Agilité et avionique

"Monteriez-vous à bord d'un avion dont des logiciels de vol sont écrits par des praticiens de l'eXtreme-Progamming?"

Emmanuel CHENU
emmanuel.chenu@gmail.com

L'objectif de ce retour d'expérience est de faire découvrir un secteur de l'industrie, l'avionique, où l'Agilité à réussi à percer. L'article présente ce que le développement Agile apporte à l'avionique et comment nous avons procédé pour introduire cette démarche dans un contexte industriel à priori peu réceptif à ces pratiques. Des faits seront exposés pour vous aider à répondre à question suivante: "Monteriez-vous à bord d'un avion dont des logiciels de vol sont écrits par des praticiens de l'eXtreme-Progamming?". La présentation évoquera la progression vers l'Agilité, les obstacles surmontés et ceux qui résistent. Elle s'adresse à un public déjà familiarisé avec la démarche Agile et l'eXtreme-Programming [XP] en particulier.

1. CONTEXTE INDUSTRIEL DE L'AVIONIQUE

Secteur

Nous développons des équipements de navigation embarqués sur aéronef, tels que des récepteurs GPS, des gestionnaires de vol, des centrales inertielles et des centrales anémobarométriques. Il s'agit d'équipements pour lesquels **une panne impacte la sécurité du vol**. Dans ces produits, une part importante des fonctionnalités est allouée à des **logiciels temps-réel embarqués**. Ces équipements et leurs logiciels doivent être **certifiés** par un organisme spécialisé pour assurer la sécurité du vol.
Le marché de l'avionique est essentiellement partagé par de grands groupes industriels à cause de coûts de développement élevés et de petites séries produites.

Contraintes absolues

En plus de certaines difficultés habituelles du développement logiciel, ce secteur est confronté aux contraintes de **sûreté de fonctionnement** et du **temps-réel embarqué**.

La certification pour vol

Un niveau de criticité est attribué à un logiciel avionique en fonction de l'impact que peut avoir une panne sur la sécurité du vol. Le document **RTCA DO-178B** de 1992 donne les objectifs à tenir lors du développement pour certifier un logiciel pour vol. Plus le niveau de criticité est élevé, plus les objectifs sont nombreux et contraignants. Ils portent sur les processus, leurs activités, les document et les preuves à produire. Entre autres, il est demandé de vérifier le logiciel dans son environnement opérationnel, c'est à dire sur le matériel cible [HW].
La démarche de développement est pilotée par les exigences, que cela soit pour la construction du logiciel *(traçabilité)* ou pour sa vérification *(couverture par les tests)*.
La certification est obtenue après une série d'audits exigeant la fourniture de preuves des activités menées pour atteindre les objectifs.
La DO-178B ne décrit que les objectifs à tenir - pas la manière de les tenir. Ainsi, il y a de la flexibilité dans la manière de réaliser un développement conforme DO-178B. Par exemple, la norme laisse le choix du cycle de vie en laissant explicitement la porte ouverte au cycle de vie itératif et incrémental. Néanmoins, puisque les objectifs concernent beaucoup la planification, les processus, les activités et les preuves des travaux, le cycle en V a historiquement été la manière intuitive de réaliser un développement conforme DO-178B.

Le logiciel temps-réel embarqué

Le logiciel s'exécute sur un HW spécifique avec un système opératoire [OS] spécifique. Le compilateur générant le code pour le processeur cible n'a pas toujours le même comportement que celui pour le processeur de développement. Le code n'a pas toujours le même comportement sur la cible avec l'OS spécifique que sur la machine de développement.
Le HW et l'OS sont souvent développés en parallèle du SW. Ils sont alors disponibles tard dans le

projet. Lorsque le HW est enfin disponible, c'est souvent en très petites quantités *(car en prototypage ou pour des raisons de coût)*.
Aussi, nous sommes confrontés aux problèmes temps-réel, de concurrence et de ressources limitées *(CPU comme mémoire)*. Enfin, la carte cible est souvent un environnement peu ergonomique en terme d'outils de développement, particulièrement pour les aspects tests et automatisation.

Culture
Les applications sont spécifiques à chaque client, ce qui fait que le modèle industriel semble intuitivement proche de l'artisanat. Mais, l'héritage culturel des standards **STD-2167**, la philosophie de la **DO-178B** et la pratique du **CMMi** ont instauré le paradigme de la production de masse avec son modèle de développement prédictif et le cycle de vie en V. Dans cet environnement, nous constatons que sont privilégiés:

- **les processus détaillés et les outils** plutôt que la communication;
- **les contrats** plutôt que la collaboration*; (avec les clients comme avec les fournisseurs)*
- **la documentation exhaustive** plutôt que le logiciel exécutable et fonctionnel;
- **la planification prédictive** plutôt que l'adaptation empirique et continue.

Néanmoins, la démarche Lean est en train de se mettre en place en développement. Ce changement est essentiellement tiré par le développement logiciel. Cependant, comme nous travaillons dans un domaine où le logiciel n'est pas considéré comme le cœur métier *(qui reste l'aéronautique)* cela ralentit l'adoption de cette démarche.

Contraintes dérivées
Le niveau 3 du CMMi est exigée par certains clients. Dans une culture déjà marquée par la planification prédictive, la documentation, la standardisation, la séparation des tâches et centrée sur les activités, le CMMi trouve un environnement de choix pour se déployer. Au niveau 3, le modèle insiste entre autre sur la standardisation des pratiques, avec ce que cela apporte de formalisme et de perte de flexibilité.
Enfin, la sous-traitance au forfait est souvent imposée dès le devis et le travail est parfois réparti sur plusieurs sites.

Atouts
L'avionique est aussi un secteur avec de nombreux atouts.
Le volume des développements et les ressources aux profils spécialisés font qu'il est possible de travailler en équipe pluridisciplinaire. De plus, localement, les managers accordent une grande autonomie aux équipes pour leur organisation et l'amélioration continue.
L'entreprise développe des lignes de produits stables. Il y a donc possibilité de capitaliser sur les projets et de réutiliser. Les ressources sont stables, ce qui permet de capitaliser sur les gens.
Enfin, les contraintes de sûreté de fonctionnement font qu'il existe une vraie culture de la qualité du produit, de la rigueur et de la discipline dans la démarche de développement.

2. PROBLEMES RECURRENTS DE L'AVIONIQUE ET APPORTS DE LA DEMARCHE AGILE

Il est souvent dit que les démarches Agiles tel que XP sont adaptées aux projets marqués par des inconnues significatives. Des algorithmes complexes et nouveaux, des exigences de performance ambitieuses, un HW et un OS disponibles tard dans le projet sont des inconnues classiques des projets avioniques. Plus particulièrement, XP aide à résoudre les problèmes suivants:

Problèmes de processus
L'héritage des normes xxx-STD-2167 et leur tradition de cycle en V se traduit *(comme ailleurs)* par des développements plus coûteux et longs que prévus, avec perte de visibilité sur l'avancement et la détection souvent tardive et à posteriori de défauts. Certains problèmes critiques à notre secteur se détectent tard, comme des performances non-tenues en consommation mémoire et en charge CPU.
Outre la réponse classique à ce problème classique, la démarche itérative et incrémentale permet de surveiller tôt et continuellement l'évolution des performances.

Problèmes liés au temps-réel embarqué

Les applications SW sont couplées à un HW et un OS spécifiques, disponibles tard et en quantité limitée. Donc, l'intégration sur cible est naturellement du type "big-bang" et tardive. Cette intégration sur processeur cible reste le goulot d'étranglement classique des projets *(l'entrée du tunnel)*. L'activité de test est alors peu optimisée puisque essentiellement composée de séances de «debug» sur cible. De plus, il est regrettable de devoir vérifier des algorithmes métier complexes sur une cible peu ergonomique.

Le Test-Driven Development [TDD] amène à découpler le code et donc à réduire et isoler les dépendances vers l'OS et le HW. Grâce à cela, il devient possible de conduire des tests sur la machine de développement avec des simulacres de HW et d'OS. Ainsi, le code est testé, et en grande partie intégré bien avant la disponibilité du HW cible, par des tests automatisés faisant intervenir des données d'entrée et attendues provenant de simulations produites par des experts de l'avionique. En fait, ne pas disposer du HW et de l'OS est un atout, car cela force à clairement penser les dépendances et séparer les problèmes. Le TDD libère le développeur de problèmes classiques pour qu'il puisse concentrer une partie de ses compétences spécifiques au temps-réel, à la concurrence et aux interfaces avec le HW. Ainsi grâce à la confiance que le développeur a en son logiciel développé dans cette démarche, il sait que les problèmes qu'il détectera sur la cible ne concerneront plus que les aspects temps-réel, concurrence et interface avec le HW. Les autres problèmes peuvent être reproduits et corrigés sur une machine de développement.

Problèmes de flexibilité

Il est impossible de réutiliser du code couplé au HW et à l'OS et difficile de porter une telle application pour une autre cible et un autre OS.

Comme le TDD amène à découpler le code fonctionnel du HW et de l'OS, il devient possible de réutiliser et porter les applications pour d'autres environnements. De plus, grâce une gestion de versions avec intégration continue multiprojets, il est possible de construire des logiciels dont 30% du code est réutilisé entre applications de gammes différentes et 60% entre produits d'une même gamme.

Problèmes de ressources humaines

Nous sommes également confrontés à des problèmes de ressources humaines.

Il est difficile de trouver des développeurs ayant une expérience du génie logiciel, de l'embarqué, et de l'avionique. De plus, les équipes perdent souvent en motivation car les projets sont longs et ponctués par de lourdes activités de documentation, de revue et de traçabilité. Ceci est accentué par la culture de production de masse et le CMMi qui visent à rendre les individus interchangeables en séparant penser et faire.

XP aide à pallier au manque de développeurs par la formation accélérée dans le cadre du travail en binômes, des équipes pluridisciplinaires et de l'espace partagé. De plus, la démarche motive par l'importance accordée aux individus, à l'esprit d'équipe, à l'auto-organisation et à l'appréciation continue des résultats obtenus.

La sûreté de fonctionnement

La grande préoccupation de l'avionique est la sûreté de fonctionnement. Un audit de certification va chercher à s'assurer que l'application ne contient que du code répondant à des besoins opérationnels. La prise en compte de tous les besoins opérationnels et le code devront être intégralement vérifiés par des tests. Aussi, les produits *(exigences, architecture, code, tests...)* devront être vérifiés par des revues.

Avec la construction incrémentale pilotée par des scénarios opérationnels, une si grande importance accordée aux tests systématiques de niveau recette et développeur, une relecture en continue en binômes du code et des tests, et l'intransigeance sur la qualité du logiciel, XP est une démarche appropriée pour assurer la sûreté de fonctionnement.

De plus, la disponibilité tôt et en continue d'un logiciel fonctionnel et la transparence sur l'avancement et les difficultés rassurent les certificateurs sur le fait que les développeurs maîtrisent une démarche de travail rigoureuse et disciplinée.

Néanmoins, sans adaptation, la démarche ne propose pas le niveau de formalisme requis pour les audits de certification.

3. DES ADAPTATIONS SONT NECESSAIRES

Adaptations de l'eXtreme-Programming

Nous avons une pratique de XP adaptée à notre environnement. Nous n'avons pas suivi la recommandation *"Do all XP before you customize it*". Les valeurs et les principes restent parfaitement compatibles de notre secteur. C'est au niveau des pratiques qu'il a fallu faire des adaptations.

Environ 70% du temps, l'équipe travaille en **binômes** qui permutent. Le reste du temps est réservé à du travail individuel comme des revues, la traçabilité et la documentation. Malheureusement, en plus des revues informelles en binômes, la DO178B et le CMMi nous imposent de mener des revues formelles supplémentaires pour les produits *(exigences, architecture, code, tests)*.
Les besoins sont décrits avec un formalisme plus lourd que les **scénarios**. En effet, la DO-178B exige la spécification d'exigences finement tracées par rapport à des besoins du client et aux tests.
Le **cycle hebdomadaire** et le **cycle trimestriel** ne sont pas appliqués tels quels. Certains logiciels sont essentiellement composés d'algorithmes très complexes et ont peu d'interactions avec l'utilisateur. Il est alors difficile de mesurer en continue un avancement du logiciel en boite-noire. L'avancement est donc mesuré à l'aide de trois cycles. Un cycle quotidien permet de mesurer l'avancement des tâches. Un cycle mensuel permet de mesurer un avancement du logiciel en boite-blanche. Enfin, un cycle trimestriel permet de mesurer un avancement du logiciel en boite-noire.
Tester d'abord est très difficile pour le code impacté par la concurrence. L'écriture de ce code est pilotée par les tests, mais pas dans un cadre où il y a concurrence. Certaines adaptations du TDD sont nécessaires. En plus du classique cycle très rapide de test et intégration sur machine de développement, nous ajoutons un cycle moins rapide de compilation pour le processeur cible. A cela s'ajoute également un cycle moins rapide de test sur carte cible d'évaluation *(ou sur un éventuel simulateur de processeur cible)*. Enfin, lorsque le matériel cible est enfin disponible, s'ajoute un cycle moins rapide de test sur carte cible finale. Cette démarche permet de régulièrement mesurer la consommation en CPU et mémoire, qui sont des ressources critiques pour nos applications.
Il est difficile d'obtenir une **réelle implication du client** final, qu'il soit avionneur, certificateur ou pilote. Par contre, les projets ont toujours des représentants internes du client: experts métier et responsable qualité.
Le **déploiement incrémental** et le **déploiement quotidien** sont impossibles sur l'environnement de production final *(l'aéronef)*, d'autant plus que le logiciel doit être certifié avant d'être embarqué pour vol. Ces pratiques ne sont pas non plus réalisables sur un HW développé en parallèle du logiciel. Malheureusement, ces pratiques restent rares même sur un environnement simulé chez le client. Par contre, elles sont pratiquées sur un environnement simulé dans nos laboratoires.
Le code et les tests ne sont pas les seuls produits essentiels pour une certification D0-178B. Les audits de certification pour vol demandent la production de nombreux documents et preuves formelles non générées à partir du code et des tests. L'eXtreme-Programming ne dit pas qu'il ne faut pas produire de tels documents. La démarche propose de réfléchir au niveau strictement nécessaire et suffisant, d'être conscient que cela à un coût et de mesurer si cela vaut ce coût. Même si toute cette documentation n'est pas nécessaire pour produire un logiciel de qualité, elle apporte de la valeur puisqu'elle permet de passer des audits de certification. Pour éviter de remanier sans cesse la documentation, il s'agit de séparer le stable du variable, de documenter en priorité le stable et de choisir autant que possible des formats qui privilégient le travail collaboratif.
Le **contrat à contour négocié** est possible pour les versions intermédiaires, mais rarement pour la version finale. En effet, les applications avioniques contiennent peu de fonctionnalité optionnelles ou de confort.
Par contre, la traçabilité et la couverture des exigences sont si importantes pour les certificateurs que nous en avons créé une pratique supplémentaire. Au fil des activités, nous consignons les informations de traçabilité et couverture. En continue et automatiquement, ces informations sont consolidées, quantifiées, vérifiées dans la mesure du possible et enfin publiées.

Interprétation du Manifeste Agile

Comme les principes d'XP, ceux du Manifeste Agile sont également applicables. Néanmoins, il faut réinterpréter certains d'entre eux.

"Notre plus grande priorité est de satisfaire le client par la livraison tôt et en continue de logiciel de valeur."
Ce principe reste bien sûr valable, mais il faut comprendre que du logiciel avionique de valeur pour un client est un logiciel "bon pour vol". Obtenir cette valeur demande un travail bien plus conséquent que pour de nombreux autres projets de développement.

"La façon la plus efficace de transmettre l'information à une équipe et entre les membres est par des conversations face à face."
Ce principe s'applique également, mais une partie de l'information doit de plus être consignée pour l'audit de certification car les informations informelles ne constituent pas des preuves.

"Le logiciel opérationnel est la principale mesure d'avancement."
Ce principe s'applique en partie, car il faut bien comprendre qu'un logiciel avionique n'est réellement opérationnel que s'il est certifié, et la certification est une démarche peu incrémentale.

4. POURQUOI CE SECTEUR EST A PRIORI PEU RECEPTIF A L'AGILITE?

L'industrie est très marquée par la démarche Lean en production et logistique. Il est alors très étonnant que l'avionique mette autant de temps à adopter le Lean en développement. Pourquoi ce secteur semble peu réceptif?
D'abord, la peur des audits de certification induit une peur de changer des pratiques qui ont aboutit à un produit certifié. Les auditeurs ont l'habitude de certifier des projets menés dans une démarche prédictive, en phases et pilotée par les plans. Alors, nous avons peur d'être les premiers à proposer de certifier un logiciel développé avec une autre démarche.
Aussi, il existe une idée préconçue selon laquelle la démarche Agile est inadaptée aux logiciels critiques, car perçue comme peu rigoureuse - comme si "léger" était synonyme de "à la légère". Nous constations que les processus lourds, détaillés et lents rassurent.
Il existe également une idée préconçue selon laquelle le TDD n'est pas approprié pour le développement de logiciels embarqués sur HW spécifique avec OS temps-réel - car il est difficile d'automatiser et de systématiser tôt dans le projet le test sur un processeur cible. Pourtant, il nous semble que les applications embarquées n'ont jamais été autant testées.
Enfin, le vocabulaire de l'Agilité *("eXtreme-Programming", "Scrum", "ScrumMaster", "Agile" ...)* ne semble pas accrocher les industriels.

5. MISE EN PRATIQUE PROGRESSIVE DE L'AGILITE

Comment avons nous conduit le changement ?

Nous avons eu la chance d'avoir un Champion. Il s'agit de quelqu'un, écouté dans l'entreprise, qui voit l'opportunité, est convaincu de son intérêt et qui explique sans relâche.

Pour nous, la transition vers l'Agilité est une démarche progressive et continue reposant sur l'initiative de développeurs logiciel passionnés. Ceci fait d'ailleurs la fragilité de la démarche!

Introduite par des développeurs logiciel et soutenue par la hiérarchie proche, la démarche a d'abord naturellement concerné le périmètre purement logiciel. Suite aux premiers résultats positifs sur la qualité des produits et la motivation des équipes, la démarche s'est déployée sur un nombre croissant de projets. Les développeurs commencent par adopter les pratiques les plus techniques, puis passent au travail d'équipe pour finir par la gestion de projet. Ensuite, la démarche se déploie sur un périmètre plus large que le logiciel.

Chronologiquement, la progression a été la suivante:

1. Les équipes recherchent l'excellence technique et sont intransigeantes sur le niveau de qualité. Cette prise de conscience est déterminante pour la suite.
2. La démarche orientée-objet s'installe. Les principes avancés de conception orientée-objet décris par Robert Martin sont appliqués. On organise la réutilisation.

3. Les équipes sont pluridisciplinaires.
4. La programmation par contrat est systématisée avec une nette amélioration de la qualité.
5. Les tests natifs sont progressivement mis en place.
6. Les tests natifs sont automatisés.
7. L'eXtreme-Programming fait des émules dans les équipes. On applique la conception incrémentale, la construction en 10 minutes, l'intégration continue et l'automatisation. On officialise le partage du code, le travail en binômes, l'espace de travail commun, la courte réunion quotidienne d'avancement et le management visuel.
8. Les équipes passent à du développement itératif et incrémental sur des cycles très courts.
9. Les équipes travaillent sur la communication, le travail d'équipe, l'amélioration continue et la gestion de projet Agile en mettant en place des réunions de début d'itération, de fin d'itération, les rétrospectives d'itération et le suivi d'indicateurs d'avancement mesurés en continue.

Les projets menant une démarche Agile sont généralement accompagnés par un coach dédié. Il s'agit d'un développeur Agile expérimenté et/ou simplement passionné. On constate que les équipes demandent à être ainsi guidées. Par contre, on remarque que la démarche reste encore fragile puisque certaines pratiques perdent en rigueur ou ne sont plus appliquées si le coach quitte le projet.

6. COMMENT PARLER D'AGILITE A DES INDUSTRIELS?

Le vocabulaire
En tant que praticiens du développement logiciel Agile dans un domaine industriel, nous avons très souvent dû vendre - ou plutôt justifier - la pertinence de notre démarche auprès de notre management et de clients.
Au cours de nombreuses présentations et démonstrations, nous nous sommes rendu compte que nos auditeurs étaient peu réceptifs aux termes "eXtreme-Programming", "Scrum" ou même "Agile". Nous avons même constaté que ces noms desservent parfois les démarches qu'ils nomment. Travaillant dans un domaine industriel, nous avons remarqué que notre management et nos clients sont bien plus réceptifs à l'évocation de la démarche Lean. Ainsi, nous avons appris à adapter notre communication au vocabulaire du Lean pour accrocher nos décideurs.

Les résultats
Comme souvent, il faut convaincre en communiquant sur les résultats obtenus. Par exemple, lors d'un récent projet, 16KLS de C++ avec 30% de réutilisation, développés en cinq mois à quatre ont été intégrés sur cible sur site client en quatre jours. Le taux de défauts résiduels est de 0.18/KLS. Les coûts et délais du projet ont été respectés. Les défauts résiduels sont rapidement corrigés sans monopoliser le HW et sans régression. La démarche de ce projet, entre autre, est prise comme modèle de base pour un projet stratégique et d'envergure qui débute.

7. OBSTACLES QUI PERSISTENT

Inertie des grandes organisations
L'organisation de l'entreprise, en termes de processus, de standards, d'outils, de locaux et d'organisation d'équipes pluridisciplinaires ne suit pas le rythme d'évolution demandé par les développeurs.
Par exemple, il existe une masse importante de standards orientés cycle en V qu'il est difficile de faire évoluer d'autant plus qu'ils sont figés par une démarche CMMi.
De même, les outils sont standardisés et gérés par une DSI centrale qui ne suit pas l'évolution des besoins spécifiques des projets.
Par ailleurs, certaines descriptions de postes standards ne sont plus adaptées : les chefs de projet perdent une grande part de leurs anciennes fonctions au profit d'une équipe auto-organisée et d'un coach.

Une organisation à 2 vitesses
La propagation de la démarche Agile se fait projet par projet. Elle n'est pas imposée et donc est menée d'abord par les équipes les plus motivées par la démarche. La constitution d'un esprit d'équipe fort fait que les développeurs hors d'un projet Agile se sentent exclus. Le département logiciel tend à se découper en deux groupes. D'un côté, les adeptes de la démarche qui veulent l'appliquer toujours plus loin et ceux qui

s'en sentent exclus, qui ne sont pas attirés ou qui résistent au changement.

Interprétations de la DO178B
L'esprit de la DO178B amène à une dérive chez les certificateurs qui consiste à penser que la qualité et la sûreté de fonctionnement sont essentiellement le résultat de l'application d'un processus très détaillé où chaque activité est réalisable exclusivement à partir des informations produites par la précédente. Dans cette séparation de « penser » et de « faire », le développeur menant une activité est interchangeable au détriment de la flexibilité et de la créativité. D'ailleurs, lors des audits, les certificateurs cherchent à identifier et comprendre par eux-mêmes tout le code induit par un besoin opérationnel. Ceci s'oppose à l'idée selon laquelle un logiciel de qualité est le résultat du travail d'une équipe de qualité appliquant des pratiques qui incitent la flexibilité et la créativité.

Facteur d'échelle
Les logiciels sont embarqués dans des systèmes dont le développement ne suit pas une démarche Agile. Il est alors difficile de travailler en flux-tendu tiré dans un contexte plus vaste qui fonctionne en phases.

Clients
Certains clients n'ont pas confiance sur la tenue des délais. Ils s'enferment alors dans l'idée qu'ils peuvent suivre les développements avec des revues calquées sur des fins de phase et contrôler la pertinence des travaux grâce à la documentation. Un tel client n'est pas conscient de l'intérêt qu'il peut tirer d'un fournisseur ayant une démarche Agile. Malheureusement, les développeurs ont très rarement accès aux clients finaux, ce qui ne permet pas de leur exposer la démarche.
Dans ce cas de figure, nous avons parfois une attitude schizophrène: nous voulons être Agile mais nous avons peur de la réaction des clients. Alors, nous cachons la démarche derrière une façade de développement prédictif en phases.

L'automatisation et le TDD sont difficiles au niveau système
Plus le niveau du test est élevé, plus l'automatisation devient difficile. Aussi, le TDD ne permet pas de tester la concurrence. Néanmoins, il aide à séparer les problèmes et donc à isoler les problèmes de concurrence.

Organisation des équipes
La pratique de l'équipe complète est respectée. Malheureusement, le travail est souvent perturbé par une comptabilité par métier et non par projet. Ceci aboutit trop souvent à des intérêts conflictuels au sein de l'équipe. Il s'agit là d'une conséquence directe des modèles industriels centrés sur les activités et non sur les flux.

8. CONCLUSION

Contrairement aux idées reçues, la démarche Agile et l'eXtreme-Programming en particulier nous aident à développer efficacement des logiciels temps-réel embarqués. De même, ces démarches sont appropriées pour assurer la sûreté de fonctionnement d'un logiciel. Néanmoins, elles doivent être adaptées pour fournir le niveau de formalisme requis par les audits de certification pour vol.

Il y a quelques années les industriels sont passés avec beaucoup de précautions au développement orienté-objet de logiciels avioniques. Les certificateurs sont actuellement en train d'auditer cette première vague d'applications. Nous pouvons alors envisager que dans un futur proche, de plus en plus d'industriels vont chercher à certifier des produits développés avec des processus plus légers mais garantissant au moins la même sûreté de fonctionnement. Alors, les audits devront peut-être également aller dans le sens de la fluidité, peut-être en adoptant une certification incrémentale ou en continue en déléguant un auditeur sur site.

En tout cas, en tant que développeurs, nous sommes rassurés de constater que la démarche en cours a permis de diminuer la complexité des applications, d'améliorer la testabilité, d'automatiser la non-régression, de baisser très sensiblement le nombre de défauts résiduels, de permettre la réutilisation et d'accélérer les développements tout en assurant la sûreté de fonctionnement.

Alors, monteriez-vous à bord d'un avion dont des logiciels de vol sont écrits par des praticiens de l'eXtreme Programming? Bientôt, vous le ferez sans en avoir conscience.

9. BIBLIOGRAPHIE

- Considérations sur le logiciel en vue de la certification des systèmes et équipements de bord – RTCA DO-178B / ED-12B - 1992
- eXtreme Programming Explained - Kent Beck
- Manifesto for Agile Software Development - http://agilemanifesto.org/
- Towards An Agile Avionics Process - Andrew Wils, Stefan Van Baelen - 2007
- XP in a Safety-Critical Environment - Mary Poppendieck - 2007
- Embedded Extreme Programming: An Experience Report - Nancy Van Shooenderwoert - 2004
- Effective Test Driven Development for Embedded Software - Michael J. Karlesky, William I. Bereza, Carl B. Erickson - 2006
- Extreme Programming And Embedded Software Development - James Grenning - 2002
- Progress Before Hardware - James Grenning - 2004
- Embedded Test Driven Development Cycle - James Grenning - 2004
- Launching XP Extreme Programming at a Process-Intensive Company - James Grenning - 2001