ISIMA 3ème année - MÉTHODES ET OUTILS D'INGENIERIE LOGICIELLE

Première session 2003/2004

Durée : 1 heure.
Documents autorisés : notes de cours manuscrites et polycopiés distribués cette année.

Le Monde Informatique n° 1007 du 19/12/2003

La programmation de mieux en mieux encadrée

Outils et méthodes de détection ou de prévention des pannes se multiplient afin de limiter la dérive des développements, en termes de coûts comme de délais.

En période de réductions d'effectifs et de budgets, diminuer le nombre d'anomalies dans les programmes est devenu essentiel. D'une part, parce que l'arrêt d'une application peut être très pénalisant pour l'activité de l'entreprise. D'autre part, parce que les coûts de maintenance corrective d'une application deviennent vite exorbitants si l'on ne détecte pas ces dérives au plus tôt. Pour autant, l'intérêt des utilisateurs pour le sujet ne date pas d'aujourd'hui. Le sujet reste l'une des préoccupations majeures des utilisateurs, qui réclament des applicatifs fiables et performants.

La modélisation s'impose

Partant de ce principe, analystes et éditeurs préconisent de contrôler la qualité du logiciel dès la phase d'analyse du projet, à partir des méthodes de modélisation. En offrant un cadre rigoureux, ces méthodes permettent de contrôler l'adéquation du logiciel produit aux besoins des utilisateurs, mais aussi de limiter le nombre de défauts et de simplifier la maintenance future. Au côté des méthodes classiques, dont l'antique et célébrissime Merise, les méthodes s'appuyant sur le standard UML et l'**architecture MDA gagnent en importance, en proposant une démarche plus indépendante des technologies et plus globale (question 3)**. Preuve de l'intérêt pour cet outil de modélisation, une nouvelle version majeure (UML 2.0) est attendue dans les prochains mois.
Mais au-delà de la modélisation, **c'est surtout au moment des développements que se joue la qualité d'un logiciel. Avec la question : une fois le modèle décrit, comment éviter ou limiter les erreurs de programmation ? La meilleure solution réside dans les tests (Question 1).** Tests qui doivent commencer dès les phases préliminaires du développement et être répétés tout au long du processus. Pour vérifier le code et détecter les erreurs encore plus en amont, afin toujours de réduire les coûts et le dérapage des délais, d'autres approches ont vu le jour. L'idée : déceler les possibilités d'erreurs avant que celles-ci ne soient commises. En passant ainsi d'une démarche corrective à une démarche préventive, voire prédictive les développeurs espèrent gagner en qualité et en temps.
La démarche pour la prévention des erreurs, ou AEP (Automated Error Prevention), que propose Parasoft vise à détecter un défaut ou une erreur, à identifier sa cause, à introduire des étapes dans le processus de développement pour éviter qu'elle ne se reproduise, à appliquer des procédures pour mesurer le succès (question 4). Le tout étant de comprendre pourquoi tel processus déclenche telle erreur, et d'éviter tout dysfonctionnement similaire par la suite.
Outre la gamme d'outils couvrant ces différents aspects proposée par Parasoft, on trouve sur le marché de nombreuses autres solutions (détecteur d'anomalies de codage, automate de tests, gestionnaires de défauts logiciels…) parmi lesquelles celles de Compuware, d'Ideo Technologies, de McCabe, de Mercury Interactive, de Polyspace, de Sodifrance ou encore de Telelogic. McCabe et Compuware ont, ainsi participé au projet lancé par la DSI Voyageurs de la SNCF pour surveiller la qualité de ses développements.
Certains outils voisins, tels que celui proposé par Cast Software, répondent à des besoins plus larges : analyse d'impact pour gagner en efficacité, ou encore contrôle du respect de certaines règles de conception. Ainsi, Cast dispose d'automates pouvant vérifier que les développeurs ne s'écartent pas de standards définis préalablement par l'architecte.

Industrialisation des processus de développement

Toujours dans le but de minimiser les risques d'erreurs, des démarches plus générales visent une amélioration des processus d'ensemble. A l'instar des modèles CMM, Six Sigma ou Spice, qui définissent les meilleures pratiques pour optimiser les performances d'un environnement donné (voir encadré), le modèle SW-TMM (Software Testing Maturity Model) est un outil d'amélioration du processus de test. Selon la même logique, plusieurs offres de méthodologie outillée permettent d'industrialiser ce processus. Ainsi, Mercury Interactive a packagé ses outils, prestations et bonnes pratiques en 5 "centres d'optimisation" consacrés chacun à un aspect de la problématique : gouvernance, qualité, performances, disponibilité métier, résolution des problèmes. L'objectif étant de corréler outils d'optimisation informatique et performances métier.
Cette offre d'assurance qualité repose sur des mesures relevées par divers outils de test, sur le plan des performances du code, de la robustesse de l'application… **On sort du strict périmètre de la qualité logicielle pour s'approcher de la notion de contrat de service ou de gestion des relations entre maître d'ouvrage et maître d'œuvre (Question 5)**. Un domaine largement normalisé au travers de modèles comme Itil (Information Technology Infrastructure Library) ou Cobit (Control Objectives for Information and related Technology).

SW-TMM, UN MODELE POUR LE PROCESSUS DE TEST

En matière de conception et de développement logiciels, la progression de CMM (Capability Maturity Model), plus connu jusqu'alors en informatique industrielle qu'en gestion, est spectaculaire. Défini par le DoD (ministère de la Défense américain) et élaboré depuis une quinzaine d'années par le SEI (Software Engineering Institute), ce modèle comporte cinq niveaux d'évaluation. Plusieurs modèles voisins existent aussi, dont Spice ou Six Sigma. Néanmoins, à la différence de l'assurance qualité proprement dite (ou du label ISO et ses déclinaisons), ces modèles dépassent le strict cadre de l'ingénierie logicielle, au sens des méthodes et outils spécifiques, pour s'intéresser au déploiement des processus, respect des délais et des coûts inclus. En revanche, le modèle SW-TMM (Software Testing Maturity Model), lancé par l'Illinois Institute of Technology, se veut plus proche de la programmation et des tests. De même que CMM, il comporte cinq degrés de maturité, du niveau 1, où le **test est un processus mal défini et confondu avec le débogage (question 2),** au niveau 5, où le test est institutionnalisé dans l'organisation. A ce dernier niveau, le test est bien défini et géré, et ses coûts et son efficacité sont contrôlés.

Thierry Parisot

Répondre en une dizaine de lignes aux questions suivantes (attention, certaines questions peuvent se recouper, veiller à bien « répartir » vos arguments) :

Question 1 - Etes-vous d'accord avec l'auteur de ce texte lorsqu'il affirme « c'est surtout au moment des développements que se joue la qualité d'un logiciel… La meilleure solution réside dans les tests » ?

Question 2 - En quoi un processus de test ne doit pas être confondu avec le débogage ?

Question 3 - Pourquoi la technologie MDA « permet de contrôler l'adéquation du logiciel aux besoins des utilisateurs, mais aussi de limiter le nombre de défauts et de simplifier la maintenance future » ?

Question 4 - Proposer un exemple pour illustrer une démarche de prévention des erreurs comme AEP.

Question 5 - Expliquez la phrase « On sort du strict périmètre de la qualité logicielle pour s'approcher de la notion de contrat de service ou de gestion des relations entre maître d'ouvrage et maître d'œuvre »

Question 6 - A votre avis, suffit-il d'acquérir un outil de test pour résoudre le problème de la validation de logiciels?