Rational®

IBM®

Une publication d'IBM comportant la recherche de Gartner

Juin 2010

Introduction 2

D'où vient la nécessité de tester ? 2

Les produits logiciels sont-ils testés systématiquement, doit-on les tester systématiquement ? 2

Le rôle de testeur est-il un métier à part entière ? 3

Terminologie 4

Le testing sur le cycle de développement de logiciel en deux exemples 6

Testing basé sur les cas d'utilisation 6

L'Assurance Qualité avec DFSS. 12

Conclusion 15

Normaliser les définitions et les attentes des activités de test 16

Test ou Assurance Qualité : pile ou face ?

Un guide pratique à l'intégration de la qualité de logiciel dans le cycle de vie de développement

Gartner®

Introduction

Il existe un sujet sur lequel tous les professionnels de développement de logiciel s'accordent : il faut tester les produits basés sur le logiciel – applications, produits logiciels, logiciel embarqué et autres.

D'où vient la nécessité de tester ?

Le développement de logiciel est un processus de résolution d'un problème rencontré par des individus ou des entreprises pour lesquels la solution la plus durable et économique est un logiciel. Ce processus est gouverné par des exigences qui ne sont que la transcription des besoins de ces individus ou de ces entreprises dans un produit final dans lequel les parties prenantes collaborent pour satisfaire ces besoins. Le manque de communication lors de cette collaboration peut produire des exigences ambigües, incomplètes ou bien incompréhensibles voire parfois incorrectes. Les exigences sont la cause principale d'apparition de défauts dans le logiciel[1]. A cela s'ajoute des *oublis* ou des problèmes de conception et des défauts liés à l'implémentation du logiciel.

Chaque logiciel peut être considéré comme un ensemble de risques portés aux entreprises, aux utilisateurs et à l'environnement dans lequel il s'exécute. Le testing est une méthode qui permet de réduire ce risque stratégique porté par le logiciel.

Le test de logiciel, s'il est mis en place correctement, permet de satisfaire les intérêts des différentes parties prenantes (utilisateurs, entreprises et autres). Par conséquent, le but du test de logiciel ne se résume pas au simple processus de recherche de bogues mais consiste aussi à assurer que la problématique posée par les parties prenantes est bien résolue par le logiciel. Ceci constitue un changement majeur d'attitude dans le monde de développement où le testeur est reconnu comme étant LE premier utilisateur du système et, en tant que tel, son retour sur l'utilisation du logiciel devient primordial pour l'ensemble de l'équipe.

A ce jour, l'industrie du développement de logiciel n'a pas encore trouvé la recette d'un processus de développement parfait qui rendra le test logiciel obsolète. Les études menées par Standish Group[2] montrent que 44% des projets dans le monde ont été challengés soit par le budget, soit par des retards, ou bien par le fait qu'ils ont délivré moins de fonctionnalités dans le périmètre prévu. Le travail supplémentaire (rework) lié à la correction des exigences, de la conception et de l'implémentation, est responsable de 40 à 50% du coût total du projet de développement[3]. On peut se demander si l'intégration de la gestion de la qualité sur l'ensemble du processus de développement au sens large est la recette magique qui a permis aux 32% de projets lancés en 2009 d'atteindre le succès (source Standish Group).

Les produits logiciels sont-ils testés systématiquement, doit-on les tester systématiquement ?

Bien entendu, la réponse que nous souhaiterions entendre est : oui. Mais dans la pratique, il y a des écarts importants en fonction des équipes de livraison de logiciels. Les recherches récentes menées par un cabinet d'études[4] ont révélées que plus de 40% des applications sont livrées avec 1 à 10 défauts critiques. Dans certains projets, surtout dans les projets Agiles, on considère souvent que les tests unitaires sont suffisants pour garantir le niveau de qualité requis. Or statistiquement, il est prouvé que pour 1000 lignes de code[5], il faut s'attendre à avoir 6,5 à 9 défauts lors du test système. De même, 7% des défauts corrigés vont engendrer un nouveau défaut[6]. Les tests unitaires sont absolument nécessaires au même titre que la revue statique et automatique du code ainsi que le profiling dynamique du code.

[1] "Software Testing", R .Patton, p.17, Sams, 2005

[2] http://www1.standishgroup.com/newsroom/chaos_2009.php

[3] Steve McConnell, "Software Quality at Top Speed", http://www.stevemcconnell.com/articles/art04.htm

[4] http://blog.testlabs.com/2009/04/software-testing-and-quality-assurance.html

[5] http://blog.testlabs.com/2009/04/software-testing-and-quality-assurance.html

[6] Software Cost Estimating Methods for Large Projects, Capers Jones, 2005, http://www.stsc.hill.af.mil/crosstalk/2005/04/0504Jones.html

Le testing par les testeurs permet de réaliser un changement de perspective et des méthodes de test qui, au final, contribueront à un logiciel de meilleure qualité. Selon Grady[7], les testeurs trouveront un défaut toutes les quatre heures en moyenne lors de tests boîte noire.

Le rôle de testeur est-il un métier à part entière ?

Avec le temps, les logiciels et les tests associés deviennent de plus en plus complexes. Ceci a pour conséquence une plus grande professionnalisation des tests.

En effet, pour concevoir et produire les tests qui valideront un logiciel de manière efficace et efficiente, le testeur doit :

- connaître les méthodes statistiques,
- avoir une bonne connaissance des méthodes de développement et de la problématique métier adressée par le logiciel en développement pour anticiper l'apparition des défauts,
- bien connaître et appliquer la multitude de techniques utilisées pour tester le logiciel (test de fonctionnalité, de performance, de fiabilité, de sécurité, de non régression, …),
- définir les bonnes stratégies de test selon les objectifs de l'équipe projet,
- programmer pour accroître l'efficacité de l'automatisation de test,
- avoir d'autres compétences et qualités importantes.

Le large éventail de compétences demandées aux testeurs prouve que le testing est un métier à part entière. Pour devenir un vrai professionnel du testing, il faut une formation adaptée, une expérience et une attitude spécifique qui permettront de challenger le logiciel.

En résumé, voici les trois observations importantes pour le reste de cet article :

1. Le processus de test est un processus majeur au sein des équipes de développement. Ce processus est fortement lié, dépendant et connecté aux autres activités telles que la définition et la gestion des exigences, la conception, la gestion des changements et des configurations et autres disciplines. C'est cet ensemble qui, mis en place correctement, permettra d'améliorer la qualité, de réduire les délais et les coûts des projets.
2. Une seule technique de test n'est pas suffisante pour assurer la qualité d'un logiciel. Au-delà de l'application des techniques différentes de test, c'est l'ensemble de la chaîne de production du logiciel qui est responsable de la qualité de ces logiciels.
3. Les professionnels du test doivent posséder de nombreuses compétences pour réaliser des tests efficaces et efficients de logiciels.

Le but de cet article est de montrer, du point de vue du testeur, le lien entre l'assurance qualité et l'ensemble du cycle de vie applicatif sur deux exemples.

Source: IBM

[7]Practical Software Metrics for Project Management and Process Improvement, Robert B. Grady, Prentice Hall, 1992

Terminologie

Avant de passer à la suite, précisons quelques définitions importantes de vocabulaire.

Testing

Le processus d'utilisation d'un système ou d'une composante sous des conditions bien définies qui, au travers de l'observation et de l'enregistrement des résultats, permet de faire une évaluation de certains aspects de ce système ou de ces composantes. Le Testing valide la qualité d'un système ou d'une composante.

Objectif du testing

La définition de l'objectif de testing (ou mission de test) influencera profondément la manière dont le logiciel sera testé. L'objectif de testing fera partie des *exigences de testing*. Voici quelques exemples d'objectifs de test :

- Trouver des scénarii d'utilisation sûre du produit (malgré les défauts, trouver des scénarii où le logiciel fonctionne)
- Maximiser le nombre de défauts trouvés
- Aider à la prise de décision : livrer ou non le produit logiciel
- Evaluer la qualité
- …

En fonction des objectifs, les testeurs seront amenés à choisir des approches de test différentes.

Stratégie de test

La stratégie de test (ou l'approche de test) spécifiera comment atteindre l'objectif de manière efficace et la plus économique possible. La stratégie de test définira les techniques de test qui seront employées par les testeurs ainsi que la façon dont elles seront employées. Néanmoins, la meilleure stratégie de test répondra aux cinq critères suivants :

- Diversifiée
- Axée sur les risques
- Spécifique au produit
- Pratique
- Défendable

Il existe plus de 200 techniques de test documentées : test boîte blanche, test fonctionnel, test de non régression, test exploratoire, test axé sur les risques, test de performance, test basé sur les cas d'utilisation, test basé sur la table de décision…

Motivateur de test

La stratégie de test est guidée par les motivateurs de test. Un motivateur de test est une information ou une combinaison d'informations qui motivent une action de test. Par exemple un motivateur de test peut être une exigence à couvrir un risque de mise en évidence d'un dysfonctionnement sur une fonctionnalité, la prise en compte d'une demande d'évolution, la vérification de correction d'une anomalie,... Les motivateurs de test sont analysés lors de la définition de la stratégie de test. Ils aident à définir les priorités de test et à aligner l'équipe de test sur ces priorités.

Test

Le test est une ***activité*** dans laquelle un système ou un composant est exécuté dans des conditions spécifiées, les résultats sont observés ou enregistrés, et l'évaluation d'un certain aspect du système ou du composant est réalisée.

Le standard IEEE 829 définit le test comme :

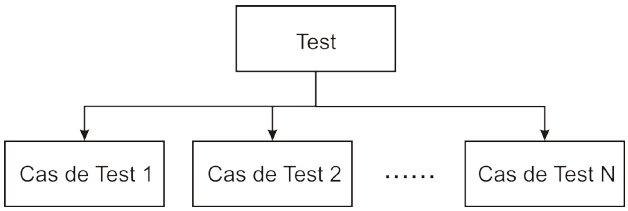
- (A) un ensemble d'un ou plusieurs cas de test,
- (B) un ensemble d'une ou plusieurs procédures de test,
- (C) un ensemble d'un ou plusieurs cas de test / des procédures de test[8].

Cas de test

Afin de pouvoir mettre en œuvre un test, il est nécessaire de connaître toutes les conditions dans lesquelles l'exécution aura lieu, la spécification de l'ensemble des entrées et des résultats attendus. C'est cela que nous appelons un **cas de test** (Figure 1). Le cas de test sera la plus petite entité qui est toujours exécutée comme une unité, du début à la fin.

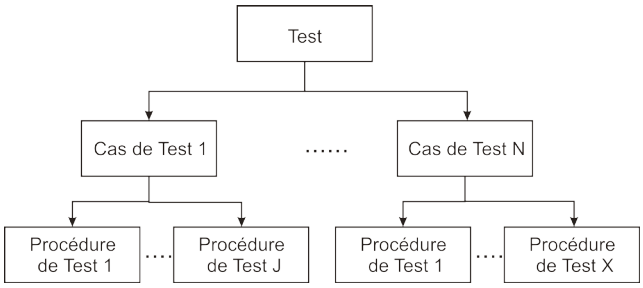
La notion d'un cas de test reste centrale dans le testing. Le cas de test est une fondation pour la conception et l'implémentation de procédures de test. Il identifie et communique clairement les conditions

[8]http://wilma.vub.ac.be/~se1_0607/svn/bin/cgi/viewvc.cgi/documents/standards/IEEE/IEEE-STD-829-1998.pdf?revision=45



Source: IBM

Figure 1 Relation entre un test et des cas de test



Source: IBM

Figure 2 Relation entre un test, un cas de test et une procédure de test

mises en œuvre par le test. Il a pour objectif de vérifier l'implémentation réussie et/ou acceptable des exigences vers le produit logiciel. Le cas de test permet aussi de dimensionner l'effort de test, de le mesurer et d'en faire le reporting. Il permet aussi de gérer les changements.

Procédure de test

Il existe de multiples façons de réaliser un test. Toute la réalisation, appelée procédure de test, doit fournir des instructions détaillées pour la mise en place, l'exécution et l'évaluation des résultats pour un cas de test. La procédure de test peut être réalisée par un script de test à exécuter manuellement ou peut être automatisée par exemple avec IBM Rational Functional Tester. La Figure 2 montre la relation entre un test, un cas de test et une procédure de test.

L'effort de test d'un système logiciel ne doit pas être considéré par les équipes projet comme une activité à part. En effet, le testing s'appuiera fortement sur les exigences envers le logiciel. La qualité des exigences va non seulement définir le processus de test, mais aussi impacter significativement la conception et l'implémentation. Elle aura une influence majeure sur les délais de livraison du logiciel. Ainsi, la maturité d'équipe en termes de gestion des changements déterminera le degré de facilité avec lequel l'équipe sera capable de mener l'analyse d'impact liée à un changement sur la conception, l'implémentation et les artefacts de test. Autrement dit : « Garbage in = garbage out ».

Source: IBM

Le testing sur le cycle de développement de logiciel en deux exemples

La portée et l'interdépendance des artefacts issus de disciplines différentes sur le cycle de vie de développement de logiciel soulignent l'importance croissante de la collaboration dans l'équipe et, au-delà, dans le contexte de projet ainsi que le partage efficace des informations entre les membres de l'équipe et les parties prenantes. Ces challenges sont adressés par la plateforme collaborative de nouvelle génération Jazz, d'IBM Rational. Grâce à l'automatisation des tâches, aux rapports automatiques et aux mécanismes d'intégration du cycle de vie de développement, les outils basés sur IBM Jazz accèdent facilement aux informations gérées par l'un et par l'autre. L'ensemble des artefacts test tels que les exigences de test, le test plan, le cas de test et autres, sont gérés par IBM Rational Quality Manager. Mais pour réaliser une collaboration efficace, Rational Quality Manager est capable de lier les cas de test aux exigences capturées par Rational Requirements Composer et gérées dans Rational DOORS ou dans Requisite Pro. De la même manière, Rational Quality Manager propose des capacités basiques de gestion et de suivi des anomalies. Il peut rattacher les anomalies détectées aux demandes de changement dans Rational ClearQuest ou aux work items de Rational Team Concert. De même, Rational Quality Manager permet une automatisation efficace des procédures de test avec Rational Functional Tester et Rational Performance Tester. Il permet aussi la reprise des automates de test des éditeurs tiers comme HP.

Les deux exemples qui suivent montreront l'importance de l'intégration des activités de test sur le cycle de vie de développement, sans pour autant traiter la question de gestion de testing.

Testing basé sur les cas d'utilisation

Application

L'application que nous étudions est une application web de vente des articles via Internet. Elle propose une interface conviviale pour sélectionner, choisir et acheter les articles en ligne, en fonction de leur disponibilité en stock. Le paiement se fait avec une carte bancaire et est réalisé en ligne par une transaction sécurisée avec la banque. Une fois l'article acheté, un ordre est envoyé au système de traitement des commandes et le stock d'articles est mis à jour en fonction de la quantité achetée. L'application propose aussi une interface de gestion ainsi qu'un mécanisme d'authentification de ces utilisateurs.

L'organisation et l'équipe projet

Au sein de la direction des études, une équipe est désignée pour réaliser ce projet en collaboration avec l'organisation fonctionnelle. L'équipe projet compte également un responsable de test, qui joue aussi le rôle de testeur, et un autre testeur. L'équipe projet utilise IBM Rational Requirements Composer pour définir et capturer des exigences en collaboration étroite avec des parties prenantes. Elle utilise, également, Rational RequisitePro pour gérer ces exigences et Rational Quality Manager avec Rational Functional Tester pour gérer et automatiser la campagne de test.

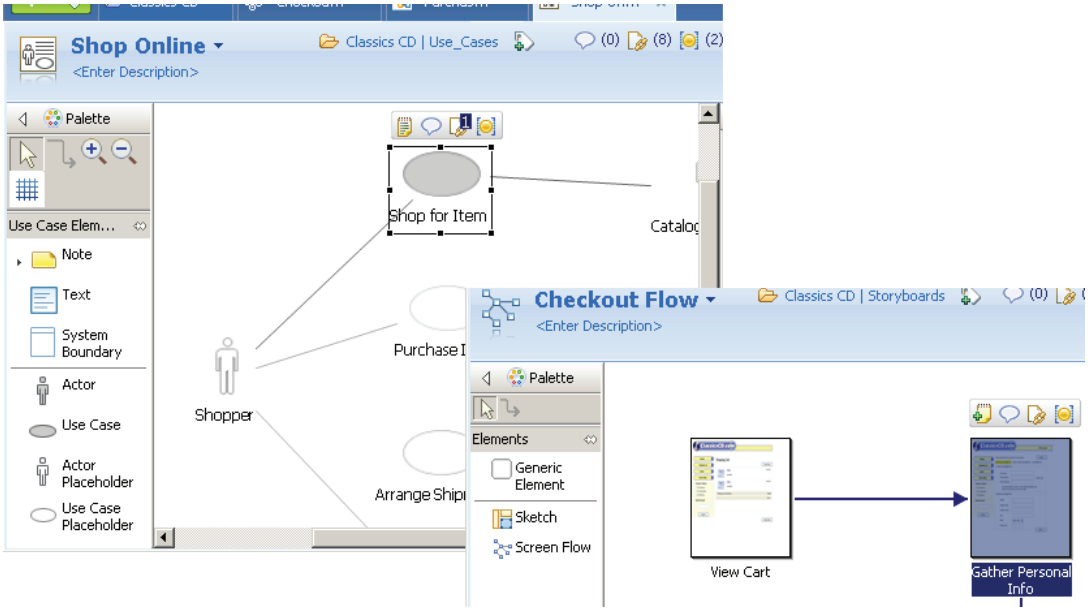
Afin d'obtenir la compréhension du système et de converger rapidement et efficacement sur son périmètre avec des parties prenantes, l'équipe projet a décidé de définir les exigences à travers des cas d'utilisation (Figure 3). Ceci favorise une approche collaborative à la définition des exigences, permet une meilleure compréhension des exigences et du logiciel en développement par toutes les parties. Le choix de la définition des exigences par des cas d'utilisation permet à l'équipe de s'occuper de tests tôt dans le cycle de développement. Bien que le processus de définition des exigences reste hors du scope de cet article, notons néanmoins que la manière dont les cas d'utilisation sont écrits affectera leur testabilité.

En ce qui concerne les applications web, les internautes attachent une grande importance à la sécurité de leurs données qu'ils échangent avec l'application. De plus, les critères comme la réactivité, la convivialité et la disponibilité de l'application sont déterminants pour générer d'avantage de trafic et assurer le succès du site.

Pour garantir le succès de l'application et réaliser les objectifs business, l'équipe projet a décidé d'impliquer les testeurs de manière formelle dans la validation des exigences, et ce dans le but de vérifier leur testabilité.

Mise en place de tests fonctionnels basés sur les cas d'utilisation

Bien que ce projet utilise plusieurs approches de test, cet article ne prend en considération que la réalisation de la technique de test fonctionnel basé sur les cas d'utilisation.



Source: IBM

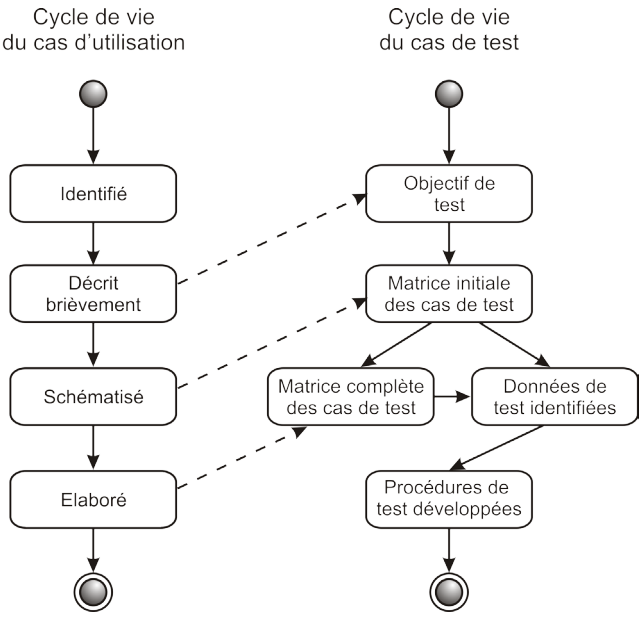
Figure 3 Définition des exigences avec les cas d'utilisation et IBM Rational Requirements Composer

Les tests basés sur les cas d'utilisation ont pour avantage de tester le système d'un bout à l'autre en fonction de son utilisation. Notons également les avantages suivants : différer la décision d'automatisation de tests avec les choix technologiques pour l'automatisation des tests.

Cette technique de test permet aux testeurs d'entamer la démarche d'assurance qualité à partir des exigences présentées en format de cas d'utilisation. En effet, cette technique favorise le développement et la réalisation de campagnes de test par itérations où les testeurs peuvent définir des cas de test même si l'application n'existe pas encore dans le code exécutable. De manière conceptuelle, le schéma (Figure 4) montre l'évolution des cas d'utilisation et des cas de test correspondants en fonction du cycle de vie de développement.

Etape 1 : Définition de l'objectif de test

L'équipe élabore le plan de test au démarrage du projet. L'équipe se met d'accord avec les parties prenantes sur l'objectif global de test – tests d'acceptation (Figure 5). Le caractère itératif de développement oblige les testeurs à définir des objectifs spécifiques de test pour chaque itération : par exemple, valider que l'architecture de l'application soit conforme aux performances requises sur les itérations précoces. Il faut noter que le plan de test évoluera avec chaque itération, tout au long du cycle de vie du projet.



Source: IBM

Figure 4 Le rapport entre le cycle de vie cas d'utilisation et le cycle de vie de ces de test

Le testing sur le cycle de développement de logiciel en deux exemples

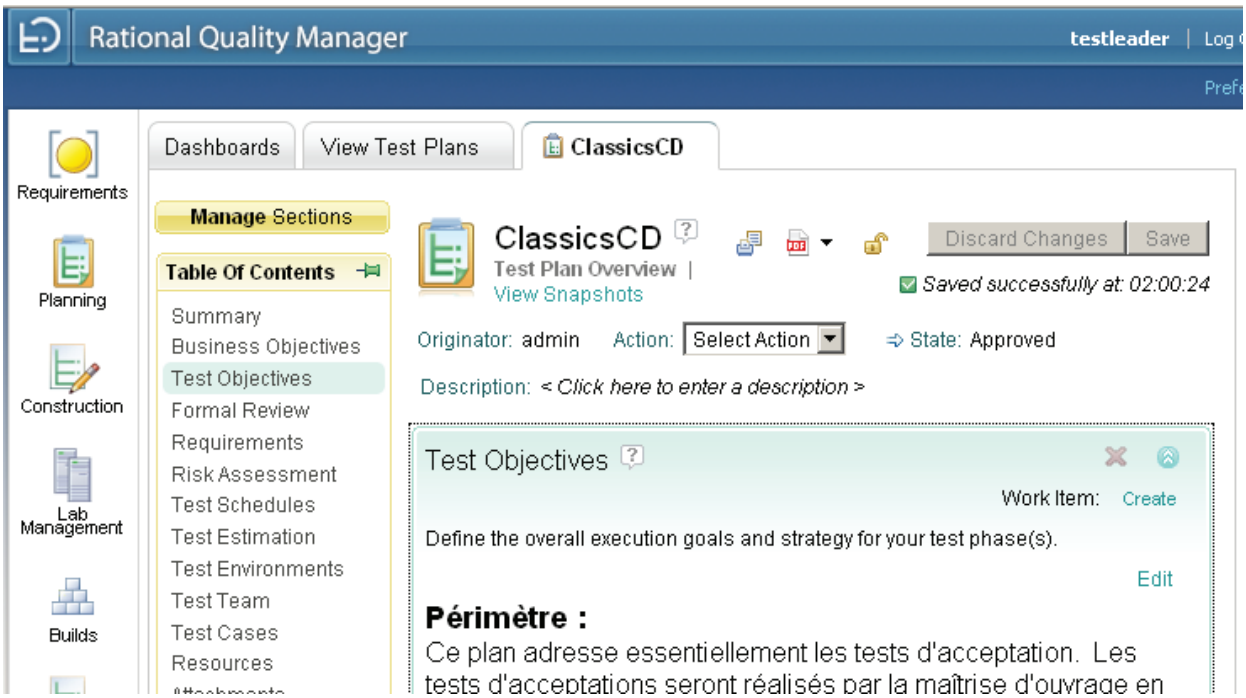


Figure 5 Définition de l'objectif de testing Source: IBM

Les motivateurs de test sont, dans Rational Quality Manager (Figure 6), associés aux exigences fonctionnelles, aux exigences non fonctionnelles, aux exigences de sécurité et à d'autres types d'exigences gérées dans le référentiel d'exigences, Rational RequisitePro. Cependant, d'autres types d'informations, telles que les anomalies, les requêtes d'amélioration et autres peuvent servir de motivateur de test.

Lors de l'élaboration de la stratégie de test, le responsable de test analyse le niveau de risque attaché à l'exigence, la priorité métier de l'exigence et son impact sur l'architecture pour définir une priorité de test en fonction de la criticité de l'exigence. Un motivateur de test avec une priorité de test élevée nécessitera une attention plus importante. Ensuite, le responsable de test associe un niveau de risque sur le motivateur du point de vue de l'activité de test. Ce niveau de risque est d'autant plus élevé que la priorité de test est importante et que la mise en œuvre du test sur cette fonctionnalité est difficile.

Etape 2 : Identification de cas de test

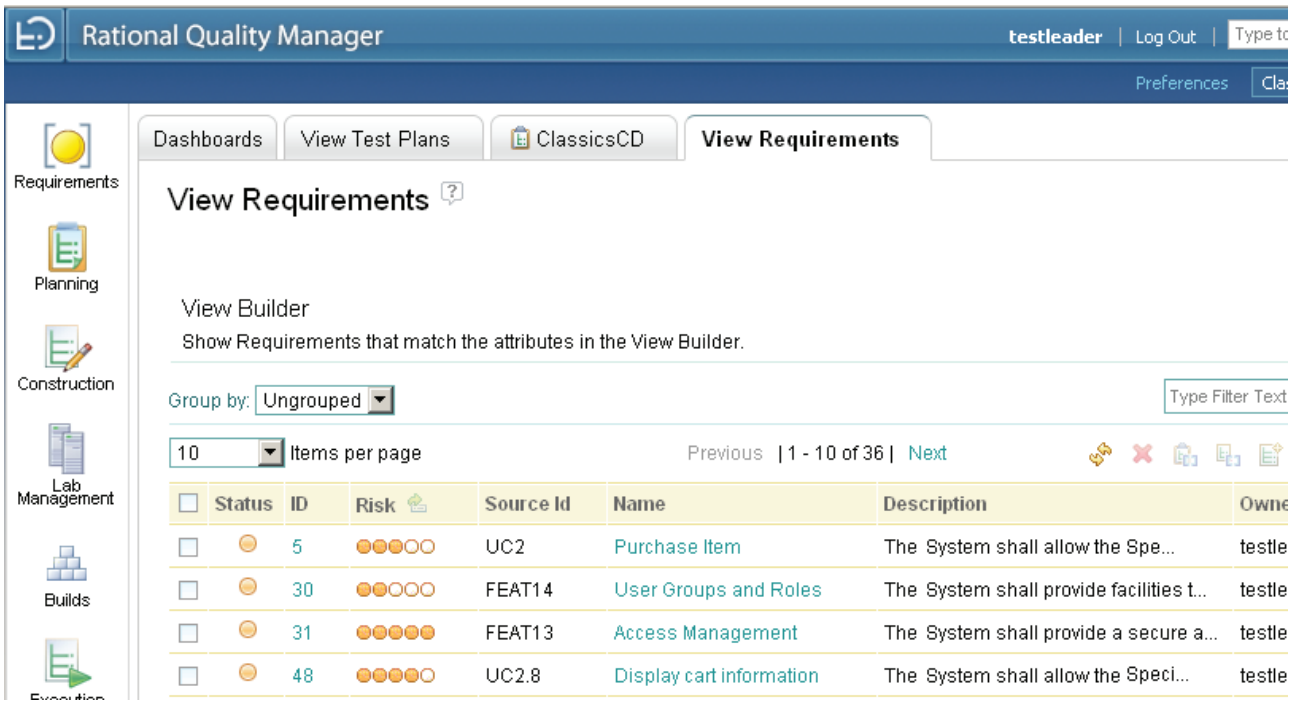
Dès que la description schématique du cas d'utilisation (outline) est prête, il est possible de définir les scénarii correspondants (Figure 7). Ces scénarii vont permettre de créer une matrice initiale de cas de test.

Pour chaque scénario unique, l'équipe de test crée un cas de test éventuel (une ligne dans la matrice) et identifie les conditions qui déclencheront l'exécution de ce cas de test. Chaque condition trouvée dans le cas d'utilisation ajoutera une colonne dans la matrice de cas de test. Pour chaque cas de test identifié, l'équipe de test détermine si la condition sera remplie (V[alide]) ou non (I[nvalide]) et le résultat attendu de l'exécution du scénario (Figure 8).

A ce stade, l'équipe de test ne fournit pas de valeurs pour les conditions identifiées. Ceci permet de voir facilement les conditions testées et de déterminer si la couverture par les cas de test est suffisante. Il est également possible de valider de manière visuelle si la matrice contient toutes les conditions et tous les cas de test nécessaires selon deux règles :

- S'il n'y a pas de condition « Invalide » sur la ligne dans la matrice, alors il manque (sauf pour le cas nominal) une condition.
- S'il n'y pas de condition « Invalide » dans une colonne de la matrice, alors il manque d'un cas de test.

La matrice initiale permet à l'équipe de test d'amorcer la pompe en créant les premiers cas de test qui seront liés aux exigences exprimées par des cas d'utilisation. Ces cas de test seront créés dans



Source: IBM

Figure 6 Exigences projet accessibles dans Rational Quality Manager

Nom de scenario	Flux initiale	Flux alternative	Flux alternative	Flux alternative
Purchase Item	FLUX DE BASE			
Unidentified Shopper	FLUX DE BASE	INTERNAUTE INCONNU		
Panier vide	FLUX DE BASE	PANIER VIDE		
Insufficient Items in Stock	FLUX DE BASE	STOCK INSUFFISANT		
User quits	FLUX DE BASE	INTERNAUT ABANDONE		
New shipping/billing address	FLUX DE BASE	GERER ADRESSES D'INTERNAUTE	NOUVELLE ADRESSE	
Different new shipping/billing address	FLUX DE BASE	GERER ADRESSES D'INTERNAUTE	NOUVELLE ADRESSE	DEMANDE JUSTIFICATIFS
Different shipping and billing addresses	FLUX DE BASE	GERER ADRESSES D'INTERNAUTE	DEMANDE JUSTIFICATIFS	
Manage customer adresses	FLUX DE BASE	GERER ADRESSES D'INTERNAUTE		
Invalid payment data	FLUX DE BASE	ERREUR DONNEES BANCAIRES		
Payment system unavailable	FLUX DE BASE	SYSTÈME BANCAIRE INDISPONIBLE		
Arrange Shipment	FLUX DE BASE			

Figure 7 Scénarii pour les cas d'utilisation

Source: IBM

Test case ID	Scenario	Customer ID	Mot de pass	# CB	Transac CB	Panier	Nouvelle adresse	Dispo en stock	Même adresse	Utiliser adresse par défaut	Résultat attendu
PU00	Purchase Item	V	V	V	V	V	n/a	V	n/a	V	Confirmation d'achat
PU01	Unidentified Shopper	I	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	Message d'erreur – fin
PU02	Panier vide	V	V	n/a	n/a	I	n/a	n/a	n/a	n/a	Msg d'erreur – continue
PU03	Insufficient Items in Stock	V	V	V	n/a	V	n/a	I	n/a	n/a	Msg d'erreur – quantité valide -continue
PU04	User quits	V	V	n/a	n/a	n/a	n/a	n/a	n/a	n/a	Fin
PU05	New shipping/billing address	V	V	n/a	n/a	V	V	V	V	I	Msg de demande de justif - panier sauvegardé - Fin
PU06	Different new shipping/billing address	V	V	n/a	n/a	V	V	V	I	I	Msg de demande de justif - panier sauvegardé - Fin
PU07	Different shipping and billing addresses	V	V	n/a	n/a	V	I	V	I	I	Msg de demande de justif - panier sauvegardé - Fin
PU08	Manage customer adresses	V	V	V	V	V	V	V	n/a	I	Confirmation d'achat
PU09	Invalid payment data	V	V	I	n/a	V	n/a	V	n/a	V	Msg d'erreur - fin
PU10	Payment system unavailable	V	V	V	I	V	n/a	V	n/a	V	Msg d'erreur - fin

Figure 8 Matrice initiale des cas de test

Source: IBM

Le testing sur le cycle de développement de logiciel en deux exemples

Rational Quality Manager. La Figure 9 montre les cas de test créés sous IBM Rational Quality Manager. Le symbole dans la colonne Status de cette capture d'écran (Figure 9) signifie que le Quality Manager a détecté une incohérence – un lien suspect – entre le(s) test(s) motivateur(s) et un cas de test. Cette incohérence est provoquée par un changement soit dans le(s) test(s) motivateur(s), soit dans le cas de test. IBM Rational Quality Manager propose un accès facile à toutes les informations nécessaires qui permettent au responsable de test de prendre les décisions nécessaires. La Figure 10 montre les exigences qui sont associées à un cas de test et qui sont détectées comme suspectes par IBM Rational Quality Manager.

Etape 3 : Mise au point de cas de test

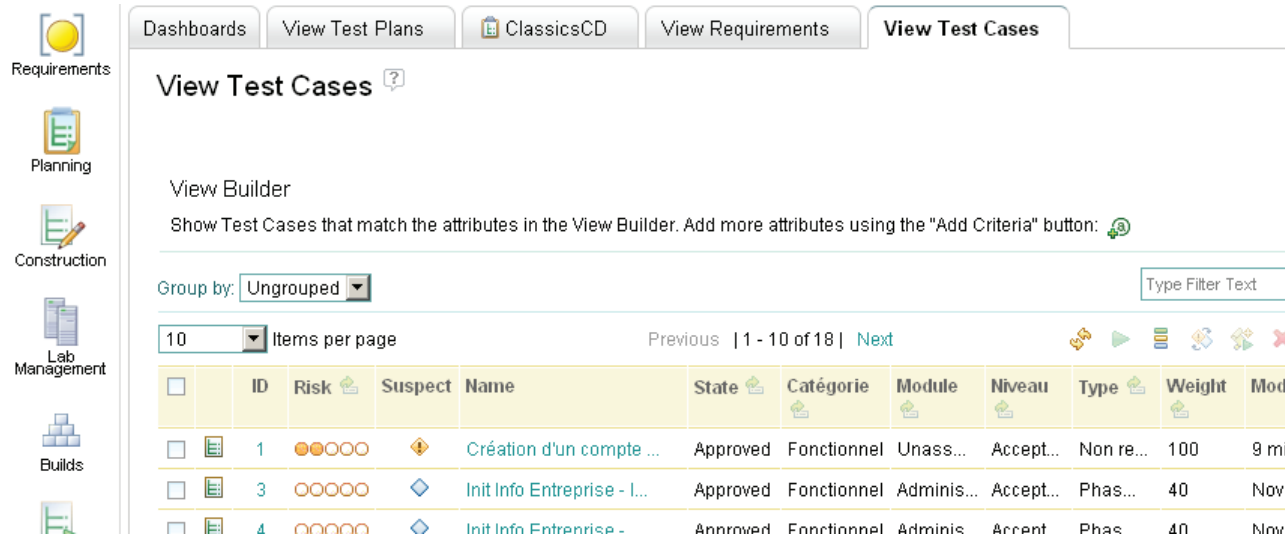
A ce stade, la matrice initiale des cas de test est prête. L'effort principal se porte sur la réconciliation, la revue des cas de test, l'ajout des cas de test nécessaires où la suppression des cas de test redondants.

L'équipe améliore la matrice initiale des cas de test en assignant des valeurs réelles aux conditions (Figure 11).

En fonction des impératifs de chaque itération, ceux du projet et selon l'avancement de ce projet, l'équipe de test pourra prendre la meilleure décision : automatiser ou non les procédures de test, procéder à l'automatisation,… Ceci grâce à IBM Rational Quality Manager.

Etape 4 : Réalisation et exécution des procédures de test

Dans un premier temps, les procédures de test sont réalisées avec Rational Quality Manager à travers les scripts manuels. Ceci permet de spécifier les procédures de test même si l'application n'existe pas encore. Puis, avec l'implémentation incrémentale de la fonctionnalité dans l'application, l'équipe de test décide d'implémenter certaines procédures de test dans des scripts automatiques réalisés avec Rational Functional Tester. Grâce à ces capacités de reconnaissance des éléments d'interface graphique d'utilisateur avancées, Rational Functional Tester permet une automatisation simple et pratique des tests fonctionnels. Il capture les interactions faites avec l'interface utilisateur et les enregistre dans un script. Le testeur peut facilement définir les points de vérification et variabiliser les données en utilisant des data pools au cours du processus d'enregistrement du script. Une fois que les scripts sont enregistrés le testeur peut modifier, supprimer ou ajouter les éléments : données dans le data pool, les data pools, les points de vérification. Ensuite ces scripts de test seront utilisés lors de l'exécution des campagnes de test.



Source: IBM

Figure 9 Les cas de test gérés par IBM Rational Quality Manager

Figure 10 Test motivateur d'un cas de test

Source: IBM

Test case ID	Scenario	Customer ID	Mot de pass	# CB	Transac CB	Panier	Nouvelle adresse	Dispo en stock	Même adresse	Utiliser adresse par défaut	Résultat attendu
PU00	Purchase Item	gdelong@yahoo.com	gser12Zq	12563400012395	# dynamique	ref #45690123	n/a	oui	n/a	oui	Confirmation d'achat
PU01	Unidentified Shopper	alen.craft@mycomp.co	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	Message d'erreur – fin
PU02	Panier vide	anick.gaiel@qsxe.net	anick123	n/a	n/a	vide	n/a	n/a	n/a	n/a	Msg d'erreur – continue
PU03	Insufficient Items in Stock	yann@gmail.com	yann0145q	3245678900070	n/a	ref #34518905 ref #37894567	n/a	pas de ref #37894567	n/a	n/a	Msg d'erreur – quantité valide -continue
PU04	User quits	michel.glen@hotmail.c	ft567qa1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	Fin
PU05	New shipping/billing address	jeromeb@myadress.or	jerome	n/a	n/a	ref #568910381	23, rue de Londres 75008 Paris	oui	oui	no	Msg de demande de justif - panier sauvegardé - Fin
PU06	Different new shipping/billing address	nicoleb@everywhere.c	dfe34qA	n/a	n/a	ref #451239003	1234, Thompson street Tween Peaks, 12345 CA USA	oui	no	no	Msg de demande de justif - panier sauvegardé - Fin
PU07	Different shipping and billing addresses	amin.dfr@free.fr	QSaaa123d	n/a	n/a	ref #341677881	1, rue Laplace 94332 Marne La Valle	oui	no	no	Msg de demande de justif - panier sauvegardé - Fin
PU08	Manage customer adresses	daniel.groove@automa	groove.com	56712300980012	# dynamique	ref #231256729 ref #120098123 ref #666214902	105, bvd Montparnasse 75005, Paris France	oui	n/a	no	Confirmation d'achat
PU09	Invalid payment data	amigo@numericable.n	123bravo_5	45666111000233	n/a	ref #129990124	n/a	oui	n/a	oui	Msg d'erreur - fin
PU10	Payment system	anne@monmail.org	23456A7ERT	53466689000123	msg erreur	ref #1903471256	n/a	oui	n/a	oui	Msg d'erreur - fin

Figure 11 Matrice complète de cas de test

Source: IBM

Le testing sur le cycle de développement de logiciel en deux exemples

Cette automatisation de test permet à l'équipe de construire une suite de tests de non-régression exécutée contre chaque nouvelle fabrication (build) livrée aux testeurs. Tout problème détecté lors du test est rapporté via des demandes de correction soumises dans Rational Quality Manager et accessibles dynamiquement au reste de l'équipe de développement dans IBM Rational Team Concert.

Les résultats de l'exécution de plan de test sur une des itérations sont présentés en Figure 12.

IBM Rational Quality Manager propose des rapports visuels et synthétiques comme, par exemple, celui présenté en Figure 13.

Résultats

L'exécution des cas de test permet de trouver des problèmes et d'évaluer la qualité globale du système. Etant donné que les exigences sont liées aux cas de test, il est possible :

- de déterminer quelle est la couverture de ces exigences par les tests,
- d'apporter un jugement sur le fait que le logiciel sous le test se stabilise par rapport aux problèmes rapportés dans le temps,
- d'évaluer s'il faut changer les techniques de test face à une immunisation possible du système par rapport à la détection de nouveaux problèmes, etc.

Mais cette vision d'ensemble n'est possible que grâce à l'intégration de testing sur le cycle de vie de développement de l'application.

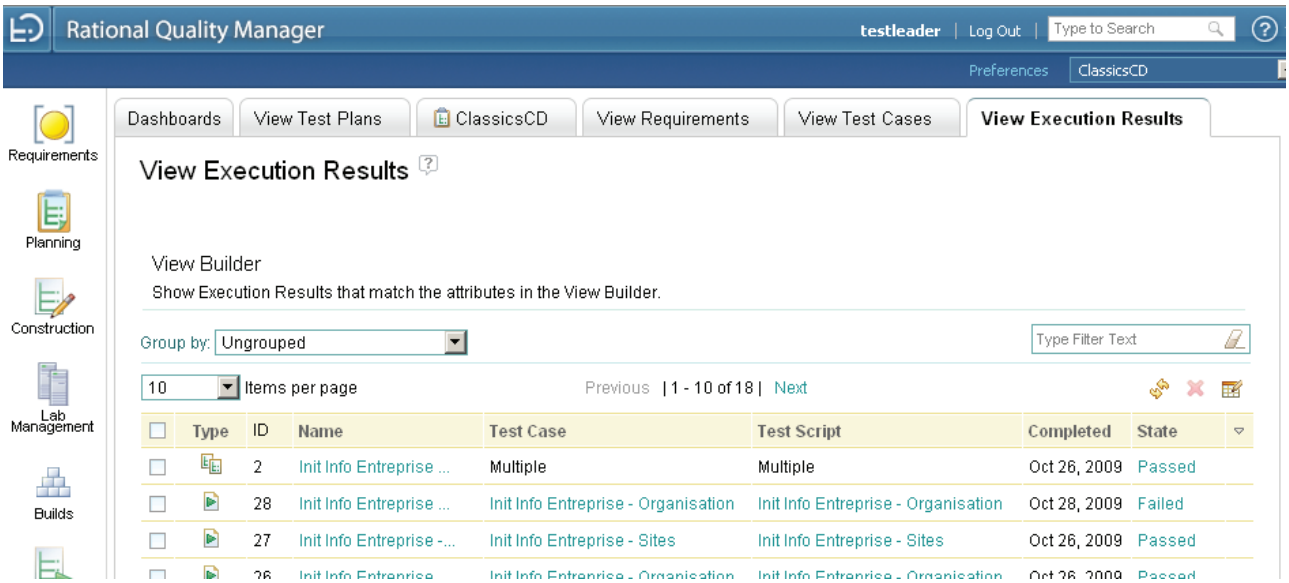
L'Assurance Qualité avec DFSS

Le système

Aujourd'hui, les équipements médicaux qui permettent le diagnostic des maladies sont des systèmes où la composante principale est le logiciel. C'est grâce au logiciel que les médecins sont capables de faire un diagnostic plus rapide et beaucoup plus précis qu'auparavant.

Le logiciel que nous étudions représente une plateforme pour des applications qui permettent de dépister des maladies en se basant sur des résultats d'examens médicaux. Ce logiciel est installé sur une plateforme Linux spécialement configurée. Il propose diverses fonctions comme la récupération de données liées au dossier patient, la récupération des examens effectués sur divers appareils, leur visualisation et leur traitement dans le but d'effectuer un diagnostic, un archivage, une impression et d'autres fonctions spécifiques.

La démarche de développement de logiciel est faite dans un cadre normatif strict (régulations FDA et autres) et, pour minimiser les risques portés par le logiciel, ce développement se fait en utilisant l'approche DMADV[9] (Define Measure Analyse Design Verify) aussi connu sous le nom de DFSS[10] (Design For Six Sigma).



Source: IBM

Figure 12 Résultats d'exécution d'un plan de test

[9]De Feo, Joseph A.; Barnard, William (2005). JURAN Institute's Six Sigma Breakthrough and Beyond - Quality Performance Breakthrough Methods. Tata McGraw-Hill Publishing Company Limited. ISBN 0-07-059881-9

[10]Pour plus d'information sur Six Sigma et DFSS http://en.wikipedia.org/wiki/DFSS

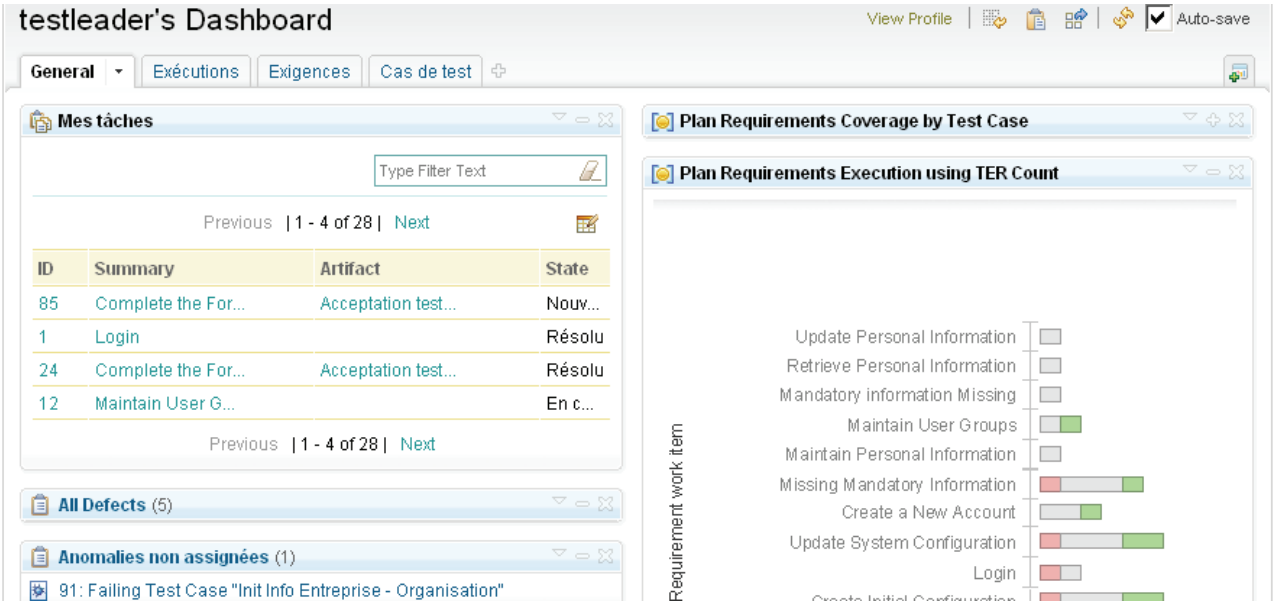


Figure 13 Tableau de bord

Source: IBM

L'organisation de l'équipe

L'équipe projet test composée d'une dizaine d'ingénieurs de développement logiciel, d'un chef de projet/architecte, d'un responsable d'installation de produit, d'un responsable de test, de deux testeurs et d'un responsable de configuration logicielle/fabrication. De plus, chaque ingénieur de développement passe une demi-journée en tant que testeur afin de renforcer l'équipe de test et d'acquérir une vision globale du système en « utilisant » la fonctionnalité du système développée par d'autres ingénieurs.

Mise en place de l'approche DMADV vis-à-vis du test

Phase DMADV : Définir et Mesurer (Define & Mesure)

Afin d'élaborer une architecture de logiciel robuste qui répondra aux exigences imposées sur le système et sur le processus de développement, l'équipe de développement a étudié l'utilisation typique d'un même type de système par ces utilisateurs.

L'équipe a constaté que les médecins travaillent avec 6 types d'examens différents. Ces examens sont générés par l'équipement qui vient principalement de quatre constructeurs majeurs (95% de cas étudiés). De plus, le système doit se comporter de manière stable lors de son utilisation permanente et ininterrompue (sans redémarrage) pendant au moins 185 jours. C'est une mesure qui est essentielle et critique vis-à-vis de la qualité, appelé un CTQ (Critical To Quality) dans la terminologie de Six Sigma.

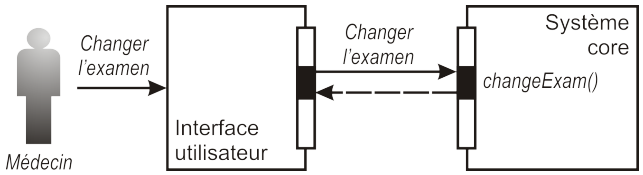
Les exigences sont définies dans des documents qui sont gérés par IBM Rational Requisite Pro. Ceci permet de réaliser la traçabilité entre les exigences et les éléments de la conception, grâce à l'intégration avec Rational Software Modeler. De plus, l'intégration entre RequisitePro, Rational ClearQuest et Rational Quality Manager permet de maintenir les liens de traçabilité entre les exigences, les demandes de changement et les tests sur l'ensemble du cycle de vie de développement. Cette traçabilité est un élément important pour le développement car c'est une contrainte, un autre CTQ, imposé par FDA sur tous les appareils médicaux vendus sur le territoire américain. Tous les artefacts, y compris les exigences sur lesquelles la baseline a été établie, font l'objet d'une gestion de configuration avec Rational ClearCase.

Pour garantir la qualité intrinsèque du logiciel, les testeurs sont appelés à valider les exigences afin d'assurer leur testabilité. De plus, la testabilité du logiciel (autre CTQ) est l'une des exigences vis-à-vis du système.

Le testing sur le cycle de développement de logiciel en deux exemples

Phase DMADV : Analyser (Analyse)

Pour satisfaire l'exigence de testabilité, les ingénieurs en charge de la conception du produit ont décidés de séparer l'interface utilisateur du reste du système en l'encapsulant dans un module séparé. Chaque manipulation avec l'interface utilisateur se résume par un appel vers le proxy d'interface utilisateur dans une partie interne du système (Figure 14).



Source: IBM

Figure 14 Permettre la testabilité par design

Cette approche permet de tester toute les fonctionnalités proposées par le logiciel même sans interface utilisateur.

Pour réaliser le projet de développement dans le temps imparti avec une qualité requise, il a été décidé d'utiliser la politique de prévention proactive des défauts à travers des tests unitaires, de l'analyse du code statique et dynamique automatisée, à l'aide des outils comme Rational PurifyPlus. La mise en place d'une telle politique doit permettre de réduire le temps passé par l'équipe dans le debugging du code (une recherche effectuée par un cabinet d'études constate que, dans les grandes entreprises, 37% du temps d'un développeur est dépensé sur les tâches de débogages). Pour renforcer cette politique, les ingénieurs de développement logiciel doivent tester leur travail avant de le soumettre (checkin) sur leur branche de développement dans le référentiel. Quand l'ingénieur demande la fabrication, l'analyse automatique du code source se déclenche. Puis, toutes les fabrications faites sur la branche d'intégration déclenchent l'exécution de la batterie des tests unitaires et une analyse statique du code. Si cette analyse ou les tests unitaires génèrent des erreurs, la fabrication est vouée à l'échec.

Pour la suite de cet article, seule la fonctionnalité d'impression des examens sera couverte. Mais la même approche est utilisée pour le test de toute autre fonctionnalité proposée par le logiciel.

Pour déterminer la meilleure conception de test, l'équipe de test a dû procéder à une analyse supplémentaire des exigences et de l'utilisation de l'équipement diagnostic. Il a été constaté qu'en moyenne les médecins font entre 12 (examens sophistiqués) et 96 (urgences) examens en 24 heures avec 4 à 10 impressions par examen.

Pour simuler le travail fait sur 185 jours de fonctionnement ininterrompu de logiciel, il faut exécuter 4.262.400 (6 types d'examens x 4 constructeurs x 96 examens x 10 impressions x 185 jours) combinaisons possibles. En procédant à l'analyse de classes d'équivalence, l'équipe réduit à environ 30.000 le nombre de combinaisons à exécuter pour simuler l'impression des examens sur une période de 185 jours. Ce nombre reste élevé pour procéder aux tests manuels et par conséquent l'automatisation de ces tests avec Rational Functional Tester s'imposait. Sous Linux, comme sous Windows, Rational Functional Tester (RFT) propose des capacités de scripting avancées avec Java. Grâce à la conception du système qui permet la testabilité, il est possible d'attaquer le système directement à partir des scripts développés dans RFT et d'appeler la fonctionnalité nécessaire comme si cela était fait à partir de l'interface utilisateur.

Phase DMADV : Concevoir (Design)

Pour simuler le workflow réel du travail du médecin et couvrir le plus de cas possible, la randomisation des flux d'exécution et des données a été utilisée (rendue possible grâce à un des assistants de RFT). Cette randomisation a été journalisée afin de permettre aux ingénieurs de développement de reproduire les étapes jouées par les scripts. Puis, la suite de scripts a été exécutée sur le système. Au début des tests, le système s'arrêtait sans raison apparente après une centaine de cas joués par le script. L'utilisation de Rational PurifyPlus (Figure 15) lors de l'exécution des tests a permis de diagnostiquer la fuite de mémoire, jamais détectée auparavant, qui s'accumulait avec le temps et qui a causé le crash du système. Ainsi quelques défauts liés à la manipulation erronée de la mémoire ont été trouvés. Les problèmes rencontrés avec le système ont été enregistrés dans ClearQuest et ils ont été corrigés par la suite.

Phase DMADV : Vérifier (Verify)

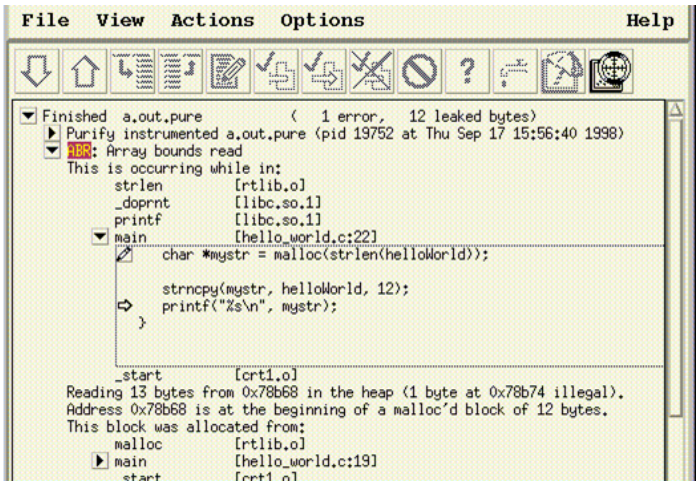
Les tests fonctionnels automatiques ont été inclus dans la suite de test de non régression, exécutée lors de chaque fabrication sur la branche d'intégration. Quand les scripts de test contenaient des choix aléatoires, plus ils étaient joués, plus des cas de test étaient présentés au système et plus le système était testé rigoureusement ; il en résultait par conséquent une meilleure qualité. Cette approche avait permis d'assurer la qualité ***durable*** du système dans le temps.

Résultats

L'équipe de test a été impliquée dès le début du cycle de développement. Ceci a permis :

1. une meilleure qualité des exigences car les testeurs validaient la testabilité des exigences,
2. la prise en compte en amont dans la conception des exigences de testabilité du logiciel,
3. une analyse approfondie des exigences et une meilleure compréhension du système par les testeurs afin de modéliser une utilisation réaliste du système lors de tests,
4. d'assurer la couverture complète des exigences par le test et de répondre aux exigences réglementaires (FDA par exemple).

De plus, le professionnalisme de l'équipe, sa connaissance des utilisateurs, sa maîtrise des techniques de test ont permis d'obtenir un système qui intègre la Qualité par le Design.



Source: IBM

Figure 15 Exemples d'erreur transmis par IBM Rational PurifyPlus

Conclusion

Aujourd'hui, testeur est un métier à part entière dans le processus métier de développement de logiciel. Dans cet article, nous avons vu que le testing ne peut plus rester dans un silo, séparé du reste du cycle de vie de développement. Les outils d'IBM Rational aident à enlever les barrières de communication entre toutes les parties prenantes impliquées dans le développement de logiciel. Grâce à sa plateforme Jazz, IBM Rational permet l'automatisation, le reporting et la collaboration à l'intérieur d'une équipe projet mais aussi au sein de l'entreprise, dans sa globalité et au-delà. Plus de visibilité, plus de maîtrise, plus de capacités à réagir dans le contexte où l'environnement change rapidement, voilà les vrais atouts de la plateforme. Cela permet aux testeurs de rester efficaces et de se concentrer sur leur métier.

Sommes-nous tous prêts à considérer le testing autrement et ne serait-ce pas un nouveau départ qui permettrait au final, d'augmenter considérablement la part de projets réussis, rapportée dans les études annuelles de Standish Group ?

Michel Speranski
Rational Market Manager
IBM France

Normaliser les définitions et les attentes des activités de test

Cette étude a pour objet de définir différents termes et approches en matière de qualité et de réalisation de tests. Les responsables chargés du développement des applications peuvent s'appuyer sur cette étude pour établir un socle de compréhension commun afin qu'à tous les niveaux de l'entreprise les approches appropriées soient appliquées pour des circonstances spécifiques.

Constat

- Chaque entreprise utilise les mêmes termes de test à sa manière, ce qui se traduit par une certaine ambiguïté et un manque de clarté concernant les attentes.
- L'évaluation de la qualité des logiciels ne se résume pas à identifier les défauts ; il s'agit d'un processus prenant en compte différents facteurs tels que la sécurité, l'ergonomie et l'efficacité de développement.
- Une approche orientée sur l'assurance qualité ne doit pas se limiter aux activités de test, elle doit se concentrer sur l'amélioration continue des processus, la gestion et la traçabilité des exigences, ainsi que la gestion des modifications. Elle doit également viser à supprimer la source des défauts.
- Les activités concernant l'évaluation de la qualité doivent s'étendre sur l'ensemble du cycle de vie d'un projet.

Recommandations

- Établir des définitions standard pour les activités de test et la description des défauts.
- Mettre en œuvre diverses activités de test pour garantir la qualité globale des logiciels.
- Favoriser la prévention des défauts grâce à l'assurance qualité plutôt que d'effectuer un contrôle de qualité pur.

ANALYSE

Dans le cadre de ses conversations avec les entreprises informatiques, Gartner a pu constater un manque de cohérence dans la terminologie utilisée pour décrire le processus de qualité global des logiciels et les attentes concernant les différents types d'activités de test. Les entreprises doivent comprendre les différents types de test disponibles et par conséquent savoir quels outils et quelles compétences employer pour réaliser des tests efficaces. En outre, il est crucial qu'un vocabulaire précis et des définitions standard soient utilisés pour les communications et les mesures. Bien que les termes décrivant la plupart des types de tests courants, tels que les tests de charge, de stress et de régression, soient compris par le plus grand nombre, d'autres termes tels que les tests de recette utilisateur peuvent être interprétés de différentes manières. Par conséquent, il est difficile d'appliquer des métriques cohérentes et de répondre avec la même constance aux exigences des utilisateurs. Cette étude dresse une brève liste des diverses activités de test, en donne une courte description et indique leur usage. Les entreprises doivent compléter cette liste par des informations provenant d'autres organismes de définition de normes, notamment l'ISTQB (International Software Testing Qualifications Board), l'ISO (International Organization for Standardization), l'IEEE (Institute of Electrical and Electronics Engineers), ou de leurs fournisseurs d'outils agréés. L'objectif étant de disposer d'attentes réalistes et de définitions cohérentes.

Bien qu'il soit judicieux de réaliser des tests le plus tôt possible dans le cycle de vie, il n'est pas utile de commencer certains tests avant que le code ne soit prêt. La planification des tests commence lors de la conception des applications ou de leur implémentation. Toutefois, les activités de test et de vérification de la qualité ne se limitent pas au test du code des applications ; la vérification de la qualité doit commencer par la validation des spécifications (par exemple, la complétude et la testabilité). En outre, les activités de vérification de la qualité ne dépendent pas uniquement de l'équipe chargée de la réalisation des tests. Les entreprises informatiques ont souvent des attentes irréalistes concernant les résultats de certains tests ou bien elles sous-estiment le travail requis pour qu'ils soient efficaces.

Les entreprises ont tendance à employer des métriques basées sur le nombre de bogues détectés ; ce n'est pas l'objectif du test de logiciels. L'objectif est de vérifier que le système réagit conformément aux exigences et qu'il se comportera de même dans son environnement de production. En s'appuyant sur les meilleures pratiques en la matière, cela permet d'identifier la source du défaut et d'en supprimer la cause, plutôt que de supprimer plusieurs fois le même défaut. C'est le cœur de l'assurance qualité.

Assurance qualité ou contrôle qualité ?

De nombreuses entreprises utilisent indifféremment les termes « assurance qualité » et « contrôle qualité ». L'assurance qualité doit être envisagée comme une approche proactive visant à améliorer la qualité. Il ne s'agit pas de rechercher seulement les défauts mais d'en rechercher la source afin de les supprimer. Le contrôle qualité, quant à lui, est un processus de vérification réactif visant à rechercher les défauts. Il consiste à vérifier et examiner le travail réalisé.

Dans son étude complète sur la vérification de la qualité dans le domaine de la production, W. Edwards Deming déclare : « L'inspection dans le but de rechercher les mauvais éléments et de les supprimer est trop tardive, inefficace et coûteuse. La qualité provient non de l'inspection mais de l'amélioration du processus. » Cela est avéré pour les logiciels ; la recherche constante de défauts une fois que le code est créé est un processus de contrôle qui intervient trop tard. Les entreprises ont besoin de l'assurance qualité et du contrôle qualité mais l'assurance qualité est plus efficace (voir les exemples d'activités de chaque processus dans le tableau 1) :

Tableau 1. Couverture de la méthode par phase

Assurance qualité	Contrôle qualité
Audit de qualité	Procédure pas à pas
Définition du processus	Test
Sélection des outils	Inspections
Formation	Vérification des points de contrôle

Source : Gartner (novembre 2008)

Il existe différentes manières d'utiliser les inspections, par exemple, dans une approche proactive, comme indiqué dans la section « Qualité du code » plus bas. Dans cette approche, des outils d'évaluation ou des logiciels sont utilisés avant la phase de fabrication (build) de l'application (et éventuellement au cours de la phase de conception et de codage) pour améliorer en amont la conception et éviter que les mêmes erreurs se reproduisent. Une entreprise favorisant l'assurance qualité peut également se rendre compte que la qualité ne relève pas uniquement de la responsabilité de l'équipe chargée des tests. L'assurance qualité repose sur la contribution et la participation d'autres intervenants tels que l'équipe chargée de l'infrastructure et de l'exploitation, l'équipe de développement, les utilisateurs et les analystes des processus métiers.

Types de tests

Test des exigences

- **Complétude :** toutes les exigences fonctionnelles et non fonctionnelles (telles que les performances, l'environnement et l'intégration) sont-elles prises en compte ?
- **Points d'évaluation :** les modalités visant à vérifier que les exigences sont respectées sont-elles claires ? Est-il possible de tester les exigences ?
- **Alignement :** le plan de test respecte-t-il les exigences en matière de planification, d'objectifs et de priorités ?

Les entreprises mettant en œuvre les meilleures pratiques impliquent l'équipe chargée de l'assurance qualité dès le début du projet. Une fois que les exigences initiales sont réunies pour chaque point d'élaboration, l'équipe chargée de l'assurance qualité doit disposer d'un point de contrôle (avant les phases de conception et de mise en œuvre) pour vérifier qu'aucune exigence ne manque et que toutes les exigences peuvent être testées, ainsi que pour planifier les test en laboratoire. Si l'équipe de test ne peut pas déterminer de quelle manière elle vérifiera que le code fonctionne correctement, les exigences ne peuvent pas être testées et il est fort probable que la personne chargée de la réalisation des tests ne pourra que supposer.

Tests de performance

- **Performance :** ces tests permettent de savoir où les goulots d'étranglement se produisent, ces tests sont réalisés par des développeurs, souvent à l'aide d'outils dits outils de profilage.
- **Charge :** ces tests visent à analyser les performances d'une application à mesure que la charge ou que le nombre d'utilisateurs ou de messages augmente.
- **Stress :** en s'appuyant sur les tests de charge, les tests de stress analysent ce qui se produit lorsque les ressources du système s'épuisent.

Normaliser les définitions et les attentes des activités de test

L'objectif des tests de performance est de rechercher les causes sous-jacentes des mauvaises performances d'une application, ce qui implique un examen approfondi des performances de l'application au niveau de l'exécution du code. Ces tests ont également pour objectif de vérifier que les algorithmes utilisés et les implémentations dans le code sont adaptés aux besoins de l'application. Les tests de performance doivent intervenir tôt dans le cycle de vie de l'application, lorsque les modifications structurelles apportées aux algorithmes auront le moins d'effet sur les autres composants du code.

Les tests de charge et de stress fournissent une vue de l'application à un plus haut niveau. Ils ont pour objet d'examiner le comportement de l'application à mesure que le nombre d'utilisateurs et la complexité ou la composition de la charge utile augmentent. Les utilisateurs sont générés de façon artificielle à l'aide d'ordinateurs de contrôle de la charge. L'objectif de ces tests est de garantir la stabilité de l'application et de vérifier que les performances du système répondent aux attentes des utilisateurs même sous charge appliquée. Bien qu'il soit important de savoir que le système peut fonctionner correctement sous la charge prévue, il est également crucial que les équipes réalisant les tests de charge et de stress fournissent à l'équipe d'exploitation les informations qui lui permettront d'assurer le contrôle et la maintenance du système. Le fonctionnement de l'application dépend-il du processeur ou de la mémoire ? Qu'est ce qui indique que le système est sur le point de tomber en panne ? Ces informations seront utiles pour le support à long terme du système à mesure que de nouvelles applications sont disponibles en ligne ou que l'activité utilisateur augmente au-delà des volumes initialement prévus. Enfin, il est important de connaître la cause des pannes sous la charge, leur effet (par exemple, la perte de données) et leur impact éventuel sur d'autres systèmes dépendants. Cela s'avérera de plus en plus important du fait que les entreprises adoptent de plus en plus aujourd'hui une architecture orientée services et qu'elles partagent des chaînes de service, internes et externes, complexes.

Tests fonctionnels

- **Boîte blanche :** ces tests impliquent une compréhension explicite de l'implémentation ; les tests unitaires en sont un bon exemple.
- **Boîte noire :** l'implémentation sous-jacente n'est pas requise pour ces tests. Ces tests se concentrent sur l'interface ou les spécifications externes ; la majorité des outils de test fonctionnent de cette manière.
- **Tests unitaires :** ces tests se concentrent sur des unités, des modules ou des composants spécifiques et ils indiquent si des règles peuvent en être tirer.
- **Automatisation :** un logiciel est utilisé pour automatiser l'exécution des tests, la consignation des résultats et la gestion des scénarios de test.
- **Tests de régression :** ces tests répétitifs et automatisés permettent d'éviter que les modifications apportées au système introduisent des défauts.
- **Tests d'intégration :** ces tests sont effectués entre les différentes unités pour vérifier que les interactions fonctionnent correctement.
- **Tests du système :** ces tests sont effectués sur le système entier dans une configuration de production.

Les tests fonctionnels sont au cœur de la plupart des activités de test. Ils peuvent être effectués manuellement ou automatiquement sur le code ou sur l'API (l'interface de programmation d'une application). Ils permettent de savoir si le code répond aux attentes et de garantir la stabilité à long terme de l'application. La plupart des tests fonctionnels sont effectués manuellement. L'utilisation des cadres de tests unitaires et d'une approche davantage axée sur les tests (développement piloté par les tests) est devenue une pratique de plus en plus courante pour détecter les défauts le plus tôt possible. L'automatisation réussie de tests « boîte noire » continue d'avoir un impact sur les attentes de retour sur investissement de nombreuses entreprises. Toutefois, avec l'utilisation accrue de logiciels packagés et de solutions hébergées, pouvoir réaliser des tests sans accéder au code va devenir une compétence fondamentale. Les tests fonctionnels doivent permettre de vérifier que le comportement d'une application est satisfaisant et que les principaux scénarios s'exécutent correctement lors des tests « boîte noire ». Pour le groupe informatique les tests de régression sont critiques, car la crédibilité de l'entreprise est en jeu. Quels que soient les avantages que présentent les nouvelles fonctionnalités, il n'est pas acceptable de briser les fonctionnalités déjà établies ; l'avantage principal des outils d'automatisation de tests est qu'ils préservent l'intégrité de ces fonctions.

Étant donné que les outils s'appuient de plus en plus sur un modèle plus centré sur les données et que le développement s'oriente vers un format plus adapté aux architectures orientées services et à l'exploitation des métadonnées, les outils d'automatisation offriront un retour sur investissement plus rapide. En outre, ils conviennent mieux à la création de tests en amont de l'implémentation.

Tests de sécurité

- **Modèles de menace :** ces modèles fournissent une définition des brèches potentielles de la sécurité (concernant les points d'entrées, les privilèges d'accès et la disponibilité des ressources). Ils permettent en outre aux entreprises de constituer une arborescence résistant aux attaques et de mettre en œuvre des mesures de contrôle de la surface d'exposition de l'application.
- **Analyse statique :** il s'agit d'une analyse du code source qui recherche dans la structure de l'application des erreurs courantes de codage pouvant créer des points de vulnérabilité.
- A**nalyse dynamique :** cette analyse teste l'application et recherche des attaques tels que l'injection de code SQL. Les techniques dynamiques impliquées ont tendance à s'appuyer sur des méthodologies de « data fuzzing » (injection de données aléatoires).

Les tests de sécurité sont devenus un élément clé du cycle de vie du développement d'une application. Les premiers outils se concentraient sur les méthodes d'analyse statique, ce qui constituait déjà un progrès, mais ils se conformaient au concept obsolète selon lequel il fallait tester la qualité directement dans une application. Pour suivre l'évolution du marché, les entreprises et les fournisseurs ont amélioré les pratiques utilisées de façon à adopter une approche orientée sur la prévention des attaques.

Tests de qualité du code

- **Analyse statique :** ces outils semblables à un compilateur, examinent le code, en tenant compte de la complexité et du respect des règles de codage, et recherchent les utilisations incorrectes des bibliothèques standards et tout autre problème lié au code.
- **Évaluation par les pairs :** ces outils fournissent un second regard sur le code. Il s'agit d'une méthode éprouvée permettant de détecter et de réduire les erreurs plus tôt dans le cycle de vie sans que cela donne lieu à un processus formel.

L'évaluation de la qualité du logiciel en amont et l'identification des erreurs, à l'aide d'outils et de techniques appropriées, afin d'éviter qu'elles aient un impact sur la fabrication, sont effectuées lors de l'analyse statique et de l'évaluation du code par des pairs. Appliquées sous forme de paire, ces techniques vérifient le logiciel sur la base des meilleures pratiques en la matière et constituent le meilleur moyen de supprimer les défauts difficiles à trouver. Ces défauts n'apparaissent pas dans la fonctionnalité générale de l'application mais ils impliquent des problèmes pouvant entraîner une perte de stabilité ou nuire à la réutilisation de l'application et ainsi augmenter les coûts de maintenance.

Autres éléments

- **Tests de recette utilisateur :** ces tests doivent être concentrés sur la validation du système livré par l'utilisateur (logiciel, supports de formations et documents connexes). L'objectif de ces tests n'est pas de rechercher des bogues mais de fournir une validation finale du système livrée ; effectuée par un tiers, cette phase de tests est appelée validation et vérification indépendante.
- **Fuzzing :** ces tests sont utilisés pour plusieurs objectifs mais ils sont surtout concentrés sur la recherche des vulnérabilités de sécurité. Cette technique repose sur l'injection aléatoire de données dans l'application au cours des tests fonctionnels.
- **Facilité d'utilisation :** ces tests se concentrent sur l'interface utilisateur et sur les interactions entre l'utilisateur et l'application. Pour ces tests, l'utilisateur, sous observation, doit naviguer dans l'application en suivant divers scénarios.
- **Globalisation/Localisation :** ces tests ont pour but de vérifier que l'application peut prendre en charge plusieurs langues pour l'entrée d'informations (globalisation) et l'affichage (localisation).
- **Accessibilité :** ces tests ont pour but de vérifier que l'application est adaptée aux utilisateurs présentant un handicap.
- **Documentation :** ces tests ont pour but de vérifier que les documents de support (manuels, aide en ligne et documentation de formation) sont corrects. Ils sont effectués en suivant une procédure pas à pas.

Normaliser les définitions et les attentes des activités de test

La qualité d'une application ne se mesure pas seulement à ses fonctions et ses performances ; elle doit également répondre à certaines règles. Un logiciel de qualité ne fonctionnera pas seulement efficacement mais il permettra aussi aux utilisateurs de travailler efficacement. Les logiciels doivent répondre aux besoins de tous les utilisateurs et étant donné que de plus en plus d'utilisateurs accèdent aux applications à partir d'appareils mobiles et de divers navigateurs partout dans le monde, l'équipe chargée des tests doit avoir les compétences requises et disposer des outils appropriés pour automatiser ces tests.

Outils de test

La plupart des types de test nécessitent des outils et des environnements spécifiques permettant de réduire les opérations nécessaires ; cependant, cela ne signifie pas que tous les tests peuvent être « automatisés » et la création de tests ne représente que la moitié du travail. Le processus d'automatisation ainsi que la gestion des changements et de l'intégration à travers de l'équipe de développement sont au cœur de l'automatisation. Les principaux outils sur le marché permettent de générer des cas de test à partir des cas d'utilisation et de gérer et de coordonner les modifications entre eux.

Source: Gartner Référence G00162958,
Thomas E. Murphy,
25 novembre 2008