UTILISATION DES AUTOMATES PROGRAMMABLES INDUSTRIELS DÉDIÉS À LA SÉCURITÉ

Guide pour le développement du logiciel applicatif[1]

☐ Automate programmable
☐ Logiciel
☐ Conception
☐ Méthodologie

► *Pascal LAMY, Philippe CHARPENTIER*
INRS, Département Ingénierie des équipements de travail

L'utilisation des automates programmables dédiés à la gestion des fonctions de sécurité des systèmes de commande de machine pose le problème du développement du logiciel applicatif. Un travail de sensibilisation à une bonne pratique de développement du logiciel est nécessaire. C'est en effet un des moyens pour éviter d'introduire des fautes dans le logiciel et s'assurer que le système développé pourra remplir sa mission.
Le projet de norme FDIS CEI 62061 comporte des prescriptions relatives au cycle de développement d'un logiciel. Cet article propose une hiérarchisation de ces prescriptions afin d'en faciliter l'usage. Une présentation sous forme d'étapes à atteindre est proposée, allant d'un ensemble minimal de prescriptions pour finir par un ensemble plus complet. Nous donnons aussi des renseignements complémentaires sur ces prescriptions et nous énumérons des moyens adaptés pour les satisfaire.

Depuis que l'utilisation des systèmes de commande à électroniques programmables (souvent appelés systèmes électroniques complexes) est autorisée en sécurité des machines [1], la tendance est de remplacer, pour la gestion des fonctions de sécurité, une logique câblée par une logique à base d'automates programmables industriels dédiés à la sécurité [2]. Ces systèmes utilisent alors du logiciel qui participe à la fonction de sécurité du système.

Un automate dont l'architecture matérielle permet de traiter des fonctions de sécurité ne suffit pas à garantir que l'ensemble de l'installation soit de sécurité. En effet, une erreur dans le logiciel applicatif [3] peut entraîner des dysfonctionnements dangereux : il suffirait par exemple d'exécuter une instruction erronée non testée lors de la conception (donc à l'origine d'une

[1] *On distingue le logiciel embarqué ou système (logiciel interne à l'automate développé par le fabricant de l'APIdS) du logiciel applicatif ou plus communément appelé « programme automate » développé par l'utilisateur de l'APIdS par exemple le fabricant ou intégrateur d'une machine.*

USE OF SAFETY-DEDICATED PROGRAMMABLE LOGIC CONTROLLERS (PLCS).
APPLICATION SOFTWARE PROGRAM DEVELOPMENT GUIDE.

Use of dedicated programmable logic controllers for managing machine control system safety functions raises the problem of developing application software. Work involving awareness of software development best practice is required. This is effectively one way of avoiding fault inclusion in the program and of ensuring that the developed system can fulfil its assignment.
Draft standard FDIS IEC 62061 includes requirements related to a software development cycle. This paper proposes prioritization to facilitate requirement implementation. Presentation in the form of stages to be reached, ranging from a minimum set to a fuller set of requirements, is proposed. Additional information on these requirements is also included and suitable means of meeting them are listed.

☐ Programmable logic controller (PLC)
☐ Software program
☐ Design
☐ Methodology

erreur latente) pour activer une sortie influant sur la sécurité de l'application. Il est donc important de minimiser le nombre de ces erreurs par des prescriptions à suivre lors du développement et de prendre les mesures nécessaires pour que ces erreurs n'aient pas de conséquences sur la sécurité.

Le projet de norme FDIS CEI 62061 [4] propose une approche globale pour la réalisation de systèmes de commande à base de technologies Électriques / Électroniques / Électroniques Programmables pour effectuer des fonctions de sécurité de machines [5]. Il faut, en suivant cette démarche :
• identifier les différentes fonctions de sécurité,
• allouer, à partir d'une évaluation des risques, un niveau de sécurité à chaque fonction de sécurité,
• permettre la conception appropriée du système de commande,
• intégrer les sous-systèmes relatifs à la sécurité,
• vérifier que le système électrique de commande respecte les spécifications de performance définies lors de l'évaluation des risques.

Il donne des prescriptions adaptées au développement et à la conception des ensembles intégrant des systèmes électroniques complexes. Il s'intéresse en particulier au développement du logiciel applicatif des systèmes programmables utilisés en sécurité des machines et fournit ainsi des prescriptions pour son développement lors des différentes phases de son cycle de vie. On notera que ces prescriptions sont essentiellement de type qualité logicielle et qu'elles peuvent être comparées à du Génie Logiciel [7].

Ce document concerne exclusivement l'aspect développement du logiciel applicatif (le logiciel défini et validé par l'utilisateur) du système de commande électrique de sécurité. A partir du projet de norme FDIS CEI 62061 qui propose le respect de prescriptions afin de s'assurer du bon développement du logiciel applicatif (cf. parties 6.10, 6.11 et 6.12 du projet), nous avons réalisé une hiérarchisation de ces pres-

LE CYCLE DE DÉVELOPPEMENT LOGICIEL

Tout concepteur d'une installation automatisée développant un logiciel applicatif devra suivre un cycle de développement. Nous détaillons ci-contre (*Figure 1*) le cycle en V du développement, processus couramment utilisé et servant de base aux prescriptions du projet de norme FDIS CEI 62061.

Cycle en V théorique

On y distingue les phases suivantes :
• la **spécification système** décrit le système complet dans son ensemble avec les fonctions à réaliser, les performances requises et les contraintes (mécaniques, électriques...),
• la **spécification logicielle** décrit ce que doit faire le logiciel, sans expliciter les moyens correspondants ; on précise ainsi les fonctions logicielles à remplir, les performances logicielles requises et les contraintes,
• la **conception générale** vise à décomposer le logiciel en composants plus simples (définition des fonctions), donc à définir la structure, l'architecture du logiciel,
• la **conception détaillée** fournit la description complète de chaque composant (algorithme, données d'entrée et de sortie),
• la **programmation permet** de traduire la conception détaillée en code du composant cible,
• la phase de **test unitaire** s'assure de la bonne réalisation des composants élémentaires générés lors de la conception détaillée,
• la phase de **test d'intégration** vérifie l'assemblage des composants (par exemple, une fonction faisant appel à des sous-fonctions),
• la phase de t**est de validation** consiste à

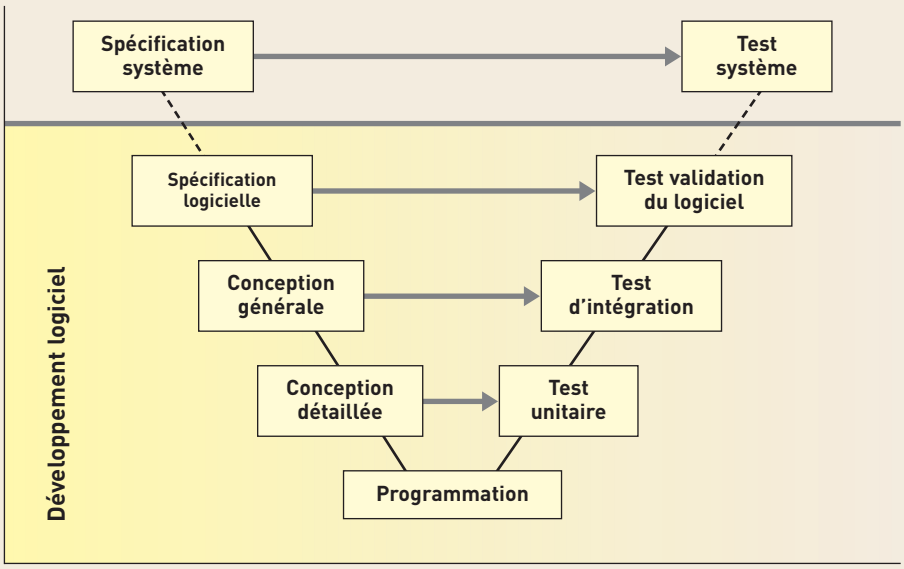

FIGURE 1

Cycle en V du développement
V-shaped development cycle

criptions. Une présentation sous forme d'étapes à atteindre est proposée (de 1 à 3, 3 étant l'étape contenant le niveau de prescriptions le plus complet). Les prescriptions de l'étape 1 doivent être satisfaites avant de pouvoir passer à celles de l'étape 2. Ces prescriptions sont de plus cumulatives, i.e. si l'on est à l'étape 2, il faudra bien sûr toujours respecter celles de l'étape 1. De la même façon, avant de pouvoir aborder l'étape 3, il faudra respecter les prescriptions des étapes 1 et 2.

Dans la mesure où les prescriptions peuvent paraître difficiles à appréhender pour des personnes non familiarisées avec une méthodologie de développement logiciel, des renseignements complémentaires précisent, si possible, des moyens pour satisfaire les prescriptions concernant la conception et le développement du logiciel.

Ce document s'adresse donc aux concepteurs d'installations automatisées ou de machines gérant les fonctions de sécurité à l'aide d'automates programmables dédiés à la sécurité. Ces utilisateurs sont amenés à créer ou modifier, lors de la phase de conception, le logiciel applicatif de leur outil de production. Nous pouvons citer par exemple les équipements automatisés pour la production de série (chaîne d'assemblage dans le domaine de l'automobile, lignes d'impression dans le domaine de l'imprimerie), des machines spéciales, des machines conçues en petite série telles que les presses mécaniques. Les personnes travaillant lors du cycle de production ou le personnel de maintenance intervenant sur l'installation automatisée ne sont pas concernées. L'aspect maintenabilité du logiciel sera toutefois facilité en respectant les recommandations (documentation, suivi des versions...). De même, les fabricants des automatismes utiliseront plutôt des normes telles que la CEI 61508 [6] pour réaliser le développement de l'automate programmable dédié à la sécurité.

vérifier le logiciel dans sa totalité. Afin de s'assurer du respect de la spécification logicielle, on y réalise des tests fonctionnels,

• la phase de **test système** consiste à vérifier le logiciel et le matériel ensembles. Afin de s'assurer du respect de la spécification système, on y réalise aussi des tests fonctionnels.

vis à vis du développement logiciel, pourront dans un premier temps se donner les moyens de respecter la première étape et ensuite, dans le cadre d'un processus d'amélioration continu, tendre vers le suivi des prescriptions de l'étape 2, puis de l'étape 3. Une entreprise ayant déjà en place un processus de développement logiciel respectant les prescriptions de l'étape 1 pourra commencer directement à l'étape 2.

Les prescriptions de l'étape 1 sont en **rouge**, celles de l'étape 2 en **orange** et celles correspondantes à l'étape 3 en **vert**. Ces couleurs sont destinées à retrouver plus facilement les prescriptions dans le texte du document.

Cas de l'automatique : état des lieux

Pour les systèmes automatisés, l'état de l'art actuel utilise souvent un cycle en V simplifié (*Figure 2*).

Ce cycle en V simplifié correspond à ce qui est souvent réalisé par des entreprises, pour des installations automatisées incluant des automates programmables. Pour la réalisation d'installations de sécurité, ce cycle de développement n'est pas suffisant et l'ensemble des prescriptions du projet de norme FDIS CEI 62061 est là pour amener les concepteurs à tendre vers le cycle théorique.

Dans le cadre d'un processus global d'amélioration, nous proposons une hiérarchisation des prescriptions. Les entreprises, en fonction de leur pratique

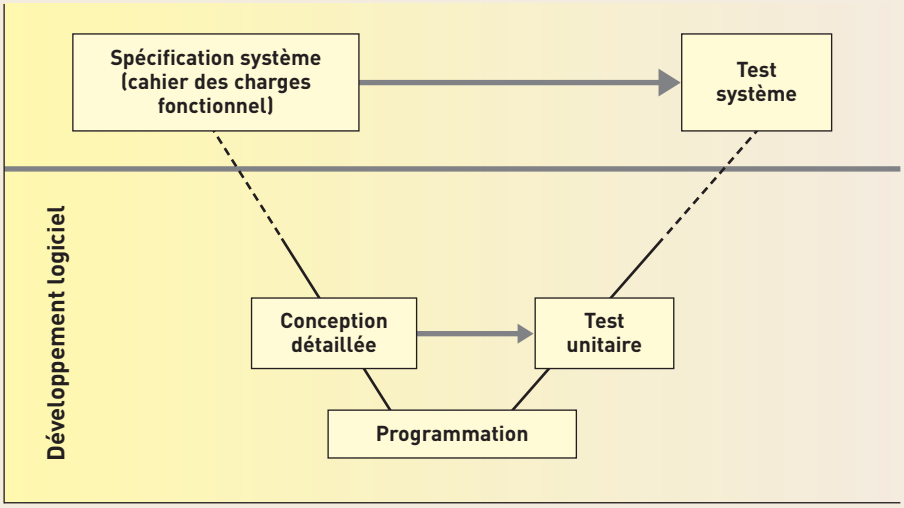

FIGURE 2

Cycle en V simplifié
Simplified V-shaped cycle

1. LA PHASE AMONT DU DÉVELOPPEMENT LOGICIEL : LA SPÉCIFICATION DU LOGICIEL

Données en entrée (a minima):
• Exigences fonctionnelles de la fonction de commande relative à la sécurité,
• Exigences liées à l'architecture,
• Exigences du plan de la sécurité fonctionnelle.

Données en sortie (a minima) :
• Exigences fonctionnelles à assurer par logiciel,
• Spécifications du logiciel,
• Revues (uniquement en étape 2),
• Plan de validation (uniquement en étape 3).

1.1 Hiérarchisation

Prescriptions pour la spécification logicielle	§	Étape		
		1	2	3
Utiliser les exigences fonctionnelles relatives à la fonction de sécurité, les exigences liées à l'architecture et les exigences du plan de la sécurité fonctionnelle	1.2	X		
Rédiger des spécifications détaillées	1.3	x		
Définir l'expression et la structuration des exigences	1.4	x		
Définir de façon pertinente les exigences fonctionnelles de la fonction relative à la sécurité traitées par le logiciel	1.3	x		
Vérifier les spécifications par des revues faites par le développeur	1.4		X	
Utiliser des méthodes/outils pour la formalisation de l'analyse	1.5		X	
Rédiger un plan de validation	1.6			X

1.2 Documents préliminaires à obtenir

La spécification des exigences de sécurité du logiciel sera dérivée :
• **des exigences fonctionnelles de la fonction de commande relative à la sécurité**,
• **des exigences d'architecture** du système de sécurité (une redondance matérielle peut entraîner par exemple une redondance logicielle si on fait appel à un compilateur différent dans chaque partie redondée),
• **des exigences du plan de sécurité fonctionnel**.

Le plan de sécurité fonctionnelle doit définir, entre autre, la stratégie pour le développement du logiciel applicatif. Ce plan, établi lors de la spécification du système, décrit aussi :
• le management des différentes étapes,
• l'identification des personnes responsables des activités de vérification,
• les procédures pour enregistrer et maintenir les informations relatives à la sécurité,
• la stratégie pour l'intégration, la vérification, la validation du logiciel applicatif,
• la stratégie pour la gestion de configuration qui inclut les aspects organisationnels tels que la structure interne de l'équipe,
• le plan de la vérification et de la validation.

1.3 Élaboration des spécifications

Il faut établir **des spécifications détaillées** pour faciliter la conception, la réalisation et l'évaluation du système.

L'expression et la structuration des exigences de sécurité du logiciel doivent être :
• claires, vérifiables, testables, maintenables et fonctionnelles,
• traçables par rapport aux spécifications des exigences de sécurité du Système électronique de contrôle de sécurité,
• non ambiguës.

La définition des **exigences fonctionnelles de la fonction relative à la sécurité** à respecter par le logiciel doit inclure :
• la logique fonctionnelle de tous les blocs associés au sous-système,
• les interfaces d'E/S de chaque bloc fonctionnel,
• le format et la plage de valeurs des données d'E/S et leurs relations avec les blocs fonctionnels,
• les fonctions qui réalisent ou maintiennent un état sûr,
• les fonctions relatives à la détection, la signalisation et à la gestion des défauts des sous-systèmes,
• les fonctions liées au test périodique en fonctionnement normal et hors fonctionnement normal (maintenance par exemple),
• les fonctions qui empêchent la modification du système électronique de contrôle de sécurité sans risque,
• le fonctionnement en terme de performance (précision, exactitude), de temps de réponse et de contrainte de capacité mémoire.

1.4 Revues

Les revues réalisées par le développeur doivent vérifier que les spécifications renseignent sur :
• les fonctions de commande de la sécurité,
• la configuration de l'architecture du système,
• la performance (précision, exactitude), les temps de réponse et les contraintes de capacité mémoire,
• les interfaces avec les équipements et l'opérateur,
• les modes de fonctionnement,
• les tests de diagnostics des dispositifs externes.

1.5 Méthodes et outils

L'utilisation de méthodes/outils pour formaliser l'analyse est recommandée.

Tous les moyens peuvent être employés, en particulier la décomposition du système en bloc fonctionnel lorsqu'il s'agit d'avoir une vue synthétique et modulaire. On peut citer, par exemple, l'utilisation de méthodes telles que SADT.

Les diagrammes de séquence sont des diagrammes représentant, de façon temporelle, le séquencement des différents évènements du système. Ces diagrammes permettent d'avoir une vue dynamique du système.

1.6 Management de la validation

Il faut **rédiger un plan de validation**.

Le plan de validation doit être défini avant de réaliser le logiciel. Sa rédaction doit débuter dès la phase de spécification (préciser les objectifs, la stratégie, les techniques et méthodes, l'organisation et les responsabilités, les moyens nécessaires, l'identification des tests). Les scénarios de test fonctionnels devront par exemple être réalisés lors de cette phase.

2. ARCHITECTURE DU LOGICIEL – CONCEPTION GÉNÉRALE

Données en entrée (a minima):
• Documentation de l'architecture du système,
• Exigences fonctionnelles à assurer par le logiciel.

Données en sortie (a minima) :
• Définition de la structure interne,
• Spécification des composants,
• Spécification des connexions,
• Description de l'ensemble de techniques et mesures (uniquement en étape 2),
• Documentation des scripts de tests d'intégration (uniquement en étape 2),
• Description des dispositifs assurant l'intégrité des données (uniquement en étape 3),
• Procédures pour actions correctives (uniquement en étape 3).

2.1 Hiérarchisation

Prescriptions pour l'architecture du logiciel	§	Étape		
		1	2	3
Utiliser les spécifications de sécurité pour la conception de l'architecture	2.2	X		
Définir la structure interne	2.3	X		
Spécifier les composants	2.3	X		
Spécifier les connexions et interactions entre les composants identifiés	2.3	X		
Décomposer le logiciel applicatif en ensembles d'intégration gérables	2.2		X	
Identifier les modules non utilisés dans la fonction de sécurité	2.3		X	
Mettre à jour la description de l'architecture en cas de modification des prescriptions de sécurité	2.3		X	
Décrire l'ensemble de techniques et mesures de conception du logiciel	2.4		X	
Réaliser les scripts des tests d'intégration	2.6		X	
Préciser l'environnement des tests, les outils utilisés, la configuration des outils et leurs programmes	2.6		X	
Décrire et justifier les dispositifs pour le maintien de l'intégrité des données	2.5			X
Définir les critères d'acceptation des tests	2.6			X
Écrire les procédures pour les actions correctives suite à un test non satisfaisant	2.6			X

2.2 Conception et spécifications

La conception de l'architecture doit être basée sur les spécifications de sécurité du système électronique de contrôle de sécurité, qui tiennent compte des contraintes d'architecture du système et de la conception des sous-systèmes.

Dans le cas où l'analyse préliminaire du système a conduit à une architecture redondante, on peut imaginer avoir un logiciel redondant. De même, on pourra partager les E/S de sorte que toutes les entrées (redondantes) ne soient pas traitées par la même partie logicielle.

Le logiciel doit être décomposé en ensembles d'intégration faciles à gérer. Il faut éviter des modules trop volumineux en nombre de lignes de code, mais aussi éviter l'utilisation de sauts, générateurs de boucle sans fin.

2.3 Description de l'architecture

La description de la conception de l'architecture logicielle doit être complète :

description compréhensible de l'architecture

L'architecture du logiciel définit les composants principaux et les sous-ensembles du logiciel. Un module est un programme ou une partie de logiciel aussi autonome que possible. La construction modulaire présente des avantages sur le plan de la maintenabilité et de la portabilité. Elle facilite la maintenance en factorisant les opérations (les modules communs ne seront exécutés qu'une seule fois). On peut aussi remplacer un module défectueux, après avoir réalisé une analyse d'impact, sans toucher à l'ensemble, ce qui facilite la modification et le test du système.

Par exemple, dans l'outil de programmation de l'automate PILZ du type PSS 3000, on peut utiliser la décomposition sous forme de blocs :
• OB (bloc d'organisation),
• PB (bloc programme),
• FB (bloc fonction),
• SB (bloc standard),
• DB (bloc de données).

Pour les automates Siemens, l'outil de programmation Step 7 propose l'utilisation des blocs suivants :
• les blocs d'organisation OB,
• les fonctions FC, SFC,
• les blocs fonctionnels FB, SFB,
• les blocs de données DB.

On distingue la programmation linéaire, dans laquelle on utilise un seul bloc (d'organisation) où toutes les opérations sont écrites, de la programmation modulaire, dans laquelle les opérations des différentes fonctions sont réparties dans des blocs, chaque bloc contenant le programme nécessaire pour la tâche assignée. Il existe une programmation plus structurée faisant appel à des blocs paramétrables (comme les blocs fonctions SB de PILZ ou FB, FC de Siemens). Ces blocs sont paramétrables et nécessitent, lors de leur appel, le positionnement des paramètres.

On cherche à produire un logiciel sous forme de modules. Il existe des fonctions prédéfinies (modules) dans le logiciel de base fourni avec les API[2] ; ce sont les bibliothèques de fonctions (DFB ou EF avec le logiciel PL7 Pro de Schneider, bloc SB pour le logiciel du PSS de PILZ). L'utilisateur peut aussi créer ses propres fonctions et assurer une modularité de son logiciel.

spécification de tous les composants

spécification des connexions et des interactions entre les composants identifiés,

identification des modules de logiciel inclus dans le système électronique de contrôle de sécurité mais non utilisés dans la fonction de sécurité (effet de bord),

mise à jour en cas de modifications des prescriptions de sécurité (par exemple, changement dans le niveau de sécurité du système). Ceci permettra de s'assurer de la faisabilité de ces modifications.

2.4 Techniques et mesures pour satisfaire les spécifications

L'ensemble de techniques et mesures utilisées à la conception du logiciel applicatif pour satisfaire les spécifications sera décrit et justifié. Il devra être cohérent avec les contraintes vues dans la documentation du système électronique de contrôle de sécurité. On cherche ainsi à assurer :
• la tolérance aux fautes, par exemple en ayant recours à la redondance pour

[2] *API : Automate Programmable Industriel.*

garantir la sécurité malgré la présence de défauts dans un canal ; Est-ce qu'un défaut entraînera la perte de l'application ? Est-ce qu'une redondance logicielle est utilisée pour traiter les entrées ?
• l'évitement de fautes, par exemple en utilisant une programmation modulaire, en exécutant des tests fonctionnels ou encore en réalisant des revues d'inspection du logiciel.

2.5 Intégrité des données

Il faut **décrire et justifier les dispositifs visant à maintenir l'intégrité des données** (E/S, communication, interface, maintenance).

L'intégrité des données s'assure que les données restent intactes tout au long de leur utilisation. La garantir représente la confiance à accorder aux données. L'intégrité des E/S pendant tout le déroulement du programme pourra par exemple être assurée en empêchant de modifier la valeur réelle des Entrées dans le programme applicatif. Une partie de l'intégrité des données (i.e. maintenance) peut être couverte par la gestion de configuration.

D'autre part, afin de travailler sur des entrées stabilisées, il est préférable de prendre la mémoire image des entrées et de ne pas travailler avec les entrées physiques directes (pour s'assurer de la stabilité des entrées pour effectuer le processus de traitement et le positionnement des sorties).

2.6 Tests d'intégration

Il faut documenter les tests d'intégration sur les aspects suivants :
les scripts, qui décrivent chaque test en termes de mode opératoire, de données d'entrées et de résultats attendus,
l'environnement de tests (quels moyens sont utilisés), **outils, configurations des outils et leurs programmes** (version de tous les constituants/composants de l'outil, version du logiciel associé à l'outil si nécessaire),
les critères d'arrêt des tests. La durée des tests et les efforts engagés ne sont, en aucun cas, des critères d'arrêt des tests : aucune garantie sur le logiciel ne peut être établie sur ces seules observations. La diminution du nombre d'erreurs constatées par unité de temps n'est pas non plus un critère d'arrêt ; elle résulte souvent de la difficulté pour l'équipe de tests de trouver de nouveaux tests pour mettre en évidence des erreurs logicielles. La mesure de taux de couverture pourrait être un critère d'arrêt s'il était possible d'évaluer la probabilité de présence d'erreurs non détectées,
les procédures pour actions correctives suite à une non-satisfaction à un test. Dans un souci de reproductibilité (afin de reproduire le bug) et d'équité, il faut définir la procédure à suivre en cas d'échec d'un test pour que toute non-conformité soit bien traitée de la même façon (indépendamment de l'importance du bug), notamment par rapport aux mesures prises pour corriger le défaut.

3. CONCEPTION DÉTAILLÉE DU LOGICIEL APPLICATIF

Données en entrée (a minima) :
• Spécification logicielle des exigences de sécurité,
• Spécifications de l'architecture,

Données en sortie (a minima) :
• Documentation des modules logiciels,
• Documentation des scripts de tests unitaires,
• Documentation de tous les moyens nécessaires à la réalisation des tests (uniquement en étape 2),
• Procédures pour actions correctives (uniquement étape 3).

3.1 Hiérarchisation

Prescriptions pour la conception	§	Étape 1	2	3
Obtenir la spécification logicielle des exigences de sécurité et l'architecture avant le démarrage de la conception détaillée	3.2	X		
Réaliser une production structurée	3.3	X		
Mettre en œuvre un processus d'amélioration de la conception	3.4	X		
Préciser la documentation minimum des modules logiciels	3.5	X		
Documenter les scripts de tests unitaires	3.5	X		
Définir la conception et l'architecture interne des composants autres que des boîtes noire	3.3		X	
Documenter l'environnement de test, les outils utilisés, la configuration des outils et leurs programmes	3.5		X	
Définir les critères d'arrêt des tests	3.5			X
Écrire les procédures pour les actions correctives suite à un test non satisfaisant	3.5			X
Définir les types de test	3.5			X

3.2 Informations préliminaires

En entrée de la phase de conception détaillée, le développeur doit disposer :
des spécifications des exigences de sécurité du logiciel. Ce point permet de s'assurer que le travail à réaliser a été bien compris et formalisé afin d'éviter les non-dits. C'est le cahier des charges formalisé qui sert de base de travail,
de la définition de l'architecture logicielle
• la logique applicative,
• les moyens mis en œuvre pour tolérer les défauts,
• la liste des E/S,
• les modules logiciels génériques ; ce sont les modules en provenance d'autres applications, qui pourront être réutilisés,
• les outils utilisés,
• les procédures pour configurer le logiciel applicatif par rapport au matériel et fournir les fonctionnalités applicatives pour les E/S définies. En général, l'outil de programmation fournit le moyen de configurer le matériel par rapport aux entrées/sorties (configuration des différentes cartes utilisées),
du plan de validation.

3.3 Structure

Il est demandé une **production structurée du logiciel** afin d'obtenir :
• une modularité des fonctions,
• une modularité des E/S,
• une testabilité des fonctions et de la structure interne (il faut en effet tester les modules qui, une fois testés et validés, pourront être réutilisés sans contraintes),
• une modification sûre (traçabilité et explication des fonctions et des contraintes associées). La structuration du logiciel évite d'éventuels effets de bord en cas de modification.

La conception interne et l'architecture interne de tous les composants identifiés autres que des boîtes noires doivent être réalisées à partir de la spécification obtenue lors de la définition de l'architecture logicielle.

3.4 Processus d'amélioration de la conception détaillée

L'amélioration de la conception est basée sur l'utilisation d'un processus itératif de développement. Pour chaque composant majeur défini dans l'architecture logicielle, le perfectionnement de la conception du logiciel doit être basé sur le principe d'un processus de développement itératif/incrémental.

Il sera facilité lors d'évolutions si l'on respecte :
- la correspondance des informations d'entrées/sorties vers les modules logiciels, afin de garantir l'affectation des entrées/sorties et de structurer le logiciel,
- la réalisation des fonctions applicatives (fonctions automates) à partir de fonctions logicielles génériques (modules) et des correspondances des entrées/sorties, dans un souci du respect de l'architecture et de la traçabilité des entrées/sorties.

3.5 Documentation du logiciel

La documentation de chaque module logiciel doit décrire au minimum :
- la personne morale qui écrit le code (soit le nom de la compagnie, soit le nom de l'auteur du logiciel),
- la fonctionnalité du module afin de pouvoir situer le programme dans son contexte,
- la traçabilité aux exigences fonctionnelles de l'applicatif et la traçabilité des fonctions de la bibliothèque (par exemple, l'utilisation de plusieurs modules/bibliothèques dans un logiciel),
- la liste des entrées et sorties,
- la gestion de configuration/version ; il faut lister explicitement les éléments constituants le logiciel. Les ateliers logiciels des fabricants d'automates permettent de générer automatiquement certaines documentations ; il faut s'assurer que cette documentation est suffisante, par exemple qu'elle liste tous les blocs utilisés avec leur numéro de version.

La génération de la documentation des modules s'avère nécessaire dans la mesure où ils sont créés pour une utilisation massive (dupliquée).

L'écriture des tests doit être réalisée lors de la phase de conception. La description des tests unitaires doit contenir :
- **les scripts**,
- **l'environnement de tests** (quels moyens sont utilisés), **outils, configurations des outils et leurs programmes** (version du logiciel applicatif et de tous ses constituants/composants),
- **les critères d'arrêt des tests** ; les recommandations sur ce point sont identiques à celles formulées pour les tests d'intégration du §2.6.,
- **les procédures pour actions correctives suite à une non satisfaction à un test** ; les recommandations sur ce point sont identiques à celles formulées pour les tests d'intégration du §2.6.,
- **le type de tests à exécuter** ; cette information doit apparaître dans le plan de validation. Ces tests consistent en des tests de robustesse, des tests boîte blanche...

4. PROGRAMMATION DU LOGICIEL APPLICATIF

Données en entrée (a minima) :
- Règles de conception,
- Spécifications des modules logiciels.

Données en sortie (a minima) :
- Code,
- Revues du logiciel.

4.1 Hiérarchisation

Prescriptions pour le développement applicatif	§	Étape 1	2	3
Développer un logiciel lisible et compréhensible	4.2	X		
S'assurer que le logiciel est testable	4.2	X		
Respecter les règles de conception	4.2	X		
Réaliser des inspections	4.3	X		
Utiliser des méthodes informelles (revues)	4.3		X	
Utiliser des méthodes formelles (inspection logicielle)	4.3			X

4.2 Caractéristiques du logiciel applicatif

Le code du **logiciel applicatif** développé doit être :
- **lisible**, par exemple grâce à l'indentation des différentes lignes (pas de texte "au kilomètre") ou en suivant des règles de nommage des variables et des étiquettes afin d'obtenir des noms suffisamment clairs,
- **compréhensible** ; on veillera en particulier à la pertinence des commentaires,
- **testable** ; le test du logiciel est une des principales activités destinées à démontrer la conformité d'un logiciel. Il doit donc être facilement réalisable. Par exemple, l'utilisation de pointeurs sous PL7 est plus une technique d'informaticien qui demande une pratique suffisante. La complexité de cette technique, si elle n'est pas correctement maîtrisée, rendra le test plus délicat.

Le code développé devra de plus :
- **satisfaire les règles appropriées de conception** définies lors de la planification de la sécurité ; le respect de ces règles sera vérifié par un processus de revue,
- satisfaire aux exigences appropriées définies pendant la planification de sécurité.

4.3 Moyens de vérification

Les revues de logiciel (**inspections**, **méthodes informelles**, **méthodes formelles**) sont à réaliser pour s'assurer de la conformité :
- à la conception,
- aux règles de codage,
- à la planification de sécurité.

On se situe à ce stade en bas du cycle en V (*Figures 1* et *2*). Aucune phase de tests n'est directement associée à la programmation. Le seul moyen de vérifier la cohérence du code est donc de réaliser des revues.

Les revues du logiciel incluent des inspections du logiciel (on parle alors de revue statique), par exemple le respect des règles de codage sur la base d'un échantillonnage dans le logiciel.

On peut aussi mettre en pratique la technique des "walk-throughs" : on choisit des combinaisons d'entrées et on 'exécute' le logiciel par exemple à partir des listings afin d'en connaître le comportement, ce qui s'apparente à une simulation manuelle.

Il peut aussi être envisageable d'utiliser les méthodes formelles [8]. Ces méthodes visent à transcrire la spécification (en langage naturel) en un langage logique mathématique. La complexité de leur mise en œuvre les réservera à des applications à haut risque ou à des logiciels de taille suffisamment conséquente.

5. TESTS UNITAIRES DU LOGICIEL (TESTS DES MODULES)

Données en entrée (a minima) :
- Scripts de tests,
- Code.

Données en sortie (a minima) :
- Documentation des résultats de tests unitaires,
- Revues du logiciel (uniquement en étape 2).

5.1 Hiérarchisation

Prescriptions pour la vérification des modules explicatifs	§	Étape		
		1	2	3
Vérifier la correspondance des E/S à la fonctionnalité du logiciel par la simulation et le test	5.2	X		
Vérifier les modules par la simulation et le test	5.3	X		
Tester chaque branche des modules	5.3	X		
Tester les données aux limites des modules	5.3	X		
Contrôler que les séquences sont bien implémentées	5.3	X		
Documenter les résultats de tests	5.4	X		
Effectuer des revues concernant la correspondance des E/S	5.2		X	
Réaliser des revues des modules	5.3		X	
Réduire les tests si retour d'expérience positif ou si le module logiciel a déjà été validé/certifié	5.5			X

5.2 Correspondance des entrées/sorties

Il faut s'assurer de la correspondance entre les entrées/sorties et la logique applicative ; ceci passe par la réalisation de :
- **simulations**,
- **tests**,
- **revues**.

Si, par exemple, on utilise une architecture faisant l'acquisition et un traitement sur des entrées, il faut s'assurer que les variables intermédiaires utilisées dans le cadre du traitement sont bien reliées à la bonne logique. On s'assurera ainsi que les variables d'entrées sont elles-mêmes bien reliées à la bonne logique.

5.3 Vérification et tests des modules

La vérification est un processus basé sur l'utilisation de check-lists pour contrôler un produit, sans analyse approfondie.

Les modules logiciel doivent être vérifiés pour s'assurer que la fonction prévue est correctement exécutée et qu'il n'y a pas d'actions fortuites (effet de bords) :
- **simulation**,
- **test**,
- **revue**.

Les revues permettent de détecter les erreurs au plus tôt dans le cycle de développement, donc de gagner du temps et de diminuer les coûts.

La simulation vise à gagner du temps lors de la mise en service. Il se peut toutefois que certaines parties du logiciel ne soient validables que sur site.

Il faut réaliser un script de test pour chaque module afin d'en valider la spécification. Dans ce script, chaque test de validation sera décrit de façon détaillée en termes de mode opératoire, de données d'entrées, de résultats attendus. Les tests doivent convenir au module spécifique testé (unicité des tests). On devra s'assurer que :
- **la branche est testée**,
- **les données aux limites sont testées** ; par exemple, il faut tester les cas de dépassement si des conditions du type >, ≥ sont utilisées. Dans le cas de boucles, il faut vérifier les conditions extrêmes de ces boucles,
- **les séquences** sont bien implémentées.

5.4 Documentation

Il faut **documenter les résultats de test des modules du logiciel applicatif**. Il s'agit de préciser les conditions de réalisation des tests, les tests exécutés, les résultats obtenus, de mentionner les problèmes découverts pendant le test, d'interpréter les résultats et de faire un bilan.

Ces documents pourront être inspectés afin de vérifier que tous les tests prévus ont été réalisés.

5.5 Logiciel éprouvé

Il sera possible de **réduire les tests si on a un retour d'expérience positif ou si le module logiciel a déjà été validé/certifié par ailleurs**. En toute rigueur, si un module est utilisé dans un système et qu'il a déjà été validé par ailleurs, sa réutilisation ne pose pas de problème (au niveau module). Il faudra alors s'intéresser plus particulièrement à la phase d'intégration, notamment à ses interfaces avec les autres modules mais surtout s'assurer que le module utilisé correspond bien à la fonction à réaliser.

6. TESTS D'INTÉGRATION DU LOGICIEL

Données en entrée (a minima) :
- Scripts de tests d'intégration.

Données en sortie (a minima) :
- Documentation des résultats de tests d'intégration,
- Documentation des raisons d'un test non conforme,
- Documentation de l'analyse d'impact et des actions correctives,
- Documentation de la reconception et re-vérification en cas de modifications.

Le test d'intégration du logiciel consiste à tester un ensemble de composants qui viennent d'être assemblés, via l'activation de ses fonctionnalités. On peut citer comme erreurs détectables par des tests d'intégration :
- la mauvaise intégration des différents composants logiciels (bibliothèque),
- le mauvais séquencement des modules,

• les erreurs de compilation (mémoire insuffisante).
• Il faut rédiger les scripts de tels tests à partir de la documentation (spécification) correspondante.

6.1 Hiérarchisation

Prescriptions pour les tests d'intégration du logiciel	§	Étape 1	2	3
Vérifier les interactions entre les composants logiciels	6.2		X	
Documenter les résultats de tests	6.3		X	
Documenter les raisons d'un test non conforme et les actions correctives	6.3		X	
Réaliser une analyse d'impact en cas de modification du logiciel	6.4		X	
Réaliser les activités de re-vérification et re-conception en cas de modification du logiciel	6.4		X	
Documenter les objectifs à atteindre et les critères d'arrêt des tests	6.3			X

6.2 Interactions

Dans le but de valider la conception générale, il faut **vérifier les interactions entre les différents composants logiciels** (logiciel embarqué inclus) et la non-exécution d'autres fonctions pouvant affecter la fonction de sécurité. Il ne doit pas y avoir d'effets de bord.

Si une programmation modulaire et structurée est utilisée, il faut s'assurer que les composants logiciels agissent correctement l'un sur l'autre et n'exécutent pas des fonctions non prévues qui pourraient compromettre la fonction de sécurité (effets de bord).

6.3 Documentation

Les résultats des tests doivent être documentés pour s'assurer de leur exécution (i.e. dans le cas d'un audit) et pour avoir une traçabilité des tests effectués.

Il faut **documenter les objectifs et les critères d'arrêt des tests**.

Les raisons d'un test non conforme et les actions correctives en résultant doivent être documentées dans les résultats de test (pour garder l'historique des problèmes sous forme par exemple de base de données).

6.4 Analyse d'impact

En cas de modifications du logiciel, il faut effectuer une analyse d'impact donnant :
• les modules logiciels impactés,
• les activités nécessaires de **re-vérification et de re-conception** pour traiter et maîtriser les conséquences de la modification.

Cette analyse vise à évaluer l'importance des modifications ainsi qu'à prévoir les délais et coûts.

Il faut ensuite remettre à jour toute la documentation concernée par cette modification (spécification et test notamment).

7. TEST DU SYSTÈME DE SÉCURITÉ

Données en entrée (a minima) :
• Scripts de tests système.

Données en sortie (a minima) :
• Documentation des résultats de tests système,
• Documentation de l'analyse d'impact,
• Documentation des actions correctives,
• Documentation de la reconception et re-vérification en cas de modifications,
• Documentation de la version du système de sécurité, de la version des spécifications,
• Documentation des moyens nécessaires pour réaliser les tests (uniquement en étape 2).

Ces tests recouvrent les tests de validation du logiciel et les tests d'intégration du système (tests du logiciel avec le matériel cible). Ils visent à vérifier la fonctionnalité décrite dans la spécification haut niveau. Il faut pour cela élaborer des tests à partir de la spécification en langage naturel et noter les résultats attendus de ces tests. Lors de l'exécution des tests, on note le résultat obtenu. L'échec d'un test d'intégration du système peut remettre en cause le développement complet du système. Il peut ainsi être utile, pour des systèmes très complexes, de procéder à du maquettage et valider au plus tôt les solutions retenues. L'utilisation de la simulation est recommandée afin de ne pas valider directement l'installation sur site, mais par exemple après passage de tests avec un outil de simulation de partie opérative.

Un test sur site sera de toute façon nécessaire afin de valider par exemple les temps de réponse ou le câblage.

On cherche à valider le système à partir de l'identification des fonctionnalités requises. On peut prévoir :
• des tests aux limites,
• des tests hors limites, pour voir si ces cas hors limites sont signalés, voire récupérés,
• les tests nominaux.

En cas d'évolution du logiciel, on prévoira des tests de non-régression, servant à vérifier le système par rapport à des fonctionnalités à conserver d'une version à l'autre.

La réalisation de tests fonctionnels d'intégration système se base sur la documentation des scripts de tests en fonction des spécifications : cas de tests avec le résultat attendu pour ces tests. Lors du passage des tests, on consigne l'état observé et on analyse les éventuelles différences entre le résultat attendu et le résultat observé.

7.1 Hiérarchisation

Prescriptions pour les tests d'intégration du système	§	Étape 1	2	3
Réaliser des tests d'intégration système	7.2	X		
Tester la compatibilité du logiciel applicatif avec le matériel	7.3	X		
S'assurer du séquencement et de la synchronisation des modules	7.4	X		
Documenter la version des spécifications, la version du SRECS[3], les résultats des tests	7.5	X		
Effectuer une analyse d'impact en cas d'anomalies découvertes lors des tests	7.5	X		
Documenter les actions correctives en cas d'anomalies découvertes lors des tests	7.5	X		
Réaliser des tests fonctionnels pour vérifier l'intégrité	7.6	X		
Réaliser des tests dynamiques pour vérifier l'intégrité	7.6	X		
Documenter les critères d'acceptation, les outils et équipements utilisés, les anomalies entre résultats prévus et réels	7.5		X	
Documenter les données de calibrage	7.5			X

[3] *SRECS : Safety Related Electrical Control System ; système de commande électrique d'une machine dont la défaillance peut provoquer un accroissement immédiat du risque.*

7.2 Accord avec la conception

L'intégration du logiciel applicatif dans le SRECS sera vérifiée **par la réalisation de tests système** écrits lors de la spécification du système et non pas à partir des codes source du programme ou du logiciel applicatif. Ces tests s'assureront d'une part que les fonctions décrites dans les spécifications sont bien exécutées et d'autre part que tous les modules interagissent correctement pour exécuter leur fonction prévue et uniquement ces fonctions (pas d'effet de bord).

7.3 Compatibilité du matériel et du logiciel applicatif

L'intégration du logiciel applicatif dans le SRECS doit être testée en assurant la **compatibilité du logiciel applicatif avec le matériel** de telle sorte que les conditions fonctionnelles et de sécurité soient satisfaites.

Jusqu'à présent, dans tous les tests précédents, le système n'a pas été testé avec sa configuration matérielle. Il n'y a eu que des tests logiciels (unitaires et d'intégration du logiciel). Il faut s'assurer que le logiciel est compatible avec le matériel ; ceci passe par exemple par la vérification du câblage physique et de la validation des entrées/sorties correspondantes.

7.4 Séquencement

Il faut s'assurer que le **séquencement** des actions réalisées par le programme est bien réalisé et qu'il n'existe pas de conditions critiques de **synchronisation**.

Par exemple, dans le cadre d'une machine d'usinage avec plusieurs axes, il faut s'assurer que chaque axe a fini son travail avant d'effectuer un nouveau déplacement sinon il y a un risque d'accident. De même, dans le cas de systèmes automatisés constitués en îlot, il faut s'assurer qu'il n'y a pas de désynchronisme entre les îlots.

7.5 Documentation

Pendant le test, **on documentera** :
- **la version des spécifications** ; tous les documents relatifs au système de sécurité en cours de validation seront listés, en s'aidant pour cela du référentiel mis en place pour la gestion des versions. On peut par exemple réaliser une table listant la version de tous les documents utilisés pour la spécification du système,
- **les critères pour l'acceptation** ; il faut définir les critères qui permettent de statuer sur le système de sécurité,
- **la version du SCRECS**, si plusieurs itérations sont nécessaires pour la conception de ce système,
- **les outils et les équipements utilisés** (en incluant les **données de calibrage**), qui précisent avec quels moyens les tests ont été réalisés. Les données de calibrage consistent à s'assurer que le matériel utilisé pour la validation a bien été étalonné. Il faut alors garder une trace de cet étalonnage,
- **les résultats** de chaque test, dans le but d'avoir un document complet montrant que les tests ont bien été exécutés et renseignés,
- **les anomalies entre les résultats prévus et réels**, lorsqu'il y a un écart entre le résultat attendu (décrit à partir du cahier des charges fonctionnel) et le résultat observé. On peut par exemple mettre en place une base de données des anomalies détectées lors du passage des tests.

De plus, en cas d'anomalies, il faut effectuer une **analyse d'impact** et décider de la suite du test. **Les actions correctives doivent être documentées**.

7.6 Tests de vérification du système

Il faut effectuer des **tests fonctionnels** (boîte noire) : des données caractéristiques du fonctionnement sont appliquées et les sorties (réponses du système) sont comparées avec les valeurs attendues de la spécification. Le test boîte noire consiste à vérifier les spécifications du logiciel sans se préoccuper de son contenu. Il faut choisir des données caractéristiques du fonctionnement du système, ajuster les entrées en conséquence et vérifier que les réponses du système (sorties) sont bien celles attendues et définies dans la spécification.

Il faut effectuer des **tests dynamiques** : comportement dynamique en conditions réelles de fonctionnement (couverture des spécifications, évaluation de l'utilité et de la robustesse du système de sécurité). On cherche ici à vérifier la couverture des spécifications, on évalue aussi la performance du système de sécurité. On peut aussi réaliser un test de robustesse qui vise à déterminer le comportement du système face à des excès de charge, en s'assurant par exemple qu'il traite toujours les demandes de mise en sécurité ; on teste ainsi le comportement sous forte contrainte.

8. GESTION DES VERSIONS - GESTION DE CONFIGURATION

8.1 Hiérarchisation

Prescriptions pour la gestion de configuration/version	§	Étape		
		1	2	3
Mettre en place un système de référence	8.3	X		
Effectuer une analyse d'impact des modifications	8.5	X		
Identifier la configuration logicielle	8.6	X		
Archiver la version définitive du logiciel et la documentation	8.6	X		
Assurer la traçabilité	8.4		X	
Rédiger des procédures de modification	8.5		X	
Élaborer un plan de gestion de la configuration	8.2			X
Prévoir des revues des modifications	8.5			X

8.2 Plan de gestion de la configuration

Un plan de sécurité fonctionnelle relatif au système global développé doit être élaboré. Ce plan doit inclure toute la stratégie du cycle de développement du logiciel (développement, intégration et validation/vérification).

À partir de ce plan de sécurité fonctionnelle, il peut être utile de créer un **plan de gestion de la configuration**. On souhaite ainsi garantir que toutes les mesures nécessaires ont été prises pour démontrer que les exigences de sécurité du logiciel sont satisfaites.

Le plan de gestion de la configuration est un document qui décrit les méthodes mises en œuvre pour assurer le suivi des versions. Il fournit :
- les règles de création de la documentation,
- les règles d'identification des documents et logiciels produits,

• les règles d'incrémentation des documents afin d'en suivre les évolutions,
• les processus organisationnels mis en place pour développer le logiciel et pour assurer le suivi des modifications,
• les règles donnant la liste des documents fournis au client.

8.3 Système de référence

Il faut créer un **système de référence**, qui identifiera de façon unique les documents et le code, permettant ainsi un suivi des versions pour référencer et gérer tous les documents et éléments logiciels. Il faut référencer :
• les exigences et les analyses de sécurité,
• tous les documents de spécification et de conception,
• les modules de code source (réalisés et importés),
• les tests (cas de tests et résultats),
• les modifications (demandes clients et demandes internes, analyse, décision),
• les outils et environnements de développement utilisés (version de l'O/S, compilateur...).

8.4 Traçabilité

Il faut assurer la **traçabilité** des versions du logiciel, des composants logiciels s'y rattachant, des documents existants, des tests, des modifications et de tout autre document utilisé lors du développement du logiciel.

Dans un souci de maintenabilité, on pourra réaliser un tableau donnant, pour un composant logiciel, les différents documents (test, spécification...) s'y attachant. On parle alors de X-check (cross check ou référence croisée) entre le test et la spécification.

8.5 Modifications du logiciel

Toute modification du logiciel doit être précédée par une **analyse d'impact**.

L'analyse d'impact doit être réalisée avant de commencer les modifications dans le logiciel afin de cerner tous les modules qui seront impactés et de déterminer toutes les conséquences de la modification (par exemple en terme de délai, de coût, mais aussi de sécurité).

Il faut rédiger et appliquer les procédures de modification du logiciel pour :
• empêcher toute modification non autorisée,
• spécifier les actions à entreprendre suite à la détection d'un problème en test ou à une demande client,
• lister les parties modifiées (dans les spécifications) et le lien par rapport au cahier des charges.

Il faut **prévoir des revues des modifications**.

8.6 Statut de la configuration/version du logiciel et de la livraison logicielle

Il faut **identifier la version des différents éléments** constituants les différentes livraisons du logiciel, par exemple dans l'optique de pouvoir faire un audit, mais surtout pour pouvoir régénérer le code à tout moment.

Le statut de la version logicielle livrée permet de connaître l'état des tests associés à cette livraison.

Il faut aussi **archiver la version définitive** livrée **du logiciel** et **toute la documentation** afin de permettre la maintenance et la modification du logiciel durant l'exploitation de la machine.

9. OUTILS DE SUPPORT, MANUEL UTILISATEUR ET LANGAGES

Cette partie s'applique à tous les outils utilisés pour le développement logiciel, tant les ateliers logiciels servant à la programmation de l'automate que les outils spécifiques de simulation, de test, de gestion de configuration/version ou de suivi de documentation.

9.1 Hiérarchisation

Prescriptions pour les outils, le langage et le manuel utilisateur	§	Étape 1	2	3
S'assurer de la pérennité des outils	9.2	X		
S'assurer de la pertinence des outils	9.2	X		
Réaliser des procédures pour la pratique de la gestion de configuration/version	9.4	X		
Insister sur des procédures qui proscrivent les pratiques peu sûres	9.4	X		
Vérifier les caractéristiques du langage applicatif	9.3		X	
Inclure dans les procédures les contrôles pour la détection des erreurs de configuration/version	9.4		X	
Rédiger des procédures pour la documentation du logiciel	9.4		X	

9.2 Choix des outils

Le choix des outils (langage de programmation, gestion de configuration/version, simulation, test) utilisés lors du cycle de développement doit garantir :
• **la pérennité des outils** pendant la vie entière du projet,
• **la pertinence des outils**.

Le choix des outils de programmation est rarement laissé à l'initiative du concepteur du logiciel applicatif. Il faut en fait utiliser les outils commercialisés avec les automates (type STEP 7, PL7).

Par contre, les outils de gestion de configuration, de simulation et de test sont en général à la charge du concepteur. Les outils fournis par les constructeurs d'automate se contentent de sauvegarder un fichier sur le disque dur du PC. Il n'y a pas de gestion de configuration proposée et des outils complémentaires peuvent être nécessaires.

La version des outils de programmation est liée à l'automate utilisé. Il faut donc s'assurer de garder une version de ces outils, ceci afin de pouvoir modifier, regénérer le logiciel pendant toute la durée d'exploitation de la machine.

De même, les outils éventuellement utilisés pour la simulation ou le test devront être pérennes pendant toute la phase d'exploitation de la machine (pour être capable de revalider le logiciel en cas de modification de sa fonctionnalité).

9.3 Langage applicatif ou langage du programme automate

En règle générale, le choix du langage applicatif est lié à l'automate utilisé. C'est le constructeur de l'automate qui

fournit le logiciel qui impose le type de langage applicatif.

Les **caractéristiques du langage applicatif** choisi doivent être les suivantes :
• le compilateur/traducteur générant le logiciel applicatif a du être évalué afin de s'assurer qu'il remplit bien sa mission. Cette évaluation vise à s'assurer des garanties de bon fonctionnement. Si un certificat de validation n'est pas disponible, on pourra, par exemple, se baser soit sur le retour d'expérience de logiciels déjà en exploitation, soit sur le retour d'expérience du fournisseur du langage,
• le langage est complètement et clairement défini. On souhaite ainsi s'assurer de la maturité du langage (vérifier par exemple qu'il existe une norme internationale le définissant telle que la CEI 61131-3 [9] pour les langages automates),
• le langage correspond aux caractéristiques de l'application. On s'assure de l'adéquation du langage avec l'application (taille, type) à développer. Par exemple, la capacité à implanter le modèle de données (pointeurs par exemple) et aux types représentés (TOR, analogique...),
• le langage a la faculté de détecter des erreurs de programmation. Cette mesure est couverte par l'outil de programmation par exemple pour les erreurs de syntaxe au niveau de l'écriture des instructions ou la détection d'erreurs lors de la compilation/transfert du fichier de la console vers l'automate (notamment pour le langage par instruction),
• le langage contient des caractéristiques correspondant à la méthode de conception (par exemple, possibilité d'utiliser la modularité).

9.1 Procédures pour l'utilisation du langage applicatif

Il faut écrire des **procédures** pour l'utilisation de l'atelier de développement logiciel qui :
• décrivent la bonne **pratique de gestion de configuration/version** ; il faut ici insister sur la nécessité d'effectuer une gestion de la configuration du logiciel,
• **proscrivent les pratiques peu sûres** ; par exemple les caractéristiques non définies du langage (cas de l'utilisation d'instruction non maîtrisée par le développeur, ce qui ne devrait pas arriver si la documentation de l'atelier logiciel est consistante) ou conception non structurée,
• décrivent **les contrôles pour la détection des erreurs de configuration/version** ; il faut prévoir les moyens destinés à s'assurer que le logiciel créé contient les bons éléments. Si par exemple, le logiciel est constitué de modules, il faut s'assurer par contrôle que les différents modules sont mis dans le logiciel et avec la bonne version,
• décrivent **les procédures pour la documentation du logiciel**.

CONCLUSION

La conception du logiciel applicatif d'un système de contrôle/commande d'une application de sécurité est une activité qui peut être génératrice d'erreurs et donc de risques pour l'opérateur intervenant sur une machine.

Afin de limiter ces erreurs, il est nécessaire de suivre une méthodologie de développement. Nous avons présenté ici les prescriptions du projet de norme FDIS CEI 62061 que nous avons hiérarchisé selon 3 étapes afin que les concepteurs se rapprochent du cycle en V théorique (*Figure 1*). Le respect de ces prescriptions est un des moyens permettant de s'assurer que le logiciel développé répond bien à ses spécifications.

GLOSSAIRE

Architecture : ensemble de structures comprenant chacune des composants, les propriétés extérieurement visibles de ces composants et les relations qu'ils entretiennent.

APIdS : Automate Programmable Industriel dédié à la Sécurité.

Critère d'arrêt : il s'agit de répondre à la question « Est-ce que les tests réalisés sont suffisants pour évaluer le logiciel ? ». L'arrêt des tests se fait suite à l'analyse des évènements redoutés du système et des aspects qui ne sont pas couverts par aucune des étapes de test. On en déduit si le risque lié à cette non-couverture est acceptable et conforme aux prescriptions applicables.

E/S : Entrées/Sorties.

Gestion de configuration : ensemble d'activités (manuelles et automatisées) permettant d'identifier et de définir tous les constituants d'un produit logiciel, ainsi que leurs relations entre eux. Elles consistent à maîtriser les évolutions du produit logiciel durant son cycle de vie, suivre l'état d'application de ces évolutions, archiver chacun des états successifs et vérifier que chacun de ces états est complet et cohérent.

Itération – Incrémental : à partir d'une phase donnée du cycle de développement, le processus est itéré plusieurs fois. Chaque incrément correspond à un logiciel opérationnel s'approchant à chaque fois davantage du produit final par adjonction de fonctionnalités.

Intégrité : L'intégrité des données vise à s'assurer que les données restent intactes tout au long de leur utilisation. L'intégrité du système assure que le système n'est pas modifié.

Logiciel applicatif : logiciel développé et validé par l'utilisateur de l'APIdS, appelé aussi programme automate.

Logique applicative : terme désignant la fonctionnalité que le logiciel applicatif doit réaliser.

Logiciel embarqué : logiciel réalisé par le constructeur de l'automate, non accessible à l'utilisateur de l'automate programmable. Analogie possible avec le système d'exploitation d'un ordinateur de bureau.

Module : un module est une partie de programme (sous-programme, tâche) aussi autonome que possible. Un module peut être manipulé séparément (sauvegarde, duplication, effacement...).

Sécurité fonctionnelle (définition de la FDIS CEI 62061) : sous-ensemble de la sécurité globale se rapportant à la machine et au système de commande de la machine qui dépend du fonctionnement correct des SRECS, des systèmes relatifs à la sécurité basés sur une autre technologie et des dispositifs externes de réduction de risque.

Reçu le : 21/07/2004
Accepté le : 17/09/2004

BIBLIOGRAPHIE

[1] Ministère de l'Emploi et de la Solidarité. Note relative à l'acceptation de certains automates programmables pour gérer des fonctions de sécurité sur machines. Mai 1998, *e23/APIDS/26 mai 1998*, 5 p.

[2] *Gestion des fonctions de sécurité par automate programmable dédié à la sécurité* - D. DEI-SVALDI, M. KNEPPERT, NST 224, INRS, septembre 2002.

[3] *Logiciels applicatifs relatifs à la sécurité – Étude des problèmes liés à leur exploitation* C. NEUGNOT, M. KNEPPERT, Cahiers de Notes Documentaires N° 187 , 2ème trimestre 2002.

[4] FDIS CEI 62061. *Sécurité des machines - Sécurité fonctionnelle des systèmes de contrôle électriques/électroniques/électroniques programmables*. Version FDIS, 2004, 95 p.

[5] Sécurité des systèmes de commande de machines – Le point sur la normalisation P. CHARPENTIER, revue REE n°2, février 2002

[6] CEI 61508. Sûreté fonctionnelle : systèmes relatifs à la sûreté. Partie 1 à 7, 1998.

[7] *Précis de génie logiciel* - M.C. GAUDEL, B. MARRE, F. SCHLIENGER, G. BERNOT, MASSON, 1996, 142 p.

[8] Utilisation des méthodes formelles en sécurité des machines – Pascal LAMY, Jean-Christophe BLAISE - HST à paraître.

[9] CEI 61131-3 *Automates programmables* – Partie 3 : *Langages de programmation*. 1993, 411 p.