Publié dans la revue Genie logiciel, numéro 65, juin 2003

Conduite Statistique de la Validation d'un Système

Hélène Le Guen

Résumé : Bien qu'elle prenne une place importante dans le cycle de vie d'un système, la validation n'est pas, pour la plupart des projets, la phase la mieux structurée. Cette phase est majeure, car elle permet, d'une part, de valider les fonctionnalités et, d'autre part, de certifier la qualité du système. Elle est d'autant plus sensible qu'elle est fortement contrainte par les délais, car il s'agit de la dernière étape d'un projet de développement. Le présent article propose une nouvelle approche de la gestion de cette phase, commençant avec la planification des tests et allant jusqu'à la livraison du produit. Cette approche repose sur le test statistique d'usage et s'inscrit dans la modélisation, sous forme d'un modèle probabiliste (chaîne de Markov), des échanges entre le logiciel et son environnement Les cas de test sont automatiquement engendrés à partir de ce modèle. Nous avons bâti un processus spécifique utilisant ces cas de test, qui permet de connaître le délai restant avant la livraison, de suivre la campagne de test et de savoir quand interrompre les opérations de test. Ici l'objectif principal est de montrer l'application du test statistique d'usage dans un contexte industriel avec l'utilisation de mesures facilement calculables pour la gestion d'une campagne de test.

Mots clés : validation, test de logiciel, test statistique, fiabilité, chaîne de Markov

1. INTRODUCTION

Nous nous sommes intéressés au test au niveau d'un système considéré comme une boîte noire. En d'autres termes, nous souhaitons valider la conformité entre les spécifications d'un système et son implémentation, tout en nous assurant de sa fiabilité. Le présent article décrit un processus de test basé sur la technique du test statistique d'usage (SUT). L'objectif est de montrer qu'un processus de test défini de manière rigoureuse est une aide pour la planification, l'exécution et la gestion de la phase de test. Dans ce qui suit, un premier chapitre présente la technique du test statistique d'usage, puis les chapitres suivants décrivent les différentes étapes du test, à savoir l'élaboration d'un plan de test, l'exécution des tests et le suivi de la phase de test. Il y est montré l'intérêt de l'utilisation du test statistique. Les résultats obtenus sur des projets réels sont donnés.

2. LE TEST STATISTIQUE D'USAGE

La particularité principale de la méthode du test statistique est l'utilisation d'un modèle d'usage pour la génération de séquences de test. Introduite au début des années 70 [6] dans un contexte industriel, cette méthode n'est pas nouvelle ; elle relève d'un principe connu dans le test des circuits électroniques. Depuis, elle a donné lieu à de nombreux travaux et publications [2,3,4,7]. Les séquences de test ainsi engendrées sont pertinentes et représentatives de l'utilisation, dans des conditions normales, de l'application. Une séquence de test décrit les entrées et les sorties du logiciel échangées entre le logiciel sous test et l'exécution des tests. Un échange composé d'une entrée et d'éventuellement plusieurs sorties est désigné, par la suite, sous l'appellation « pas de test ».

Méthode

A l'instar de la méthodologie Cleanroom (méthode de développement appliquée par IBM [1], la méthode présentée a comme objectif de tester le logiciel selon un profil d'usage, sans s'attacher à savoir si tout le code d'un système logiciel a été testé.

Nous utilisons comme modèle d'usage une chaîne de Markov à temps discret irréductible et homogène, dont les états représentent ceux du fonctionnement et les transitions les actions de l'utilisateur. Une transition correspond à une entrée de l'environnement au logiciel sous test. Ainsi, la probabilité qu'un utilisateur applique une action donnée quand le logiciel est dans un état particulier d'usage est représentée par la probabilité de la transition correspondante

Le modèle utilisé contient toujours deux états particuliers, l'état *Invoke* et l'état *Terminate.* Le premier représente le logiciel avant que celui-ci ne soit activé et *le second* représente l'état du logiciel lorsque son activation est terminée.

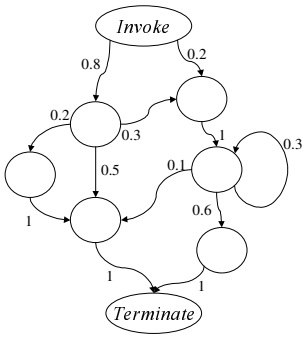

Figure 1 : Modèle d'usage

À partir de ce modèle il est possible d'engendrer des séquences de test, cas qui correspondent à des chemins dans la chaîne de Markov allant de l'état *Invoke* à l'état *Terminate*.

Avantages

Comme on le sait, tout ne peut pas être testé dans un logiciel ; en effet, une campagne de test exhaustive, qui consisterait à tester tous les enchaînements différents d'entrées, est en général inconcevable dans un contexte industriel voire impossible à réaliser en raison d'une infinité de combinaisons. C'est pourquoi il est intéressant d'utiliser des méthodes qui permettent de déterminer et de sélectionner un ensemble de séquences de test pertinentes : celles qui permettront de garantir le fonctionnement du logiciel de la façon la plus certaine pour l'usage auquel il est destiné. Le test statistique d'usage est particulièrement satisfaisant au regard de cette considération.

Un objectif des tests, engendrés d'après un profil d'utilisation, est de valider l'application selon son utilisation habituelle. Les probabilités représentent le comportement de l'utilisateur face à l'application.

De plus, la fiabilité ne dépend pas du nombre de défauts résiduels dans le code, mais dépend de la probabilité que ces défauts apparaissent durant l'utilisation du logiciel. Il s'agit de la notion de fiabilité recherchée dans le but de garantir un temps moyen de bon fonctionnement en utilisation opérationnelle. Si les probabilités associées à la chaîne de Markov ne correspondent pas à l'usage, la fiabilité calculée n'est plus valide, cependant les séquences de test engendrées peuvent être pertinentes au regard d'autres objectifs.

Outre les précédents, la représentation sous forme de chaîne de Markov offre d'autres avantages :

- La modélisation de l'usage d'une application améliore la qualité de la spécification par une représentation dynamique du système, tout en garantissant l'indépendance entre le développement et les tests.
- La modification des probabilités est aisée et permet d'engendrer des séquences de test non intuitives ainsi que des séquences de test spécifiques de fonctions données.
- La création de tests de non-régression est immédiate par la génération automatique d'une nouvelle séquence de test à partir du modèle modifié en fonction des modifications introduites dans le logiciel.
- La validation de nouvelles fonctionnalités dans le système est facilitée : en effet, il suffit d'adapter la chaîne de Markov pour que celle-ci contienne ces nouvelles fonctionnalités.
- La durée du test est optimisée dans la mesure où l'on est plus à même de déterminer le moment adéquat pour la livraison en fonction de la fiabilité du système.
- L'automatisation de la phase de test est possible pour la génération de séquences de test, le suivi et, parfois, pour l'exécution des séquences de test.

Actuellement, la technique du test statistique est peu utilisée par les sociétés qui conçoivent des logiciels, la culture d'un cahier de test est très présente. Ce faible niveau de pratique tient peut être, en partie, au fait que les promoteurs de la technique, relevant souvent du monde académique, n'ont pas intégré les contraintes et les réalités industrielles. On peut également remarquer qu'il existe peu de méthodes simples et d'outils opérationnels permettant sa mise en œuvre.

3. ÉLABORATION DU PLAN DE TEST

La première partie du processus des opérations du test consiste à définir le plan de test. Celui-ci décrit la totalité des actions qui seront effectuées lors de la phase de test :

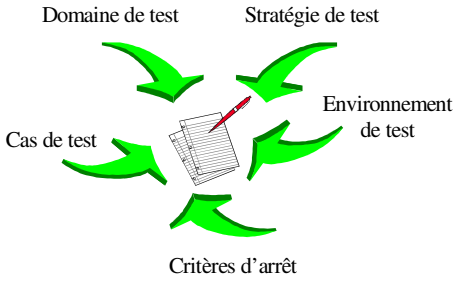

Figure 2 : Rôle du plan de test

Le domaine de test permet de définir quelles fonctionnalités seront testées. L'environnement de test définit, quant à lui, l'architecture de test, c'est-à-dire les interfaces de test ainsi que les conditions de test. La stratégie de test définit comment et dans quel ordre la phase de test va se dérouler. Évidemment, le plan de test doit inclure les séquences de test. Dans le cas présent où les séquences de test sont engendrées à partir d'un modèle d'usage, il inclut le modèle d'usage et non pas un échantillon de séquences de test qui peuvent être engendrées. On y trouvera également les objectifs de la phase de test c'est-à-dire les conditions d'arrêt.

Écriture du modèle d'usage

L'élaboration du plan de test commence avec la création du modèle d'usage. Ce dernier permet de décrire l'ensemble des échanges entre le système logiciel et son environnement. Il est écrit sous la forme d'une chaîne de Markov, les événements qu'un utilisateur peut appliquer sur le système étant affectés de la probabilité de leur occurrence. Le modèle utilisé ici est basé sur celui introduit dans [9] : chaque transition de la chaîne de Markov correspond à la probabilité qu'un événement d'entrée soit produit lorsque le système est dans un état donné.

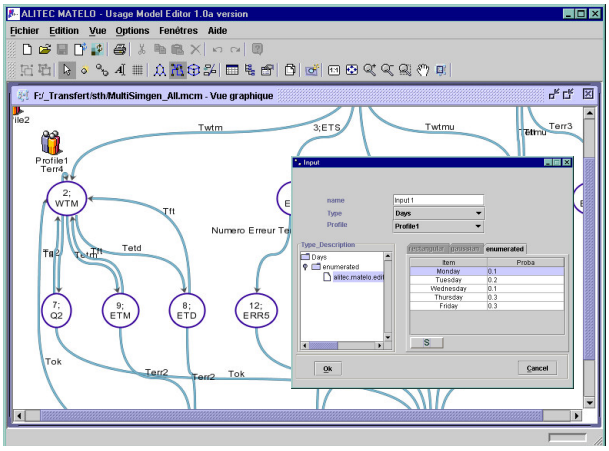

Figure 3 : Edition du modèle d'usage avec MaTeLo

Nous avons en plus ajouté des informations pour permettre une plus grande automatisation et rendre au maximum générique la description des échanges entre le système sous test et l'exécuteur des tests, qu'il s'agisse de test automatique ou manuel.

Ce modèle n'a pas pour objectif de fournir une description complète des résultats attendus. Cependant, il permet tout de même de lier à une transition un ensemble de données qui permet de déterminer si l'application a un comportement correct.

Une transition est liée à une entrée et éventuellement à une ou plusieurs sorties jusqu'à ce que le logiciel soit dans un état stable ; il ne peut ensuite y avoir de nouveaux échanges sans l'émission d'une nouvelle entrée par l'environnement.

Un **événement** est décrit avec un identifiant et un ensemble de paramètres typés qui constitue le contenu de l'événement. Des probabilités sont associées à tous les paramètres d'un événement d'entrée. Plus le système est compliqué plus la modélisation le sera. Pour cette raison, il est nécessaire de définir différents **niveaux hiérarchiques** [9] dans le modèle ; cela permet d'avoir une abstraction du modèle qui est plus facile à manipuler. De plus, le modèle permet de manipuler un ou plusieurs **profils** pour représenter le comportement d'utilisateurs différents.

Le premier intérêt de ce modèle est de permettre de valider le comportement dynamique des spécifications du système. Celui-ci gagne en intérêt à être construit tôt dans le cycle de vie du système. Le second intérêt du modèle est d'engendrer automatiquement des séquences de test pour valider le système lui-même. Ces séquences de test sont un bon échantillon de l'infinité de séquences de test possibles. De plus, dans la mesure où elles sont représentatives du comportement futur du logiciel, l'exécution de celles-ci permet d'avoir une estimation non biaisée de la fiabilité.

Une vérification du modèle d'usage est faite par des calculs numériques qui indiquent si les probabilités sont bien pertinentes. Par exemple, il est possible de déterminer la probabilité qu'un état donné figure dans une séquence de test, la distribution stationnaire de la chaîne qui permet de voir quelles seront les fonctionnalités le plus intensément testées ainsi que la longueur moyenne des séquences de test qui seront engendrées.

Définition des critères d'arrêt

Un autre point important est la définition des critères d'arrêt. Cette définition est, dans un grand nombre de compagnies, souvent incomplète, car elle doit réellement tenir compte des particularités du système à tester. Ces critères doivent reposer sur des principes solides, par exemple la conformité entre les spécifications et le système, la vérification d'objectifs de qualité (performances, temps de réponse), la fiabilité recherchée, la couverture fonctionnelle ou structurelle. Naturellement, cette liste n'est pas exhaustive, et il convient que ces critères soient définis en fonction de chaque livraison et de l'utilisation qui sera ensuite faite du logiciel.

En plus des objectifs de qualité qui sont eux propres à chaque type de logiciel développé, nous utilisons deux critères de livraison. Ces critères sont déterminés en fonction de la classe d'utilisation du logiciel. Le premier critère est l'obtention en fin de test d'une couverture fonctionnelle et le second est le nombre de pas de test effectués sans erreur. Ce dernier critère permet d'avoir une estimation anticipée de la fiabilité. Le nombre de pas de test est calculé grâce à la méthode du test d'hypothèse et de la loi binomiale [8], loi qui permet d'obtenir la relation suivante :

$$1 - C = F^N + NF(1-F)^{N-1}$$

où C est le seuil de confiance, F la fiabilité et N le nombre de pas de test.

Nous avons utilisé cette relation avec un seuil de confiance de 0,99 et avons déduit les critères d'arrêt respectivement associés à des fiabilités de 0,99, 0,995 et 0,999. Comme tous les types de logiciel ne nécessitent pas la même qualité, nous avons associé ces fiabilités à différentes classes de logiciel. Les données fournies dans le tableau suivant sont indiquées uniquement à titre d'exemple et doivent naturellement être discutées lors de la détermination des critères d'arrêt d'un logiciel qui entre en phase de test.

Tableau 1 : Critères d'arrêt

	Application sans fonctions critiques utilisée quelques heures par jour Fiabilité = 0,99	Application utilisée par plusieurs utilisateurs, de façon régulière Fiabilité = 0,995	Application avec des contraintes : temps réel, temps continu Fiabilité = 0,999
Couverture des états	90%	95%	100%
Couverture des arcs	80%	90%	99%
Nombre de pas de test à exécuter sans défaut	459	918	4603

Cet ensemble de critères d'arrêt permet de valider une grande partie de nos développements avec un niveau de qualité adaptée, c'est-à-dire un minimum de défauts trouvés en exploitation du système.

Comment estimer le temps nécessaire pour exécuter des tests ?

L'écriture du modèle d'usage peut sembler coûteuse, mais il est nécessaire de tenir compte du fait que la génération des séquences de test est automatique une fois le modèle disponible et que l'ensemble des séquences de test est potentiellement infini. Un autre avantage est que le système est largement plus complet et plus facile à maintenir qu'un ensemble de séquences de test. Alitec a noté que la construction du modèle prenait en moyenne entre 0,5 et 3 jours par 1000 lignes de code (ldc). Le nombre de jours est naturellement dépendant de la complexité et de la connaissance de la spécification, du nombre d'échanges avec l'environnement et de la complexité de la documentation.

Le temps pour exécuter les séquences de test doit être évalué en fonction des objectifs qualificatifs définis comme critères de livraison. Nous utilisons les résultats précédents sur les projets pour avoir une quantification plus précise. En fonction de la taille de la séquence de test à exécuter sans erreur et du nombre moyen de campagnes de test nécessaires pour atteindre cette taille, nous déterminons le temps nécessaire à allouer à la phase de test.

Tableau 2 : Temps d'écriture du modèle d'usage et de passage des tests

	Logiciel 1	Logiciel 2	Logiciel 3	Logiciel 4	Logiciel 5
Temps pour écrire le modèle (nombre de jours pour 1000 ldc)	0,24	1,27	0,55	1	0,61
Temps pour tester (nombre de jours pour 1000 ldc)	2,42	1,45	5,97	8,3	1,77

Sur le tableau 2 nous pouvons observer que le temps pour l'écriture d'un modèle d'usage est globalement similaire pour différents systèmes, alors que le temps de test peut être très variable. Le temps de test est dépendant d'un grand nombre de paramètres : la qualité recherchée a naturellement une incidence, mais également la stratégie de test, le temps d'exécution des tests qui peut varier en fonction du système entre 20 et 200 pas de test à l'heure.

4. L'EXÉCUTION DES TESTS ET LA GESTION DES DÉFAUTS

La difficulté de cette phase est l'enregistrement de suffisamment de données pour être en mesure de gérer sans ambiguïté le progrès de la phase de test. Pour cela, il est nécessaire d'avoir une bonne séparation des activités et des rôles bien définis dans la phase de test.

Dans le présent article, nous nous sommes intéressés seulement à la validation ; nous avons pour cela considéré trois types de tests. Le premier est le test « d'autoroute » qui consiste à vérifier que le comportement nominal du logiciel est correct. Le second type de test permet de vérifier quelques comportements spécifiques du logiciel que l'on ne souhaite pas valider avec les tests aléatoires, par exemple la performance d'une fonctionnalité particulière ou un test de résistance à un événement (exemple : reprise après une panne de secteur). Le troisième type de test est constitué par les séquences de test engendrées à partir du modèle d'usage. Ces séquences de test constituent la partie essentielle de la phase de test. Ce sont eux qui fourniront le plus de données sur la progression de la phase de test.

Les tests engendrés aléatoirement sont exécutés de la façon suivante :

- Exécution des tests.
- Si le test révèle un défaut bloquant, l'exécution de la séquence de test est stoppée et la séquence de test suivante est exécutée. Si plus de 50% des séquences de test révèlent un défaut et si au moins séquences de test ont été exécutées, les défauts sont corrigés et une nouvelle version du logiciel est produite.
- Génération d'une autre séquence de test pour la nouvelle version de logiciel. La taille de l'échantillon de séquences de test est déterminée en fonction du nombre de séquences de test exécutées sans erreur sur la version précédente,
- Le test est terminé quand la taille de la séquence de test est égale à celle définie dans le plan de test, et que les autres critères d'arrêt sont également satisfaits.

De cette manière, il est possible de garantir que le système sous test présente la qualité recherchée.

Un autre point est le processus de gestion des défauts ; il est important de classifier tous les défauts selon leur type et leur importance en y attachant un descriptif et la séquence de test permettant de les mettre respectivement en évidence. Pour chaque nouvelle version, nous décidons quels défauts vont être corrigés, en fonction de leur importance et de la facilité de leur correction. En effet, l'expérience a montré que lorsque l'on ne corrige pas tout de suite un défaut non bloquant, il n'est pas rare que lors de la poursuite des tests, les tests suivants révèlent plus d'information sur les cas où le défaut se déclare et permettent donc une correction plus rapide. Dans chaque nouvelle version, nous avons choisi de corriger 100% des défauts bloquants, 80% des défauts majeurs et quelques défauts mineurs.

5. GESTION DE LA CAMPAGNE DE TEST

Les données durant la phase de test sont enregistrées. Elles permettent de gérer la campagne de test et de déterminer si le délai de livraison est adéquat. Un outil interne, *Testor,* permet d'enregistrer les temps passés et le nombre de défauts détectés dans les différents types de tests (autoroute, complémentaires et statistiques). Nous enregistrons également les fonctionnalités ayant produit des erreurs. Ces données permettent d'avoir des mesures intéressantes. On peut distinguer deux types de mesures : celles reposant sur l'exécution du processus (temps de test, nombre de défauts,…) et celles reposant sur le modèle d'usage et sur les calculs de la chaîne de Markov (fiabilité, couverture fonctionnelle,…).

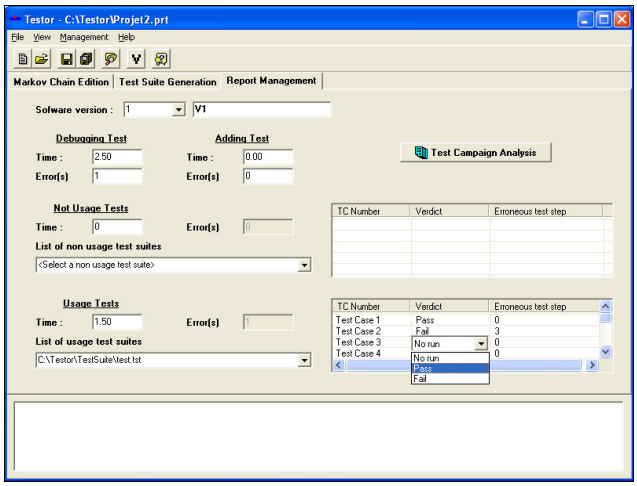

Figure 4 : Enregistrement des données durant la phase de test avec MaTeLo

Nous pouvons obtenir les mesures suivantes : l'intensité de défauts (nombre de défauts/nombre de pas de test exécutés sans erreur) par version, l'efficacité du type de test (temps pour trouver un défaut), le progrès de la couverture fonctionnelle.

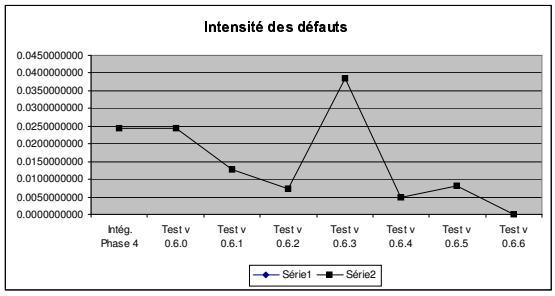

Figure 5 : Intensité des défauts

L'intensité des défauts permet de vérifier que le nombre de défauts trouvés décroît proportionnellement à l'effort de test. Par interpolation des points avec une courbe théorique, il est possible d'avoir une estimation du moment où le logiciel ne révèlera plus de défauts. Pour cela, il est nécessaire d'utiliser le temps comme axe des abscisses.

Sur l'exemple fourni par la Figure 4, on observe que l'intensité de défauts décroît bien au fil des versions, hormis lors de la version 0.6.3.

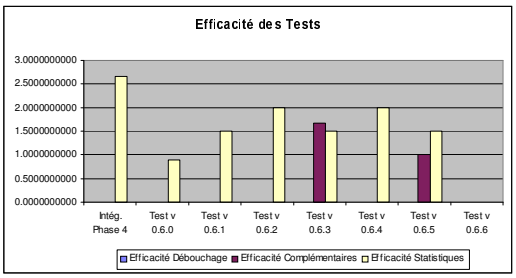

Figure 6 : Efficacité des tests

L'efficacité des tests est établie pour chaque type de test (autoroute, complémentaires et engendrés aléatoirement) ; cela permet d'identifier, par exemple, les types de tests qui sont les plus efficaces à un moment donné. Nous gardons les valeurs sur les projets pour être en mesure d'avoir des valeurs de référence qui permettent des comparaisons sur les projets en cours et de détecter ainsi d'éventuels problèmes. Nous observons sur la Figure 5 que le temps moyen pour trouver un défaut sur le projet cité en exemple à l'aide des tests statistiques est en général inférieur à 2 heures. Le contrôle de ces données permet de révéler dans la pratique si une stratégie de test est insatisfaisante.

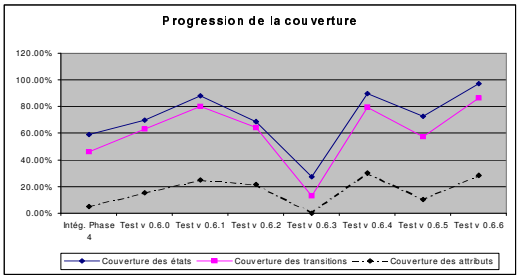

Figure 7 : Couverture fonctionnelle

Les couvertures des arcs et des états sont utilisées comme critère de livraison. Mais si les autres critères sont déjà satisfaits, il est possible d'atteindre plus rapidement la couverture par identification des états et des transitions non couvertes et de passer des séquences de test spécifiques permettant de les couvrir. À la fin des tests, il est nécessaire que les couvertures des états et des transitions soient proches de 100%.

Un grand nombre de mesures peut être déduit à l'aide de calculs sur les chaînes de Markov. Par exemple, nous pouvons calculer la fiabilité selon la méthode définie dans [8] ; ce résultat est issu de la chaîne de Markov en calculant la probabilité de rencontrer une erreur durant une réalisation du processus stochastique. Nous pouvons également identifier les fonctions non activées durant la phase de test et les fonctions qui ont produit le plus grand nombre d'erreurs.

6. L'OUTIL MATELO

Dans le cadre d'un projet Européen (IST 32-402), un outil MaTeLo pour le support de la méthodologie est développé. Ce projet regroupe 6 partenaires industriels : Alitec (France), Danet (Allemagne), Alenia Spazio (Italie), Israel Aircraft Industries (Israel), Magnetti marelli (France) et Nec(France) et 2 partenaires universitaires : l'université de Lund (Suède) et l'université d'Erlangen (Allemagne). Ce projet se terminera à la mi-2004, mais des premiers prototypes sont déjà utilisés par l'ensemble des membres du projet. L'outil MaTeLo, permet la gestion de la campagne de test à l'aide des chaînes de Markov pour la génération automatique. Cet outil est adapté à tous types de système et, en particulier, aux systèmes embarqués (automobile, aéronautique et télécommunications).
Cet outil inclut :

-Un **éditeur de modèles d'usage**, qui permet d'éditer graphiquement des graphes complexes répartis sur plusieurs niveaux hiérarchiques. Des fonctions de réorganisations automatiques, des vues arborescentes et tabulaires du graphe sont également incluses.
-Un **convertisseur** permettant de ré-utiliser des fichiers de spécifications décrites selon des **MSC-96** (Message Sequences Chart) ou des diagrammes de séquence d'**UML 2.0** au format XMI. Les diagrammes d'entrée sont convertis en modèles d'usage, que l'on peut compléter par la suite avec l'éditeur de modèles d'usage.
-Un **générateur de cas de test** à partir d'un modèle d'usage incluant plusieurs algorithmes que l'on utilise selon l'objectif de test : atteindre rapidement la couverture du modèle, augmenter la fiabilité, tester aux limites ou encore générer les séquences les plus probables. La sauvegarde des séquences de test peut se faire au format TTCN-3 ou dans un format textuel pour l'exécution manuelle. Il est également possible d'adapter les scénarios de test à TestStand.
-Un **compilateur** de séquence de test TTCN-3 permet d'exécuter automatiquement des séquences de test dans des environnements hétérogènes.
-Un a**nalyseur** de résultat de test, fournissant un rapport sur la campagne de test, permet de gérer la progression des tests.

Les bénéfices apportés aux futurs utilisateurs de cette technique sont immédiats : ces derniers seront en mesure d'augmenter et de gérer la fiabilité des logiciels développés grâce à des indicateurs quantitatifs relatifs à la gestion de la campagne de test ainsi que d'évaluer le temps restant pour atteindre la qualité souhaitée. Cette technique permet d'optimiser le temps utilisé et les ressources allouées au test.

7. CONCLUSION

La démarche proposée dans cet article couvre l'ensemble des étapes de la phase de validation, allant de l'établissement du plan de test jusqu'à la livraison du logiciel. La particularité de la méthode repose sur l'utilisation des méthodes statistiques pour la génération de cas de test, ainsi que pour la livraison du logiciel. L'expérience faite par Alitec de l'utilisation de la méthode présentée est concluante : d'une part le coût de la phase de test est maîtrisé, et d'autre part les logiciels livrés atteignent un niveau de qualité satisfaisant et prévisible.
L'intérêt, pour l'industrie, de l'utilisation des tests statistiques pour l'amélioration du processus de test est évident. L'avantage principal est le modèle d'usage qui permet la génération automatique de séquences de test représentatives. Cette technique de test peut être utilisée dans un grand nombre de projets pour estimer la qualité du logiciel. Actuellement, la société Alitec développe des méthodes pour adapter ces techniques à l'automatisation des tests, notamment en améliorant le modèle en vue de prendre en compte les standards ASN.1 et TTCN-3. Elle développe également des algorithmes pour la génération de séquences de test pour l'amélioration de la campagne de test en termes de délais et de coût.

Références

[1] Michael Dyer : *The Cleanroom approach to quality software developpement* ; Wiley and Sons, 1992.

[2] John D. Musa : *Software reliability engineered testing* ; McGraw-Hill, 1998.

[3] Simon Ntafos : *On random and partition testing ;* in International Symposium on Software Testing and Analysis, pages 42-48, mars 1998.

[4] Thomas J. Ostrand et Marc J. Balcer *: The category-partition method for specifying and generating functional tests ;* Communications of ACM, vol. 31, n°6, juin 1988.

[5] Stacy J. Prowell : *TML: a description language for Markov chain usage models* ; Information and Software Technology, 42(12), pp.835-844, 2000

[6] Jean-Claude Rault : *Extension of hardware fault detection models to the verification of software* ; in Program test methods, W. C. Hetzel (réd.), Prentice-Hall, 1973, pp. 255-262.

[7] Pascale Thévenod-Fosse et Hélène Waeselynck : *Statemate applied to statistical software testing*. Report 92485, Laboratoire d'Automatique et d'Analyse des Systèmes, 1993.

[8] James A. Whittaker : *Software testing and reliability analysis ;* PhD Thesis, University of Tennessee, mai 2002.

[9] Denise M. Woit : *A framework for reliability estimation* ; in Proc. 5th IEEE International Symposium on Software Reliability Engineering, pages 18-24, novembre 1994.