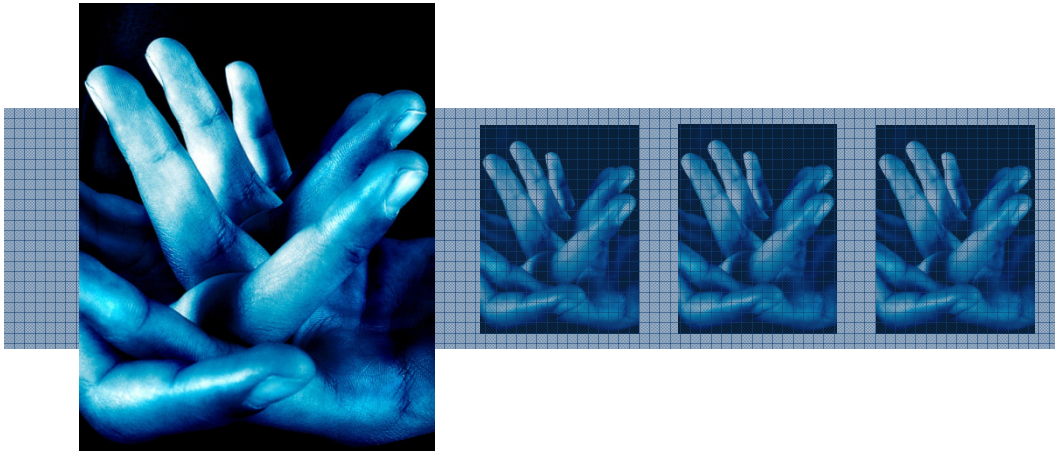
des

Systèmes Complexes

Franck Sadmi

SOMMAIRE

1 – Introduction à la SdF du logiciel

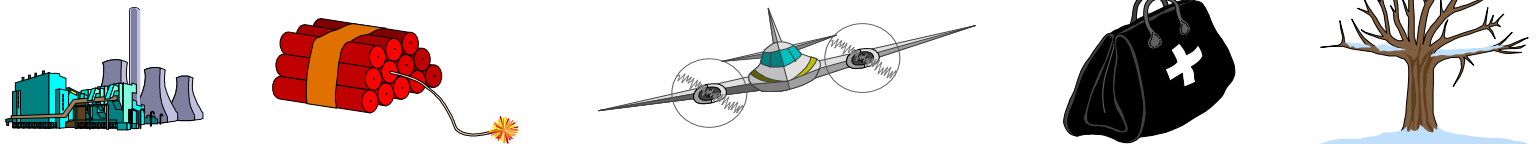


ALL4TEC
ETUDES & CONSEIL

1. Concepts de la SdF du logiciel
2. Construction et validation de la SdF du logiciel
3. Impact sur les techniques de développement

1.1 Concepts

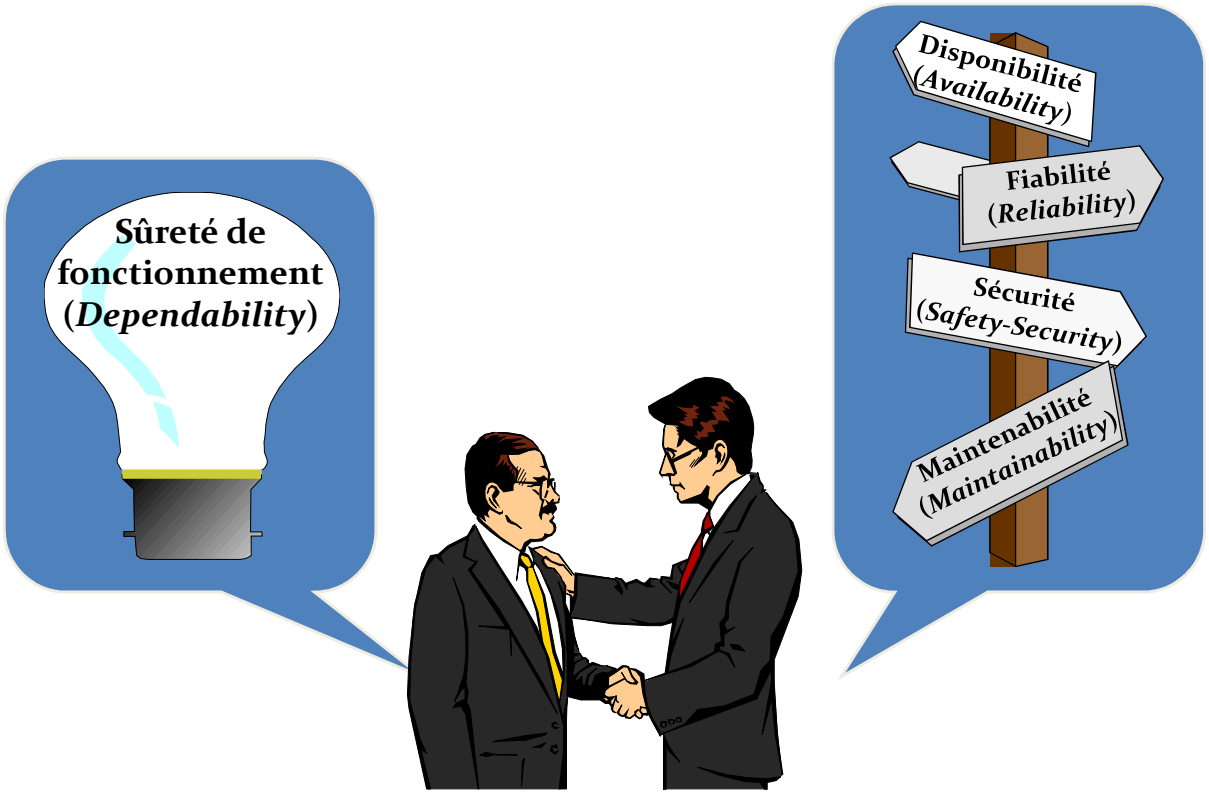
- **Positionnement de la SdF**
  - Les risques des projets industriels sont de différentes natures



  - La sûreté de fonctionnement s'intéresse à la « confiance que l'on peut placer dans l'accomplissement des missions attribuées à un système »
  - En sûreté de fonctionnement, le risque est l'apparition d'un événement indésirable (événement redouté) caractérisé en général par sa probabilité d'occurrence et sa gravité
  - Les exigences de sûreté de fonctionnement sont fonction de la nature des risques
  - La maîtrise de la sûreté de fonctionnement s'appuie sur sa construction et sa démonstration

1.1 Concepts

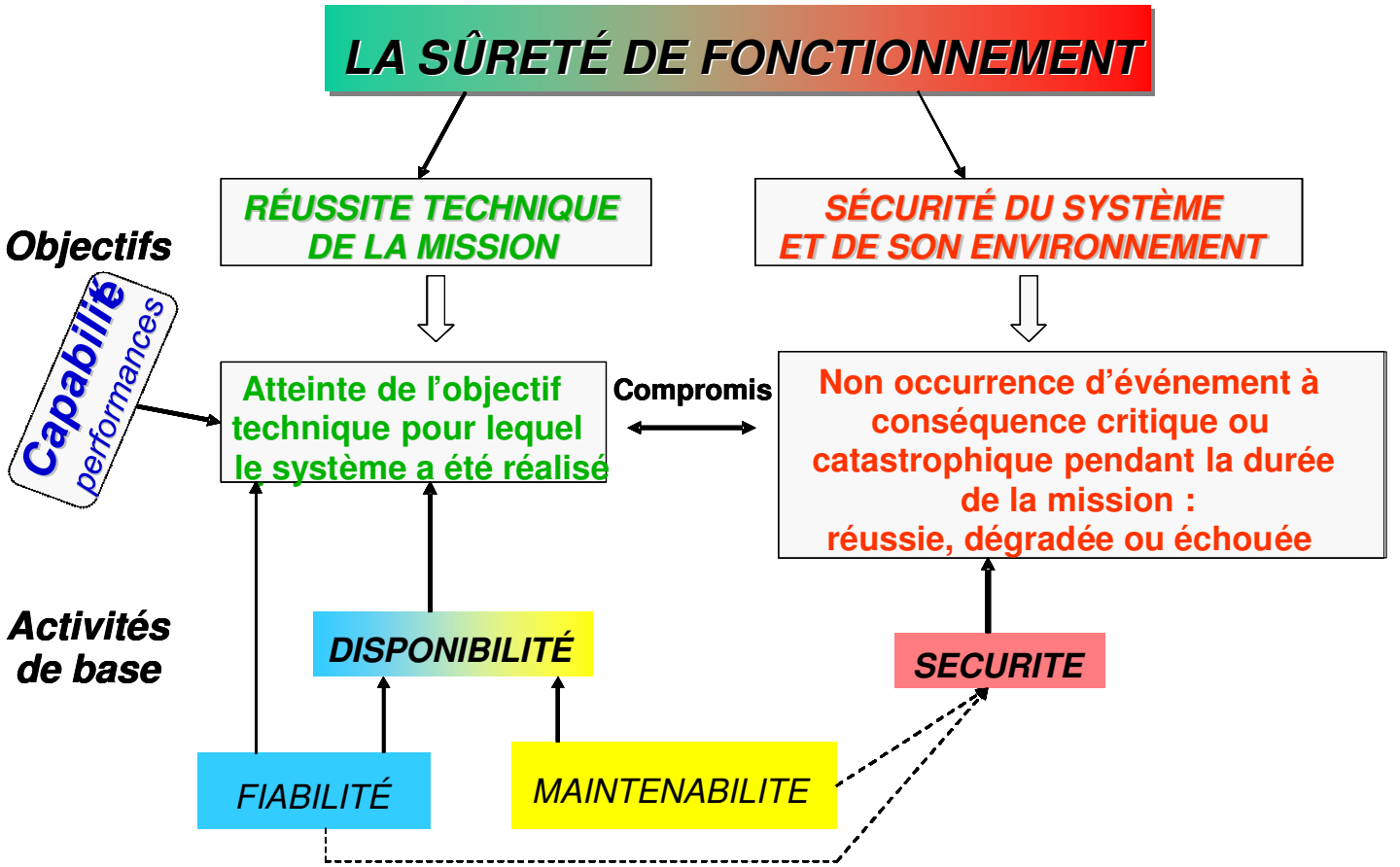
- **La Sûreté de Fonctionnement …**



… c'est surtout la réponse au besoin du client

1.1 Concepts de la SdF du logiciel

- Mais aussi …



1.1 Concepts

- **Fiabilité**
  - Probabilité de succès de mission
  - MTTF (Mean Time To Failure)
- **Disponibilité**
  - Taux de détection
  - Taux de fausse alarme
  - Capacité de reconfiguration

1.1 Concepts

- **Sécurité**
  - Nombre et nature des barrières indépendantes
  - Probabilité d'occurrence des événements critiques
- **Maintenabilité**
  - MTBF (Mean Time Between Failures)
  - MTTR (Mean Time To Repair)
  - Durée de remise en service
  - Niveau d'interchangeabilité
  - Facilité de correction/évolution
  - Pérennité des composants

1.1 Concepts

- **Principes généraux**
  - Dès le début du projet il faut prendre des dispositions pour traiter les risques identifiés
  - Entre l'expression des besoins et le code binaire exécutable, le logiciel passe par une série d'étapes de plus en plus abstraites
  - Il faut démontrer que les risques résiduels sont acceptables (Dossier de Justification de la SdF – Safety Case)
  - La maîtrise de la SdF du logiciel passe par la maîtrise de son processus de production

1.2 Construction et validation de la SdF

- On parle bien de Construction
  - Il faut impérativement prendre en compte la SdF dès les premières phases de réalisation du logiciel
  - Il faut savoir le justifier
  - Il faut être capable de « recoller les morceaux »
  - Le logiciel passe par plusieurs « états »
  - Problème général de la traçabilité

- Via la mise en place de :
  - Concepts
  - Méthodes
  - Outils
  - Techniques

1.2 Construction et validation de la SdF

- **Spécificités du logiciel par rapport au matériel (1/3):**
  - **Structure**
    - Tolérance aux fautes facile à implémenter
    - Redondance classique inefficace
    - Forte dépendance entre les composants
  - **Cycle de vie**
    - Evolutions
    - Vieillissement
  - **Comportement**
    - Défaillances difficilement reproductibles
    - Réparations impliquant une amélioration

1.2 Construction et validation de la SdF

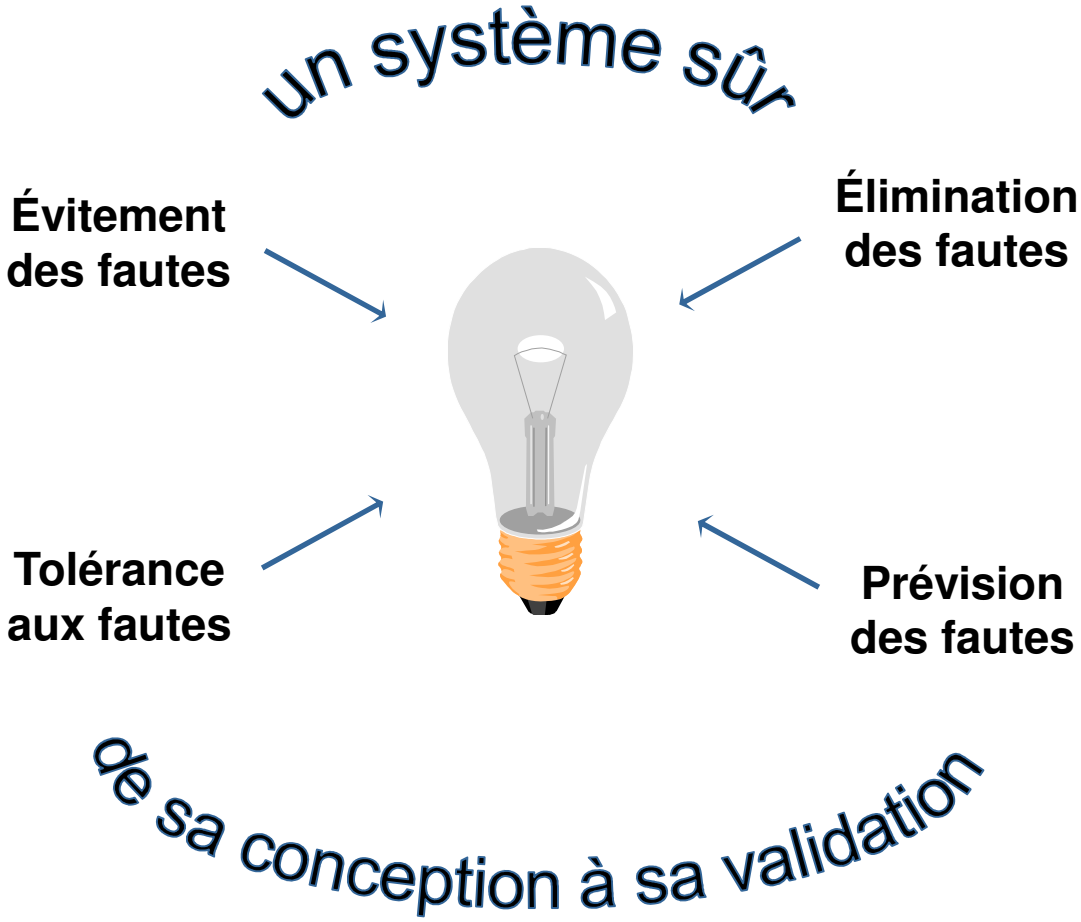
- **Spécificités du logiciel par rapport au matériel (2/3):**
  - **Gestion de défaillances**
    - Cas non traités de la fonction
    - Dépassement de valeurs
    - Conflit entre événements
  - **Lisibilité du code**
    - En fonction des langages utilisés
    - Passage de variables sans instructions particulières (Variables globales, Paramètres, …)
    - Instructions non visibles (Positionnement de « flags » automatiques, gestion de pile, héritage ...)
    - Taille du code

1.2 Construction et validation de la SdF

- **Spécificités du logiciel par rapport au matériel (3/3):**
  - **Difficultés d'appréhension du comportement du logiciel**
    - État initial difficile à déterminer
    - Interactivité des traitements parallèles
    - Activité monoprocesseur pour différents traitements temps réel
    - Instrumentation perturbatrice
    - Options d'optimisation du générateur de code ou du compilateur
  - **Impact du langage de programmation utilisé**
  - **Paramétrage**
    - Impact sur le comportement
  - **Test exhaustif impossible**

1.2 Construction et validation de la SdF

- **Principales actions en construction**



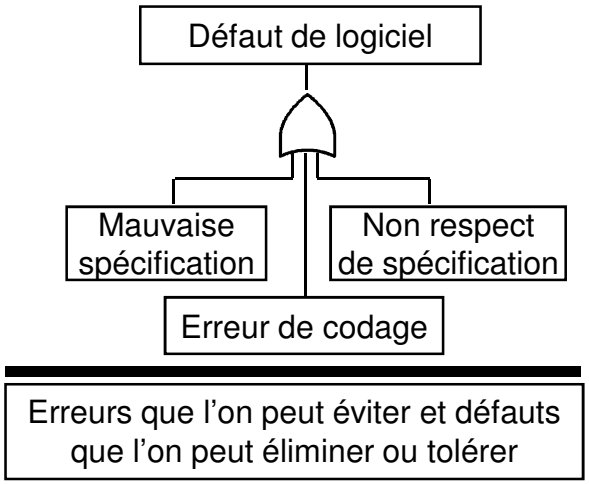
1.2 Construction et validation de la SdF

- **Évitement des défauts**
  - Obtention, par construction, d'un système robuste et de qualité, afin d'empêcher l'introduction de défauts
- **Tolérance et orientation des défaillances**
  - Spécification ou conception d'un système qui sera capable de fonctionner correctement en présence de défaillances, éventuellement de façon dégradée
- **Élimination des défauts**
  - Suppression des défauts résiduels avant la mise en service et vérification de l'absence de ces défauts
- **Prévision des défaillances**
  - Analyse et positionnement des caractéristiques de fiabilité du système, au regard des exigences exprimées par le client

ALL4TEC
ETUDES & CONSEIL

1.2 Construction et validation de la SdF

- **Types de défaillances**



1.2 Construction et validation de la SdF

- **Assurance de la SdF**
  - **Elle regroupe les activités qui doivent justifier que :**
    - Le but est identifié et conforme aux besoins
    - Les exigences de SdF sont réalisables
    - Les moyens sont identifiés et adaptés
    - Ils sont effectivement et correctement mis en œuvre
    - Les bonnes méthodes sont employées
    - Les fournitures sont conformes et produites dans les délais
    - Les bilans de SdF sont réalisés
    - Les actions décidées sont menées à bien et sont efficaces

  soit, en résumé, justifier que le but est atteint in fine

  - **En complément indispensable, la traçabilité permet la consolidation des études**
    - Elle apporte la justification de la confiance

ALL4TEC
ETUDES & CONSEIL

1.2 Construction et validation de la SdF

- **Il faut dissocier les activités de la responsabilité du chef de projet**
  - Identification du but
  - Identification des moyens requis
  - Mise en œuvre des moyens

  ... des activités d'assurance de la SdF...

MAIS

- **L'activité de SdF doit être intégrée au projet**

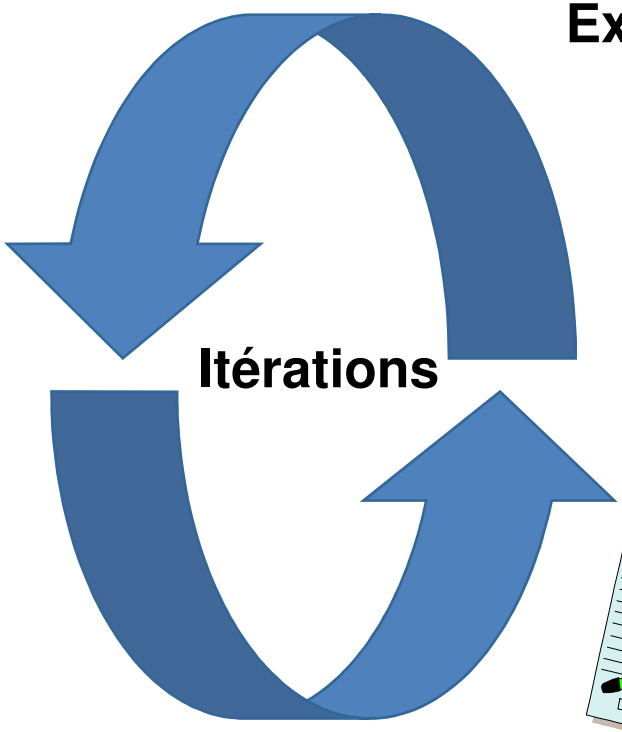
- **Les dossiers de SdF font partie de la justification de la définition du logiciel**

1.2 Construction et validation de la SdF

- **Management de la SdF**
  - **Objectifs**
    - Définir les activités de SdF
    - En identifier les acteurs
    - Vérifier la bonne mise en place de l'organisation de SdF
    - Faire le suivi des activités de SdF
  - **Moyens**
    - Plan de SdF
    - Revues de suivi
    - Gestion des actions
    - Indicateurs
    - …

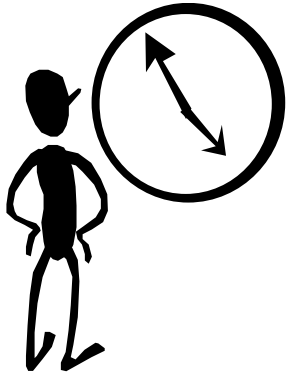
1.2 Construction et validation de la SdF

- **Plan de SdF**

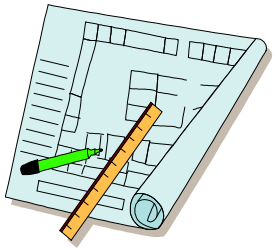


Itérations

Exhaustivité



Pertinence



Précision adaptée

1.2 Construction et validation de la SdF

- **Principes de V&V**
  - **Vérification : identifier les défauts introduits lors de la transformation des entrées en sorties d'une activité, dans le but de les éliminer**

    « Le logiciel est bien fait »

  - **Validation : établir la conformité de la sortie d'une activité par rapport aux entrées**

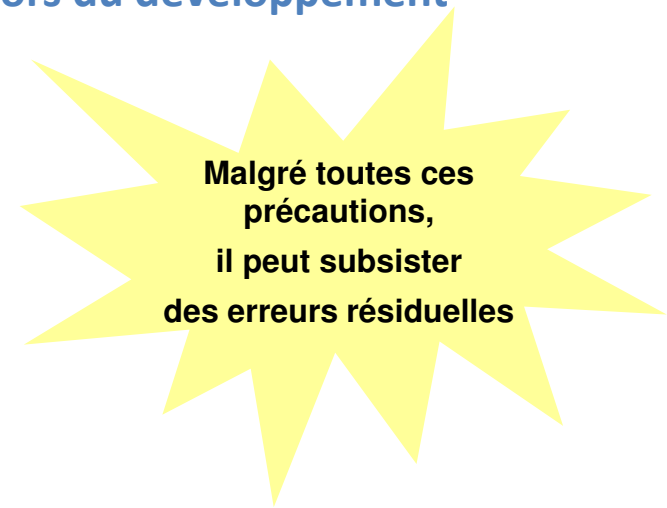
    « C'est le bon logiciel qui est fait »

- **Activités SdF : Relecture / Test / Preuve / …**

1.3 Impact sur les techniques de développement

- **Spécificité du logiciel**
  - Tout défaut de logiciel est introduit lors du développement
- **Méthodes de qualité logicielle**

  ⇒ Evitement des fautes
  - Est-ce qu'on a fait le bon logiciel ?
  - Est-ce qu'on a bien fait le logiciel ?
  - Processus de développement
  - Méthodes, règles, technologies…

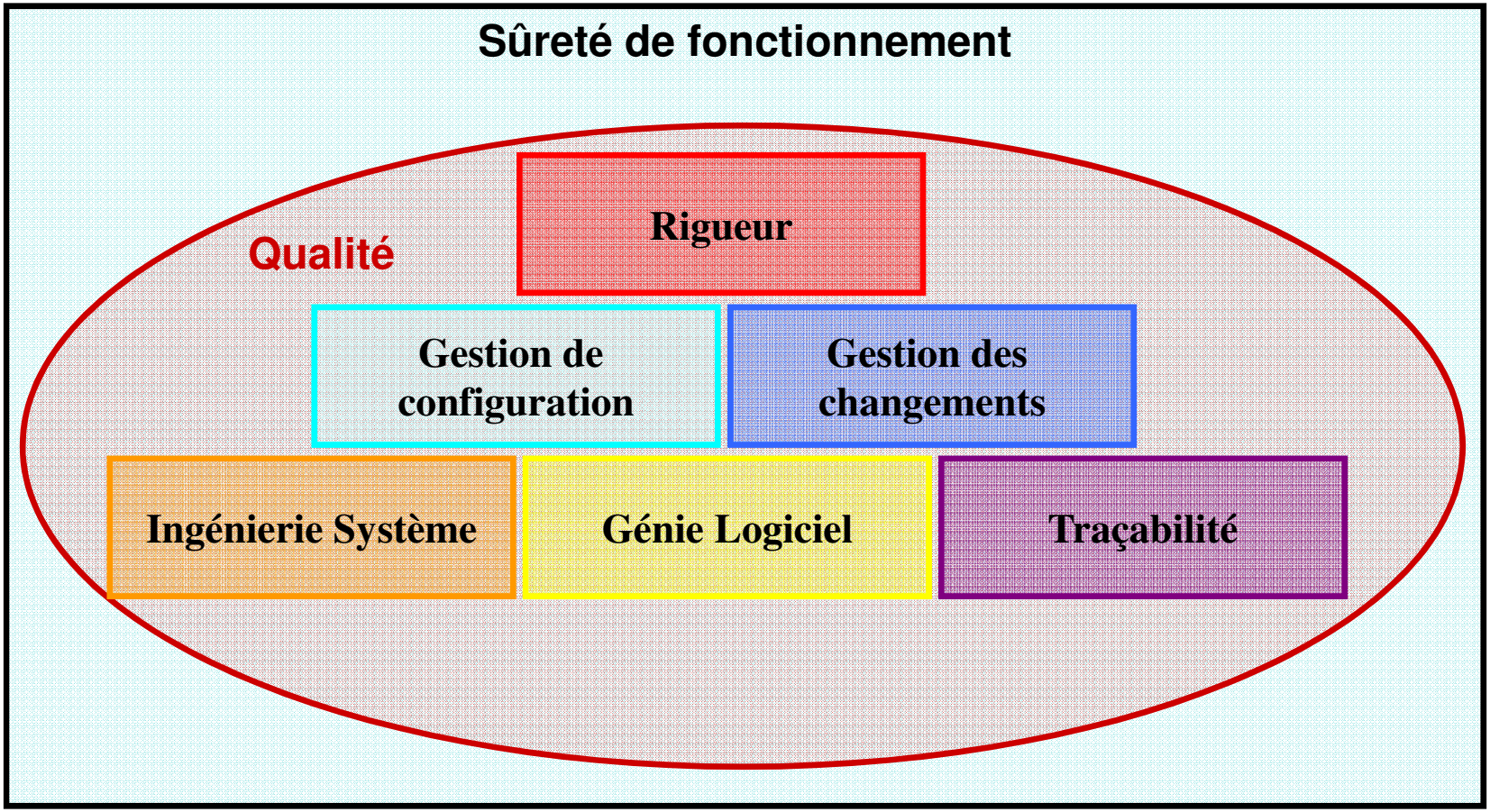


- **Méthodes de sûreté de fonctionnement du logiciel**

  ⇒ Tolérance aux fautes
  - Déterminer les risques ⇒ APR
  - Analyser le logiciel ⇒ AMDE
  - Construire un logiciel sûr ⇒ programmation défensive, BR, N-versions, …

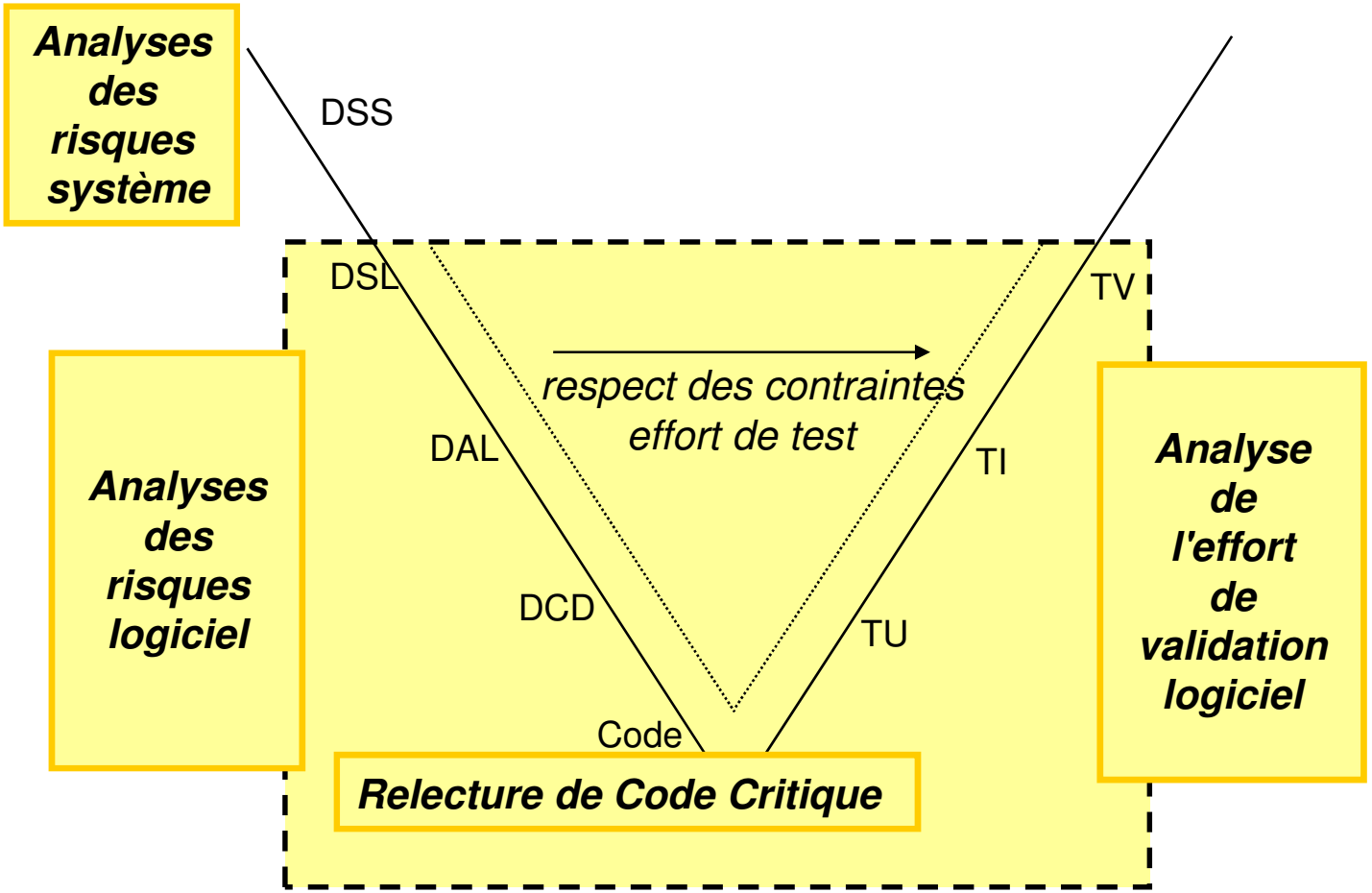
1.3 Impact sur les techniques de développement

- Pas de SdF sans Qualité



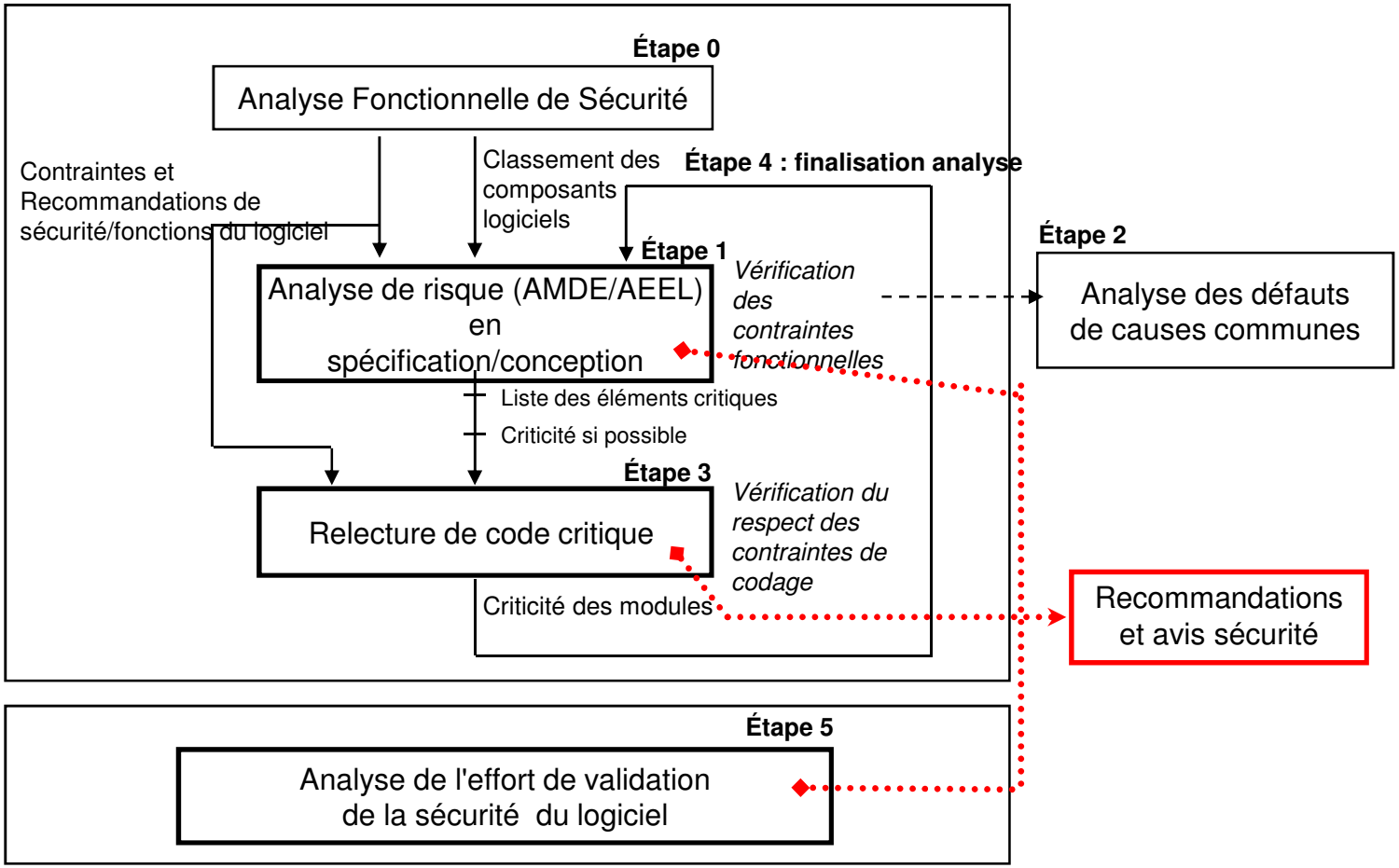
1.3 Impact sur les techniques de développement

- La SdF dans le cycle de vie en V



1.3 Impact sur les techniques de développement

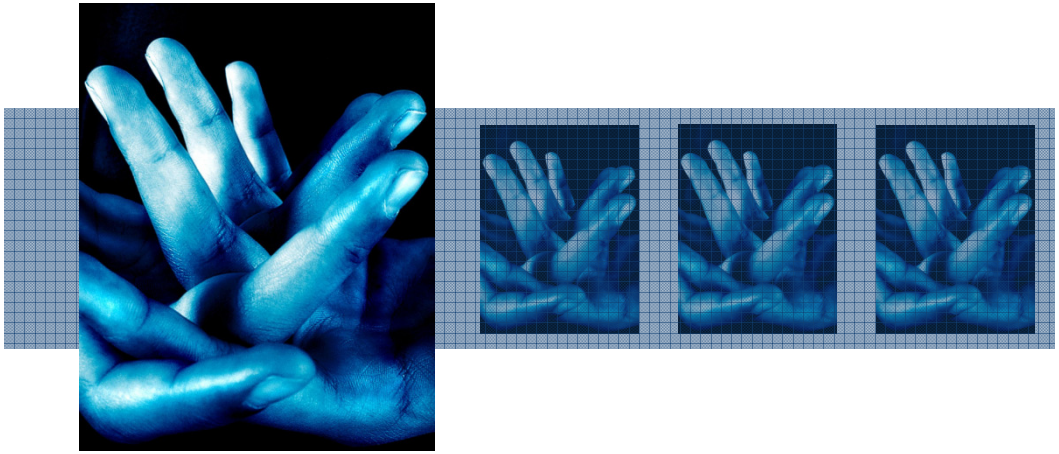
- Méthode générale



1.3 Impact sur les techniques de développement

- Recommandations générales
  - Se prémunir par construction contre certains types d'erreur
    - Exemple : le choix d'un logiciel monotâche élimine les risques de conflit d'accès à des ressources communes
  - Mettre en place des dispositifs matériels ou logiciels pour détecter des comportements anormaux éventuels du logiciel
    - Chien de garde (Watchdog)
    - Surveillance mutuelle de deux logiciels communicants
  - Utiliser un langage formel permettant d'écrire des programmes dont le comportement est prévisible
    - Lustre (SCADE), B, ...
  - Utiliser des composants éprouvés
    - Bibliothèques certifiées, COTS qualifiés, ...

2 – L’approche normative



1. Les normes existantes
2. Caractéristiques communes

2.1 Les normes existantes

- **Objectifs :**
  - Liées à des domaines d'application (secteur d'activités)
  - Fournir des recommandations sur le développement d'un logiciel
  - Fournir des exigences (activités/documentation) en vue de certification
  - Fournir un cycle de développement standard en nommant et décrivant les étapes ainsi que les documents d'entrées et de sorties
  - Fournir des descriptions sur les méthodes et outils à utiliser pour le développement du logiciel en fonction d'un SIL

2.1 Les normes existantes

- **Présentation**

  - D0 178 B : « Software Considerations in Airborne Systems and Equipment Certification »
    - Domaine d'application : Aéronautique

  - EN 50 128 : « Applications ferroviaires - Logiciels pour les systèmes de commande et de protection ferroviaires »
    - Domaine d'application : Ferroviaire

  - CEI 61 508 : «Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques - programmables relatifs à la sécurité - Partie 3: Exigences concernant les logiciels »
    - Domaine d'application : Electrique/Electronique

  - ISO 26262 : « véhicules routiers – sécurité fonctionnelle »
    - Domaine d'application : automobile

2.2 Caractéristiques communes

- Aux normes 50128 / 26262 / 61508-3
  - Elles définissent une méthodologie d'analyse de sûreté de fonctionnement
  - Elles donnent des exigences relatives à la Sécurité fonctionnelle
    - Qualité du développement logiciel
    - Validation du matériel (et des interfaces)
    - Gestion documentaire et de configuration
  - Elles sont très orientées processus : elles donnent les activités à réaliser et leurs objectifs
    - Exigences couvrant tout le cycle de vie, de la conception à la fin de vie
    - Validations pour les différentes phases
    - Exigences classées par activités
  - Elles donnent des recommandations ou des exigences quant à l'organisation à mettre en place
  - Elles se basent sur des Safety Integrity Level
  - Elles proposent un choix de méthodes et techniques
  - Elles imposent de produire une documentation dédiée à la sécurité

2.2 Caractéristiques communes

- Exemple de plan de norme (extraction DO178)

- § 1 – Introduction
- § 2 – Les aspects système en relation avec le développement logiciel
- § 3 – Cycle de vie du logiciel
- § 4 – Processus de planification du logiciel
- § 5 – Processus de développement du logiciel
- § 6 – Processus de vérification du logiciel
- § 7 – Processus de gestion de configuration du logiciel
- § 8 – Processus d'assurance qualité du logiciel
- § 9 – Processus de certification
- § 10 – Vue générale de la certification d'un avion et d'un moteur
- § 11 – Les données du cycle de vie du logiciel
- § 12 – Compléments

ALL4TEC
ETUDES & CONSEIL

2.2 Caractéristiques communes

- Exemple d'exigences (extraction 61508-3)

§ 7.5 Intégration de l'électronique programmable (matériel et logiciel)

§ 7.5.2.2 La spécification des essais d'intégration du logiciel/électronique programmable (matériel et logiciel) doit indiquer les points suivants:

a) la division du système en niveaux d'intégration;

b) les cas d'essai et données d'essai;

c) les types d'essais à effectuer;

d) la description de l'environnement d'essai comprenant les outils, le logiciel support et la configuration;

e) les critères suivant lesquels la réalisation de l'essai est jugée.

2.2 Caractéristiques communes

- Exemple de techniques (extraction 50128)

Tableau A.8 – Articles à évaluer

ARTICLE A EVALUER	Article	NISL 0	NISL 1	NISL 2	NISL 3	NISL 4
1. Niveaux d'intégrité de la sécurité du logiciel	5	HR	HR	HR	HR	HR
2. Personnel & Responsabilité	6	-	R	R	HR	HR
3. Cycle de vie & documentation	7	-	HR	HR	HR	HR
4. Spéc. des Exigences du Lg.	8	R	HR	HR	HR	HR
5. Architecture du Logiciel	9	-	R	R	HR	HR
6. Conception & Développement	10	-	R	R	HR	HR
7. Vérification	11	-	HR	HR	HR	HR
8. Intégration logiciel/matériel	12	-	R	R	HR	HR
9. Validation du logiciel	13	-	HR	HR	HR	HR
10. Assurance qualité	15	-	HR	HR	HR	HR
11. Maintenance	16	-	HR	HR	HR	HR

2.2 Caractéristiques communes

- Exemple de techniques expliquées (extraction 61508-7)

B.2.3.2 Automates finis/diagrammes de transition d'état

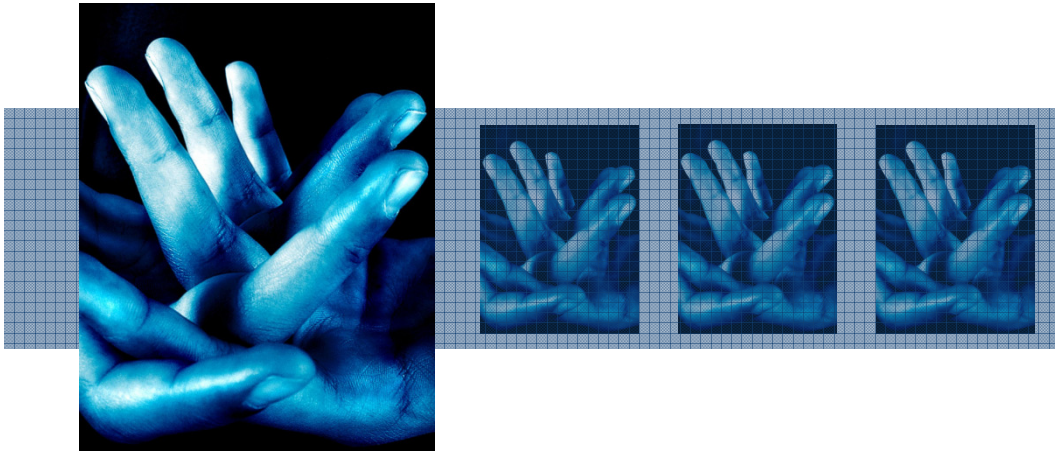
NOTE Cette technique/mesure est mentionnée dans les Tableaux B.5, B.7, C.15 et C.17 de la CEI 61508-3.

But: Modéliser, vérifier, spécifier ou mettre en place la structure de commande d'un système.

Description: De nombreux systèmes peuvent être décrits en termes d'états, d'entrées et d'actions. Ainsi, dans l'état S1, à réception d'une entrée I, un système peut exécuter l'action A et passer à l'état S2. En décrivant les actions d'un système pour chaque entrée dans chaque état, il est possible de décrire entièrement un système. Le modèle de système résultant est appelé automate fini. Il est souvent représenté sous forme de «diagramme de transition d'état» montrant comment le système se déplace d'un état à un autre ou sous forme de matrice dont les dimensions sont l'état et l'entrée. Les cellules de la matrice contiennent par ailleurs l'action et le nouvel état qui est le résultat de la réception de l'entrée dans l'état donné.

…

3 – Techniques d'analyses de risques



1. AMDE
2. Arbre de Défaillance
3. Exemples d'outils

3.1 AMDE

- **Principes d'une AMDE**

  - Méthode inductive s'appuyant sur l'étude d'impact des modes de défaillance élémentaires et permettant de :
    - Déterminer les parties critiques
      - Identifier les composants qui conduisent à un événement redouté
    - Recommander des actions pour prévenir les défaillances éventuelles
      - Renforcer les barrières de sécurité : tolérance aux fautes
      - Orienter la validation : efforts de test particuliers

3.1 AMDE

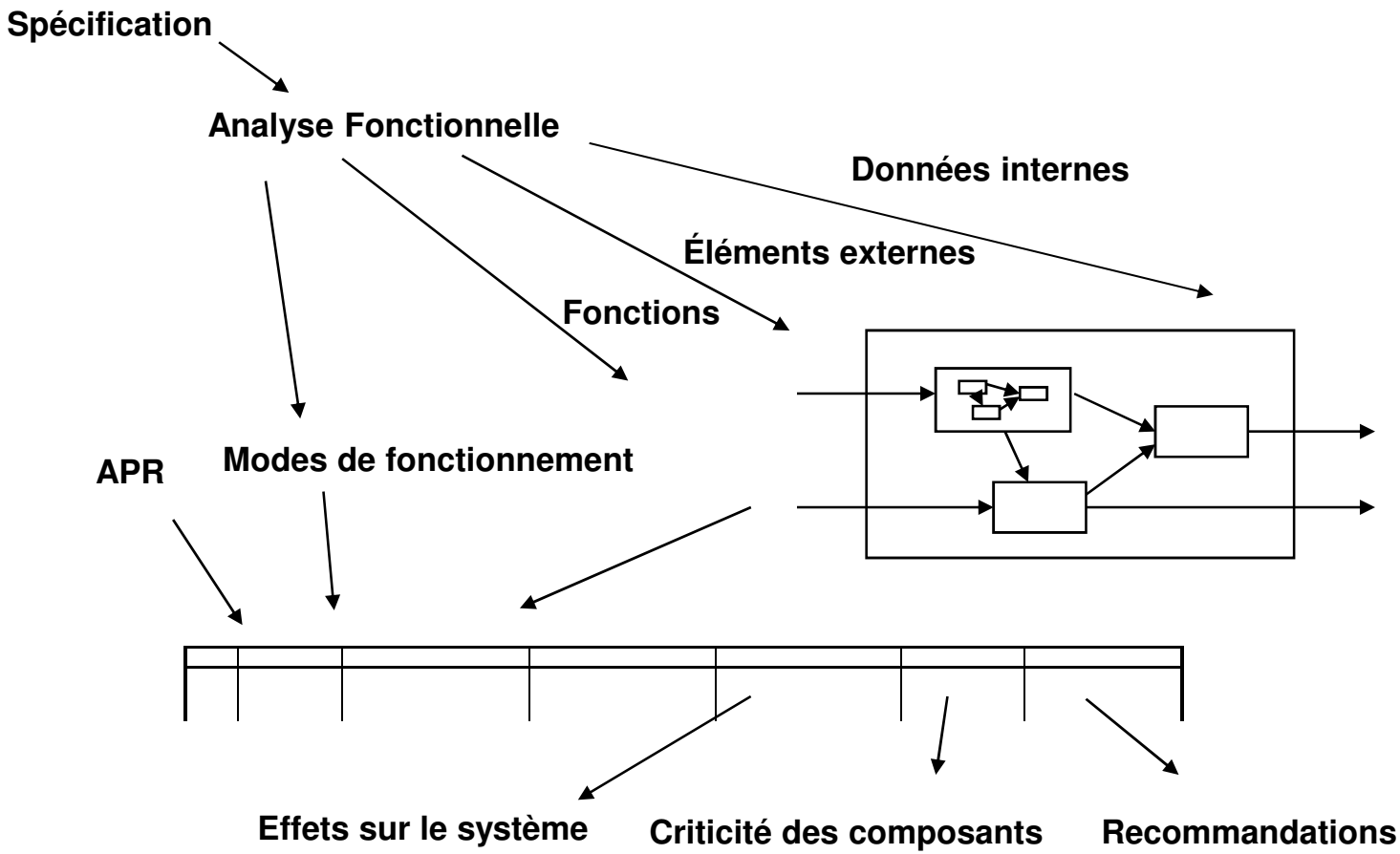
- **Objectifs de l'AMDE**
  - Détecter les erreurs d'adressage
    - Surveillance des accès en lecture/écriture à des adresses illégales de l'espace adressable du microprocesseur (traitement des "bus error")
    - Surveillance des accès en écriture dans des zones mémoires utilisées uniquement en lecture (constantes, instructions, ...)
    - Surveillance des accès illégaux dans des zones mémoires utilisées en lecture/écriture
  - Détecter les erreurs du flot de contrôle
    - Surveillance du temps de cycle : dispositif de chien de garde (Watchdog) matériel
  - Détecter les erreurs du flot de données
    - Surveillance de la cohérence des données, en utilisant la redondance de certaines informations
    - Surveillance de la cohérence des données, en utilisant les valeurs limites sur certaines informations
    - Surveillance des conditions d'appel de certaines fonctions mathématiques (division par 0, racine carrée d'un nombre négatif) ainsi que du débordement des calculs numériques
  - Et de manière générale ...

3.1 AMDE

- **Pré-requis : Architectures fonctionnelle et organique**
  - L'architecture fonctionnelle doit permettre de :
    - Identifier toutes les fonctions
    - Identifier les interactions entre fonctions
    - Décrire le comportement des fonctions
    - en s'appuyant par exemple sur : SADT, SART, Statemate, UML, …
  - L'architecture organique doit permettre de :
    - Identifier tous les modules
    - Identifier les données
    - Préciser les entrées/sorties des modules
    - Décrire le comportement des modules

3.1 AMDE

- **Synoptique**

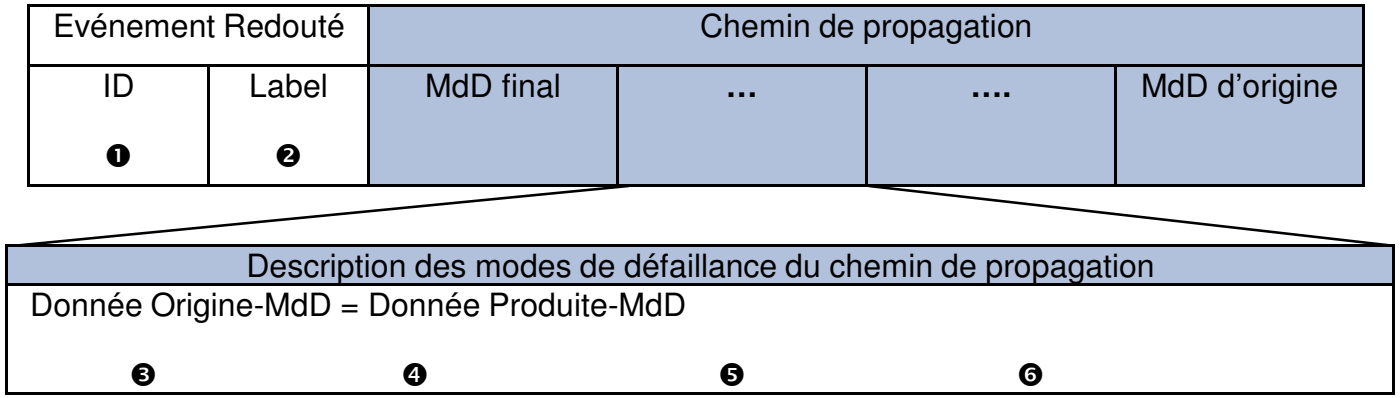


3.1 AMDE

- **Mise en œuvre (2 étapes):**
  - Analyse locale
    - Chaque effet en sortie d'un composant est traduit comme un mode de défaillance d'une entrée des composants consommateurs
    - Les modes de défaillance sont génériques :
      - Absent
      - Intempestif
      - Erroné
  - Analyse globale
    - Etude de l'impact de la propagation des modes de défaillance
    - Recherche des chemins menant aux événements redoutés
- **Dans chacune des deux étapes :**
  - Préciser les moyens de détection
    - Le bon niveau de détection est choisi en examinant la propagation
  - Donner des recommandations
    - Mesures à prendre pour détecter, isoler, atténuer fortement les défaillances
    - Cf. les mécanismes de tolérance aux fautes

3.1 AMDE

- **Présentation des résultats**



❶ donne l'identifiant de l'ER étudié ❷ donne le label de l'ERL étudié

❸ et ❹ identifient les modes de défaillance en entrée :

❸ = donnée à l'origine de la défaillance

❹ = nature générique du mode de défaillance de l'entrée (manquante, erronée, production intempestive)

❺ et ❻ identifient l'effet sur les sorties :

❺ = nom de la sortie impactée

❻ = nature générique du mode de défaillance de la sortie (manquante, production erronée, production intempestive)

3.1 AMDE

- **Suivi des résultats d'analyse**

  – Vérifier que :
    - Les moyens de détection proposés
    - Les recommandations émises
    - Les actions demandées

  ont bien été mis en œuvre en conception ou en test

  – Faire un suivi indépendant de l'analyse (livret des points critiques par exemple)

  – Maintenir en configuration

3.1 AMDE

- **Conclusion**
  - Intérêt
    - Efficace si appliquée à des composants logiciels pouvant être à l'origine de défaillances de l'ensemble du système
    - Permet d'optimiser les tests : en fonction de la criticité des composants
  - Limites
    - Analyse mal adaptée à la représentation d'évènements dynamiques
    - Les modes communs de défaillance ne sont pas gérés
    - Analyse conduisant à des redondances si les évènements redoutés sont trop inter-dépendants
  - Difficultés
    - Mise en œuvre délicate pour des systèmes complexes comportant un nombre de composants élevé
    - L'analyste doit bien connaître le logiciel
    - La qualité d'une analyse est difficile à évaluer par un non spécialiste

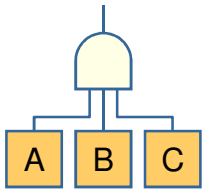
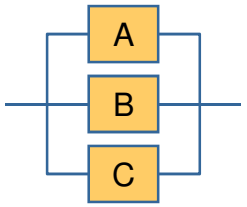
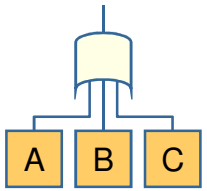
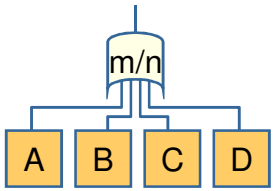
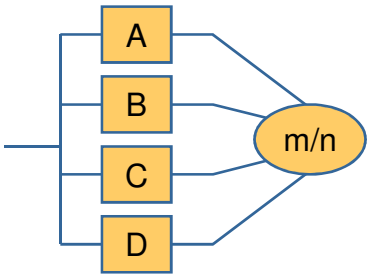
3.2 Arbre de Défaillance

- **Objectifs**
  - Quantifier la fiabilité, disponibilité, sécurité
  - Faire des allocations d'objectifs
  - Focaliser sur les événements les plus indésirables et ne prendre en compte que les combinaisons de modes de défaillances significatives pour ces événements
  - Aider au diagnostic après incident

- **Principes de mise en œuvre**
  - Utilisation des résultats d'APR et (éventuellement) d'AMDE
  - Démarche déductive par décomposition successive aux niveaux inférieurs
  - Représentation par arbre de défaillance (AdD) ou par bloc diagramme de fiabilité (BdF)

3.2 Arbre de Défaillance

- **Mode de représentation**

Opérateur	Représentation par AdD	Représentation par BdF
ET		
OU		
COMBINAISON (m, n)		

3.2 Arbre de Défaillance

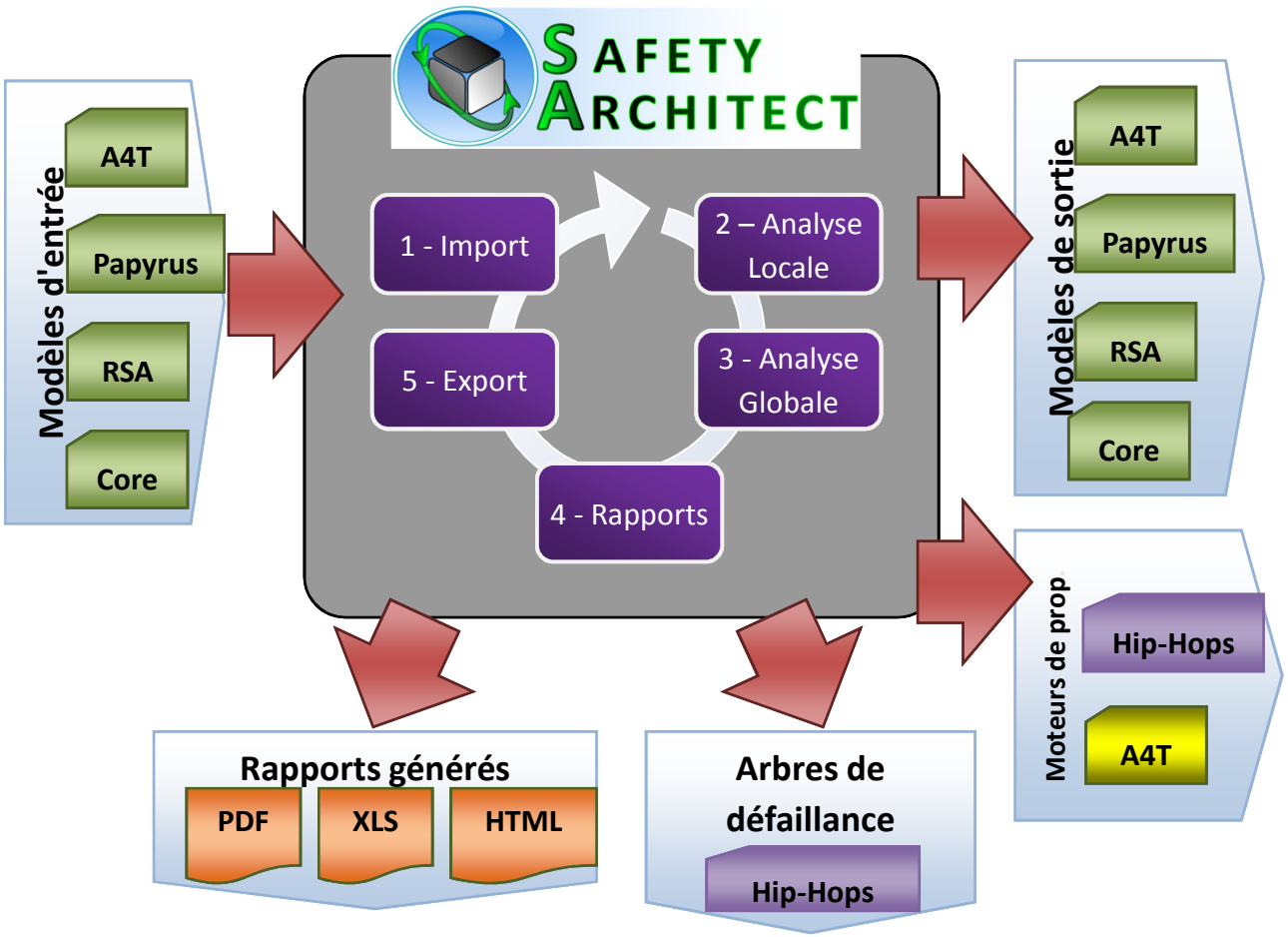
- **Construction et exploitation de l'AdD**
  - Construction
    - Recherche des causes des événements non élémentaires
    - Décomposition de l'arbre en événements élémentaires
  - Exploitation
    - Traitement qualitatif
      - Recherche des coupes minimales ou chemins critiques
      - Plus petites combinaisons d'événements élémentaires entraînant un événement redouté
      - L'ordre de la coupe est le nombre d'événements élémentaires nécessaires
    - Traitement quantitatif
      - Par calcul des probabilités (avec calcul de sensibilité)

3.3 Exemples d'outils

- Méthodes formelles : SCADE, Atelier B …
- AMDE du logiciel : Safety Architect, OCAS (BPCDAS), SIMFIA …
- Réseau de Petri : Miss-RdP, MOKA-RP …
- Arbres de défaillance : Aralia Simtree, Gamtree, ITEM toolkit, Relex …
- Analyse statique : Polyspace, Klocwork, QAC, Logiscope …
- Injection de faute : Claire, Exhaustif …
- Test selon le profil d'usage : MaTeLo …
- Calculs de fiabilité du logiciel : M-élopée …
- …

3.3 Exemples d'outils

- **Safety Architect**



4 – Questions