

APPLICATIONS EMBARQUÉES

Test logiciel : des méthodes sortent des sentiers battus

▼ Toutes les applications embarquées dans les domaines critiques sont validées par des tests. Ce n'est pas encore assez systématique dans d'autres secteurs. Les outils de test continuent de se perfectionner, mais on note surtout des changements dans la manière de les utiliser. Autrefois cantonnés au seul cycle en V, les développeurs ont désormais à leur disposition de nouvelles méthodes de programmation. La conception orientée modèle, la vérification amont ou encore la méthode agile comptent parmi celles-ci.

Les "bugs" informatiques sont devenus monnaie courante dans l'informatique grand public, à tel point qu'on se contente de redémarrer son ordinateur et de relancer l'application qui vient de planter. Toutefois, dans l'univers des logiciels embarqués, une erreur d'exécution du programme peut avoir des conséquences dramatiques. On se souvient

de l'explosion de la fusée Ariane 5 quarante secondes après son décollage à cause d'un bug informatique. L'augmentation des pertes financières liées aux erreurs de programmation s'élèveraient à environ 150 milliards d'euros par an rien que pour l'Europe. Et ce montant n'est pas près de diminuer.

L'augmentation des pertes financières liées aux erreurs logicielles s'explique par plusieurs facteurs. Premièrement, par la croissance quasi exponentielle de la taille des logiciels. On estime que la complexité des programmes double tous les 18 mois. Elle suit de très près l'évolution de la puissance des microprocesseurs, qui double elle aussi tous les 18 mois (selon la loi de Moore). Deuxièmement, par l'augmentation du nombre d'équipements électroniques embarqués. Une voiture moderne compte aujourd'hui en moyenne entre 50 et 100 calculateurs, soit environ le nombre de calculateurs embarqués à bord de la fusée qui emporta les premiers hommes vers la lune en 1969. Enfin, par une utilisation encore trop peu répandue des outils de test logiciel. Des tests simples peuvent éliminer la plupart des problèmes de programmation, mais beaucoup d'industriels hésitent encore à franchir le pas. Ces logiciels de test sont-ils trop coûteux ? Pas vraiment, car détecter les erreurs au plus tôt sera toujours plus rentable que d'effectuer des modifications a posteriori. Les experts estiment qu'une erreur logicielle coûte environ trente fois plus cher

à réparer quand elle est identifiée après la mise sur le marché du produit. Et l'argument du coût des outils de test ne tient plus depuis que des logiciels gratuits sont désormais disponibles en libre téléchargement sur Internet.

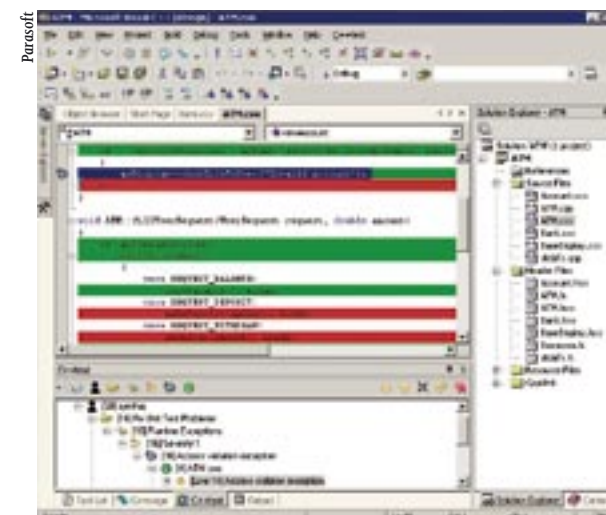
Pour quels secteurs ?

Bien entendu, les industriels qui développent des applications critiques sont déjà sensibilisés à toutes ces problématiques. Ils sont parmi les premiers utilisateurs d'outils de test logiciel. C'est le cas dans tous les secteurs pour lesquels des vies humaines peuvent être mises en danger, tels que l'automobile, le transport ferroviaire, le médical, l'aéronautique ou l'aérospatiale. Dans ces domaines, le développement logiciel est le plus souvent soumis à des normes, voire à des demandes de certification.

En revanche, pour les secteurs moins critiques, le test logiciel est une pratique encore trop peu répandue. « On voit encore beaucoup trop de logiciels "banane", lance Klaus Lambertz, cofondateur de la société VerifySoft Technology. Cette expression désigne des logiciels qui sont vendus malgré des tests insuffisants : l'éditeur fournit une application encore "verte", destinée à mûrir chez le client. On aura compris que ça n'est pas la bonne solution. Le test doit faire partie intégrante du développement d'un logiciel. Et il doit impérativement être pris en compte au plus tôt si l'on veut qu'il apporte une réduction des coûts et une augmentation de la qualité des logiciels. »

Pour quels langages ?

Le marché du test logiciel connaît actuellement une forte croissance. Des start-up se créent afin de commercialiser des technologies issues de laboratoires de recherche. Coverity, dont les produits sont basés sur une



Les générateurs automatiques de cas de test analysent le comportement d'un logiciel et tentent ensuite de le prendre en défaut afin d'évaluer sa robustesse.



D'après une étude de VDC, près de 60 % des retards dans les projets embarqués seraient liés à des modifications dans les spécifications. D'où l'apparition de méthodes plus réactives telles que la programmation agile.

méthode d'analyse statique de code mise au point à l'université de Stanford (Californie), est de celles-là. « Notre activité croît depuis la création de la société en 2002, déclare Ben Chelf, fondateur de Coverity. Sur les six premiers mois de ce début d'année 2009, nous sommes passés de 90 collaborateurs à plus de 130. » Cet engouement des industriels pour les outils de test logiciel ne touche pas que les Etats-Unis : cette entreprise a vu le nombre de ses clients français doubler en l'espace d'un an.

Le nombre et la variété des outils disponibles sur le marché vont donc en augmentant. Encore faut-il qu'il s'agisse d'un langage courant. Car pour tester un logiciel, il faut choisir un outil adapté au langage employé. Pour les langages les plus courants (à savoir

C, C++, Java et C#, qui se partagent 70 à 80 % du marché du logiciel embarqué), une large gamme d'outils est déjà disponible. Pour les langages spécifiques tels que le Pascal, l'Ada ou le Fortran, ils se font plus rares. Cela signifie que l'éventail des tests possibles sera plus réduit. Heureusement il s'agit de langages "stricts" qui laissent peu de possibilités d'erreurs, et les développeurs qui les emploient ont une expertise très pointue dans leur secteur.

Un outil pour chaque type de test

Le cycle en V s'est largement imposé depuis plus de vingt ans pour le développement des logiciels. Il associe la rédaction des spécifications et les phases de conception aux diffé-

rents tests à réaliser. Ces tests, que l'on retrouve sur la partie droite du "V", sont les tests unitaires, les tests d'intégration, les tests de validation et de recette. A chacun de ces types de tests correspondent des outils particuliers.

Les outils de test unitaire sont utilisés pour vérifier le bon fonctionnement d'une partie d'un logiciel. Une "unité" est considérée comme étant le plus petit composant du logiciel final (on parle aussi de modules dans le cas du langage C). Pour réaliser ce type de tests, on isole totalement l'unité. Elle doit être testée indépendamment du reste du programme. Certes, cela implique de réaliser un environnement de test autonome (les "vecteurs de tests" utilisés comme entrées doivent être conçus sur mesures). Le test unitaire présente l'avantage de pouvoir être réalisé dès le début du développement. Chaque partie du logiciel est testée individuellement dès qu'elle est achevée, ce qui facilite l'intégration (on assemble des composants "propres"). « Attention toutefois à rejouer une dernière fois tous les tests unitaires avant l'intégration, avertit Klaus Lambertz (VerifySoft Technology). Car si certaines unités ont dû être modifiées après les tests, il est possible de voir apparaître de nouveaux dysfonctionnements dans l'application. »

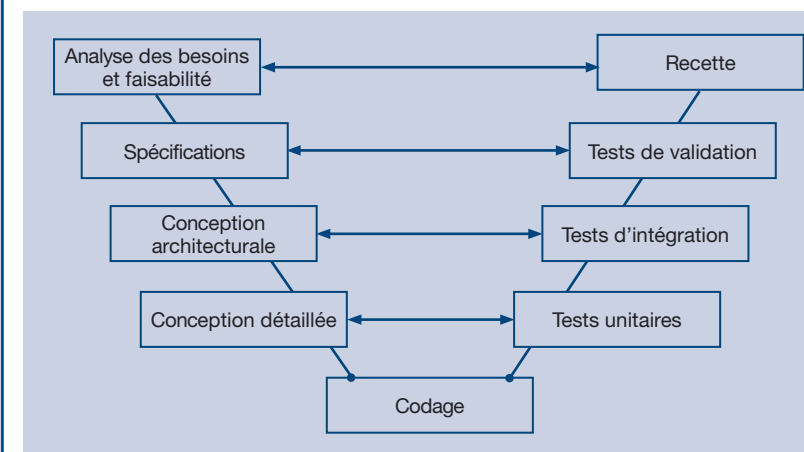
Les outils de test unitaire doivent s'accompagner d'une **vérification de la couverture de test**. On l'appelle aussi "couverture de code". Il s'agit cette fois de s'assurer que les tests unitaires définis précédemment vérifient effectivement tout le code. « La couverture structurelle consiste à vérifier que toutes les instructions sont atteintes et exécutées lors du test, explique Cyrille Comar, cofondateur et directeur général de la société AdaCore. Mais on ne passe pas forcément par tous les chemins →

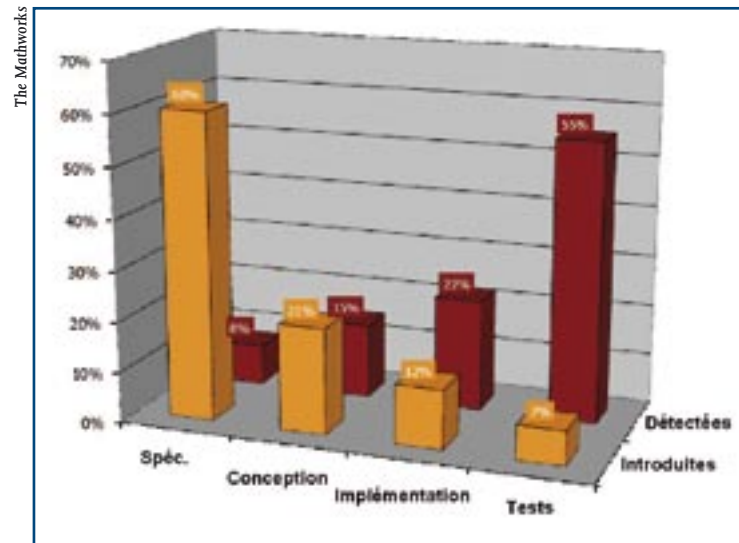
L'essentiel

- Les outils de test logiciel contribuent à réduire les coûts de développement.
- Les secteurs développant des applications critiques maîtrisent toutes ces techniques, mais la plupart des autres marchés ne sont pas encore sensibilisés.
- Les tests logiciels consistent en des analyses statiques (sur du code inactif) ou dynamiques (code en exécution).
- Les outils de tests se perfectionnent, et de nouvelles méthodes visent à se défaire du traditionnel cycle de développement en V.

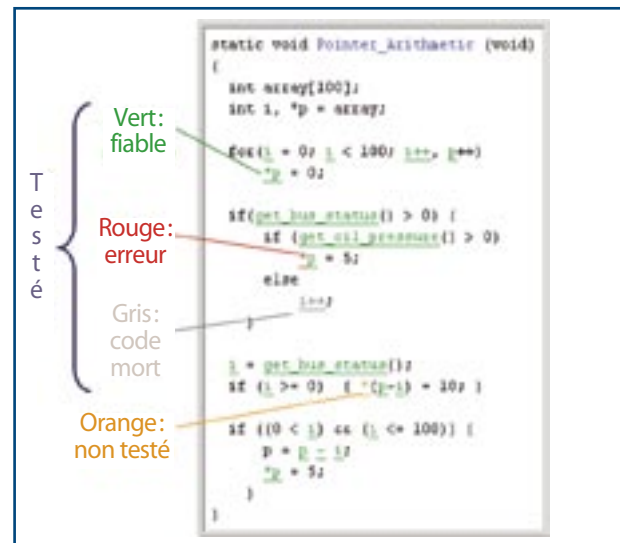
Le cycle en V de développement

Le cycle en V est devenu un standard de l'industrie logicielle depuis les années 1980. Avec cette méthode, les étapes de tests (branche de droite) s'accompagnent d'un retour d'informations vers les phases de conception. Mais en pratique, on s'aperçoit souvent que des spécifications initiales étaient incomplètes ou erronées, et les modifications sont difficiles à prendre en compte.





Tout retard dans la détection d'un problème de conception augmente le prix de sa résolution. Or cette étude de EDA Tech Forum montre que la majorité des erreurs sont introduites dès la définition des spécifications.



Le test statique, c'est trouver des erreurs sans exécuter le code. Les logiciels d'analyse statique utilisent un code couleur pour identifier facilement la criticité des erreurs.

→ possibles, ce qui serait trop long voire impossible. Imaginons que les instructions du programme constituent un nuage de points. Il est possible de mesurer le nombre de points qui sont exécutés lors du test, mais il est impossible de vérifier que tous les chemins existants entre ces points ont été vérifiés. » Pour cela, les outils du marché présentent sous forme graphique les parties du code qui ont été testées et celles qui ont été oubliées. La couverture de test n'est donc pas un test à proprement parler, mais il s'agit tout de même d'un outil indispensable pour mesurer la qualité d'un test. Cette mesure de la qualité est indispensable dans les secteurs critiques et réglementés, pour lesquels

l'intégralité des programmes et des tests doivent faire l'objet d'une documentation rigoureuse.

Les tests d'intégration s'effectuent après la phase des tests unitaires. Placés en face de la conception d'architecture dans le cycle en V, ils consistent à valider les interactions entre les différents composants du programme. Les outils du marché aident les développeurs à regrouper leurs différents composants au sein d'une application fonctionnelle, et à prendre en compte toutes les modifications apportées à la suite des tests unitaires. On valide ensuite tous les échanges entre composants.

Tests de boîte blanche et tests de boîte noire

Tests unitaires, couverture de code et tests d'intégration consistent en des vérifications du code proprement dit. Ces tests analysent le fonctionnement interne du logiciel. On parle de tests de boîte blanche. Les outils correspondants pourront détecter (par exemple) des divisions par zéro, des données corrompues, des liens surnuméraires, des liens qui n'aboutissent nulle part ou encore des valeurs de sortie qui dépassent les limites d'entrée dans les autres composants.

Au fur et à mesure que l'on remonte la branche de droite du cycle en "V", on ne cherche plus à trouver des erreurs dans le code mais plutôt à vérifier que le logiciel répond aux exigences. L'objectif est de détecter des défauts de comportement. Pour ce type de tests, on teste la réponse aux spécifications sans s'occuper de la manière dont le programme est écrit, c'est pourquoi on parle de tests de boîte noire. On distingue deux types de tests de boîte noire : les tests de validation et les tests d'acceptation.

Le test de validation, également appelé test de système, consiste à vérifier que le logiciel remplit bien toutes les exigences définies lors de la rédaction de ses spécifications. Ben Chelf (Coverity) explique que « ces outils peuvent être utilisés par exemple pour régler les problèmes de "multi-threading". Dans les applications programmées avec des méthodes orientées objet, il n'est pas rare en effet d'exécuter plusieurs tâches (ou "threads") simultanément. Les plantages les plus courants peuvent survenir lorsque deux tâches doivent accéder au même espace mémoire, ou encore lorsqu'elles attendent que l'autre soit terminée avant de

Test statique ou dynamique ?

Les outils de test logiciel peuvent être séparés en deux principales catégories, selon qu'ils effectuent du test statique ou dynamique. Dans le premier cas, on ne fait que lire le code, et dans le second, on l'exécute.

Voici quelques clés pour déterminer quelle méthode de vérification utiliser, pour quelles erreurs et à quel moment :

	Les outils de test statique	Les outils de test dynamique
Points forts	<ul style="list-style-type: none"> - sont capables de déterminer l'emplacement exact d'un défaut ; - réduisent les coûts finaux en détectant les erreurs au plus tôt ; - se révèlent relativement rapides si les tests sont automatisés. 	<ul style="list-style-type: none"> - sont capables d'identifier des vulnérabilités dans un environnement d'exécution ; - vérifient tout type de programme, même sans avoir accès à tout le code source ; - permettent de valider les résultats du test statique.
Points faibles	<ul style="list-style-type: none"> - peuvent être longs s'ils sont pilotés manuellement ; - demandent une expertise spécifique pour être correctement employés ; - peuvent donner de fausses impressions en mode automatique (des erreurs qui n'en sont pas, ou une absence d'erreur alors qu'en exécution le programme plante). 	<ul style="list-style-type: none"> - réclament une expertise spécifique pour être correctement utilisés ; - peuvent donner de fausses impressions en mode automatique (l'outil génère des cas de test seulement en fonction des règles qu'on lui indique) ; - n'autorisent pas de remonter facilement jusqu'à la cause d'une erreur.

Les programmes automatiques ne font plus bande à part

La programmation des automatismes est depuis toujours en marge de celle du logiciel embarqué. Il existe en effet des différences entre les deux secteurs. Tout d'abord, les programmes automatiques sont destinés à être employés en plus petites séries que dans l'informatique embarquée. Ensuite, les cibles matérielles sont des automatismes "standards" du commerce tandis que dans le monde de l'embarqué, on a plus souvent à faire à des cibles spécifiques.

Toutefois, cette situation est en train d'évoluer. Les secteurs privilégiant autrefois l'emploi d'automatisme peuvent se tourner aujourd'hui vers des calculateurs embarqués pour certaines applications, et inversement. Cela se vérifie par l'utilisation plus fréquente d'automatisme dans des secteurs comme le ferroviaire ou le nucléaire. Du coup, certains procédés de développement issus du monde des calculateurs, tels que la simulation et la génération de code, se sont invités dans le monde des automatismes. Geensys a notamment fait de la simulation l'un des principaux atouts de son logiciel ControlBuild. Dans le même esprit, The Mathworks propose désormais l'outil PLC Link (du danois HybridTech) qui génère du code automate à partir de modèles Simulink. Malgré ces efforts de simplification, dans la quasi-totalité des cas les programmes doivent être repris dans l'atelier logiciel du constructeur de l'automate. De son côté, Itris Automation Square a développé le premier outil de vérification statique de code automate. Jusqu'alors, il n'existait pas de moyen de vérifier que le code généré était exempt de toute erreur.

Son logiciel, PLC Checker, sera utilisé pour améliorer la lisibilité et la maintenabilité du code, mais aussi pour avertir le développeur que certaines règles de codage n'ont pas été respectées. « Comme pour

les autres outils de vérification statique, il ne s'agit pas de contrôler que le programme répond bien aux spécifications, mais plutôt de voir "à quoi il ressemble", explique Eric Pierrel, directeur marketing chez Itris Automation Square. PLC Checker sera utilisé pour relever des erreurs de syntaxe ou de commentaires ainsi que des problèmes liés aux variables (réduire le nombre de lectures/écritures par cycle, faire attention aux variables partagées par plusieurs tâches, etc.). »

Le logiciel peut être utilisé en audit à la fin du développement d'un programme, mais il donne des résultats plus satisfaisants lorsqu'il est employé tout au long du processus. En entrée, il reçoit les fichiers d'exportation des ateliers logiciels (.XEF pour Unity, .AWL pour Siemens S7, .FEF pour Schneider PL7 Pro, etc.). La vérification du code utilise un "langage pivot" : le langage automate est traduit dans un langage d'abstraction de haut niveau appelé GLIPS (un langage proche de l'Ada), ce qui permet de conserver le même moteur de vérification statique quelle que soit la marque de l'automate. Et en sortie, le logiciel fournit des indications chiffrées : taux de commentaires, niveaux d'imbrication des différents composants... Il met également en évidence les règles de construction à risque (des fonctions "GO TO" qui reviennent en arrière, ou encore des grafset avec plusieurs jetons). A noter enfin que la société grenobloise propose un système de licence particulier : le Software-As-a-Service (SAS). « PLC Checker fonctionne avec un principe de serveurs sur lesquels les clients chargent leurs applications, explique Eric Pierrel (Itris Automation Square). Les clients paient uniquement ce qu'ils utilisent, et n'ont aucune licence annuelle à payer. Il n'y a pas besoin d'amortir à tout prix la licence du logiciel de vérification. Un concept bienvenu en ces temps de crise. »

s'exécuter. Avec les logiciels de test de validation les plus récents, il est possible d'identifier et de corriger ces comportements à risque ».

Le test d'acceptation, enfin, est réalisé en situation. Il s'agit de prouver au client que le logiciel répond correctement à tous les besoins exprimés lors de la rédaction du cahier des charges.

Afin de s'assurer qu'aucune des manipulations possibles n'aboutisse à un plantage de l'application, le testeur explore manuellement toutes les possibilités offertes par le programme, dans toutes les configurations possibles. Mais ce travail peut vite devenir titanesque dans le cas de systèmes complexes ou volumineux. Heureusement, des générateurs automatiques de cas de tests ont fait récemment leur apparition sur le marché. Ces outils reprennent la liste des exigences (décrites de manière graphique) et génèrent automatiquement les cas de tests à réaliser dans un langage plus abstrait. Utiles au développeur et à son client, ils apportent une assurance supplémentaire que toutes les exigences ont bien été prises en compte et que toutes les fonctions sont bien assurées.

Il est toutefois à noter que ces générateurs de cas de tests ne sont pas toujours vus d'un très bon œil par les organismes de certifica-

tion. « Pour les applications soumises à la norme DO-178 B (relative à la certification logicielle dans l'aéronautique), notamment, il est conseillé aux ingénieurs d'écrire leurs propres tests (des tests "intelligents", qui vérifient bien les spécifications) plutôt que de laisser l'outil de test décider de la manière de tester un programme », observe Cyrille Comar (AdaCore).

Tous les outils utilisés pour le test logiciel s'inscrivent dans le cycle en V, mais ce dernier n'est pas la seule méthode de développement. Le secteur du logiciel embarqué et critique étant un marché de niche, certains éditeurs préfèrent mettre en avant d'autres méthodes. Les outils restent globalement les mêmes, mais ils sont utilisés de manière différente...

De nouvelles approches pour améliorer les tests

Le cycle en V conduit les ingénieurs à scinder leur travail en trois étapes : d'abord l'analyse pure, puis le développement, et enfin les tests. Mais cette méthode a ses limites, du fait de cette séparation qui subsiste entre les étapes. Afin de vraiment maîtriser le comportement des logiciels, leur complexité et (plus tard) leur maintenance, il est nécessaire de disposer d'un fil conducteur, d'un moyen de représen-

ter l'application qui soit utilisable tout au long du cycle de développement.

De même que pour les générateurs de cas de test, une des solutions consiste à introduire un niveau d'abstraction supplémentaire, sous la forme d'un nouveau langage. Il s'agit de la modélisation. On parle alors de Model Driven Engineering (conception orientée modèle) ou encore de Model Based Design (conception basée sur des modèles).

« La finalité est principalement économique, commente Pierre Dissaux, directeur de la société Ellidiss Technologies, car le test d'applications embarquées coûte très cher. On estime en effet qu'il représente environ 60 % des investissements en développement, et ce chiffre augmente encore pour les applications très critiques. Or le seul moyen de réduire la facture des tests est de régler les problèmes au plus tôt. C'est la raison pour laquelle la tendance actuelle est à une remontée de la programmation vers la modélisation. » L'idée est donc de formaliser au plus tôt l'architecture de l'application afin de réaliser le plus de tests possible en amont. Il faut pour cela utiliser un langage de modélisation.

Le langage de modélisation le plus couramment utilisé est l'UML (Unified Modeling Language, ou langage de modélisation unifié). Il s'agit d'un langage généraliste, →

ASB Avionics



Dans l'aéronautique, tous les logiciels doivent être certifiés car ils peuvent avoir une influence sur la sécurité du vol. La norme DO-178 B prévoit plusieurs degrés de criticité, depuis le niveau A (qui mène au crash) jusqu'au niveau E (aucune incidence sur le vol).

→ adapté aux applications web et bureautiques mais pas vraiment aux applications embarquées. L'un de ses principaux défauts est l'absence de prise en compte des aspects temps réel. Difficile en effet de modéliser en UML le déterminisme d'une application (le temps de réaction du programme à un événement donné). Différents travaux de recherche ont donc été entrepris afin de concevoir des langages de modélisation

adaptés aux applications déterministes. C'est ainsi que sont nés des standards tels que le langage AADL (Architecture Analysis & Design Language, ou langage de conception et d'analyse d'architecture). « L'AADL est employé pour modéliser le comportement d'une application en prenant en compte ses aspects temps réel, poursuit Pierre Dissaux (Ellidiss Technologies). Cette abstraction autorise l'anticipation des problèmes. Notre offre inclut un environnement graphique d'aide à la conception d'architectures en AADL, ainsi qu'un outil de test des propriétés temps réel. Et puisque l'AADL est un standard, il est possible d'utiliser toutes sortes d'autres outils de test sur le modèle. »

Quelques exemples d'outils

Voici une liste **non exhaustive** d'outils de vérification de code disponibles sur le marché

● Pour le test unitaire :

Testwell CTA++, de Testwell
Rational Test Real Time, d'IBM
Jtest, de Parasoft
CUnit (logiciel libre), pour le langage C
JUnit (logiciel libre), pour le langage Java

● Pour la couverture de test :

Testwell CTC++, de Testwell
Bullseye Coverage, de Bullseye Testing Technology
Couverture (projet Open Source initié par AdaCore)
GCOV (logiciel libre)

● Test d'intégration (ou test de système) :

Qtronic, de Conformiq
Test Designer, de Smartesting

● Outils "multifonctions" :

TestBed, de LDRA
Outils Polyspace, de The Mathworks
CodePeer, de AdaCore
VectorCAST, de Vector Software
C++test, de Parasoft

● Outils pour l'analyse statique :

Coverity Prevent, de Coverity
CodeSonar, de Grammatech
Insight, de Klocwork
DoubleCheck Static Analysis Tool, de Green Hills Software

Le concept de vérification amont

Autre éditeur, autre concept... The Mathworks, ardent défenseur du Model Based Design par le biais de son outil de modélisation Simulink, y ajoute désormais la notion de "vérification amont". Il s'agit d'offrir aux utilisateurs de Simulink une interopérabilité entre tous les outils de la gamme, dans le but de minimiser les erreurs logicielles. Quatre principaux besoins ont été identifiés auprès de ces utilisateurs. Il fallait d'abord rendre les spécifications directement exécutables sous Simulink. On utilise pour cela Simulink Design Verifier, qui relie des spécifications écrites sous Word, Excel ou Doors au modèle Simulink. Autre exigence : utiliser des modèles Simulink comme bancs d'essai, afin de valider des algorithmes et des composants plus directement. Ensuite, il fallait simuler le fonctionnement d'une application dans des environnements multiphysiques (mécanique, électronique, automatismes, équipements de vision industrielle, etc.). Enfin, les bancs d'essai créés à partir des modèles Simulink devaient être réutilisés tout au long du cycle de développement. « Notre gamme de produits couvrait déjà l'essentiel des besoins pour la programmation d'applications embarquées, et les outils de Polyspace disposent de fonctions très avancées en matière de vérification de code, assure Ascension Vizinho-Coutry, ingénieur d'application senior

chez The Mathworks. Mais grâce aux derniers développements, qui ont porté principalement sur l'interopérabilité entre les outils, nous sommes à même de proposer une plate-forme complète de modélisation, de simulation et de test. »

Des itérations plus rapides que dans le cycle en V

En utilisant des méthodes d'abstraction et en y associant la simulation, les tests démarrent au plus tôt dans le processus de développement. Il n'en reste pas moins que le cycle en V est relativement long et "directif" (il ne favorise pas les changements d'orientation pour le programme). Certains éditeurs optent donc pour d'autres méthodes, aux temps de cycles beaucoup plus courts et aux itérations plus nombreuses.

Ainsi, la société AdaCore met en avant une méthode de développement "agile", aux temps de cycle de l'ordre de la journée. « Cette méthode bouleverse les habitudes de programmation, commente Cyrille Comar (AdaCore) : avec le cycle en V, les vérifications et l'intégration sont effectuées en dernier, ce qui pose souvent des problèmes de compatibilité entre les composants. En programmation agile, on pense dès le début à l'intégration et les tests sont effectués à chaque itération. A chaque temps de cycle on effectue les tests unitaires sur les nouveaux composants et les tests d'intégration sur le système global. Une grande majorité des erreurs de programmation est ainsi détectée au plus tôt. »

Dans le cadre d'un développement agile, les cas de tests sont définis immédiatement après les spécifications, avant même de commencer à coder. Très utile dans le cas d'applications certifiées : les normes aéronautiques telles que la DO-178 B demandent justement que les tests soient écrits au plus tôt et soient directement tirés des spécifications. La méthode agile apporte d'autres avantages. Les développeurs se réunissent très régulièrement. Ils disposent ainsi d'une meilleure vision du planning ce qui contribue à diminuer les retards. Le codage proprement dit s'effectue en binôme afin de s'apercevoir des erreurs au plus tôt, et on pousse les équipes d'ingénieurs à sans cesse se remettre en question (en effet, le développement dit agile est à rapprocher des méthodes de type Lean, qui responsabilisent les employés). « Tous les jeunes développeurs issus des milieux web sont déjà formés aux méthodes de type agile, observe Cyrille Comar (AdaCore). Nous ne faisons que transposer des méthodes modernes dans des secteurs (l'embarqué et le critique) pour lesquels le cycle en V prédomine depuis plus de vingt ans. »

Frédéric Parisot