Testeur Certifié

Syllabus Niveau Avancé
Analyste de Test

Version 2012

International Software Testing Qualifications Board

Groupe de travail Analyste de Test avancé: Judy McKay (Responsable), Mike Smith, Erik Van Veenendaal; 2010-2012.

Traduction Française :

- Réalisée par le Comité Français des Tests Logiciels
- Les remarques et corrections à apporter sur la traduction sont à transmettre par mail à traductions@cftl.fr

Historique des modifications

Version	Date	Remarques
ISEB v1.1	04SEP01	ISEB Practitioner Syllabus
ISTQB 1.2E	SEP03	ISTQB Advanced Level Syllabus from EOQ-SG
V2007	12OCT07	Certified Tester Advanced Level syllabus version 2007
V2007FR	31DEC08	Testeur Certifié Niveau Avancé, version Française
D100626	26JUN10	Incorporation des changements acceptés en 2009, division des chapitres pour les différents modules
D101227	27DEC10	Acceptation des changements de format et des corrections n'ayant pas d'impact sur la signification des phrases.
D2011	31OCT11	Division du syllabus, reformulation des OAs et modifications de texte pour correspondre aux OAs. Ajout des objectifs métier.
Alpha 2012	09Fèv12	Incorporation de tous les commentaires reçus des comités nationaux sur la version d'octobre.
Beta 2012	26Mar12	Incorporation des commentaires reçus à temps des comités nationaux sur la version Alpha.
Beta 2012	07APR12	Version Beta soumise à l'AG
Beta 2012	08JUN12	Version éditée distribuée aux comités nationaux
Beta 2012	27JUN12	Incorporation des commentaires des groupes de travail EWG et Glossaire
RC 2012	15AUG12	Livraison de la version candidate – inclusion des commentaires finaux des comités nationaux
GA 2012	19OCT12	Mises à jour finales et nettoyage pour la livraison à l'assemblée générale

Table des matières

Remerciements

Ce document a été produit par une équipe issue du sous-groupe de travail Niveau Avancé de l'International Software Testing Qualifications Board – Analyste de Test Avancé: Judy McKay (Responsable), Mike Smith, Erik van Veenendaal.

L'équipe remercie l'équipe de revue ainsi que tous les Comités Nationaux pour leurs suggestions et entrées.

Au moment de la finalisation du Syllabus Niveau Avancé le groupe de travail Niveau Avancé était constitué des membres suivants (par ordre alphabétique):

Graham Bath, Rex Black, Maria Clara Choucair, Debra Friedenberg, Bernard Homès (Vice Chair), Paul Jorgensen, Judy McKay, Jamie Mitchell, Thomas Mueller, Klaus Olsen, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Jan Sabak, Hans Schaefer, Mike Smith (Chair), Geoff Thompson, Erik van Veenendaal, Tsuyoshi Yumoto.

Les personnes suivantes ont participé aux relectures, revues et choix pour ce document:

Graham Bath, Arne Becher, Rex Black, Piet de Roo, Frans Dijkman, Mats Grindal, Kobi Halperin, Bernard Homès, Maria Jönsson, Junfei Ma, Eli Margolin, Rik Marselis, Don Mills, Gary Mogyorodi, Stefan Mohacsi, Reto Mueller, Thomas Mueller, Ingvar Nordstrom, Tal Pe'er, Raluca Madalina Popescu, Stuart Reid, Jan Sabak, Hans Schaefer, Marco Sogliani, Yaron Tsubery, Hans Weiberg, Paul Weymouth, Chris van Bael, Jurian van der Laar, Stephanie van Dijk, Erik van Veenendaal, Wenqiang Zheng, Debi Zylbermann.

Ce document a été officiellement approuvé pour publication par l'Assemblée Générale de l'ISTQB® le 19 octobre 2012.

0. Introduction à ce Syllabus

0.1 Objectif de ce Document

Ce syllabus forme la base de la Qualification Internationale en Test de Logiciels au Niveau Avancé pour l'Analyste de Test. L'ISTQB® fournit ce syllabus comme suit :

1. Aux Comités Nationaux, pour traduction dans leur langue et pour l'accréditation des fournisseurs de formation. Les Comités Nationaux peuvent adapter le syllabus aux particularités de leur langage et modifier les références pour les adapter à leurs publications locales.
2. Aux Comités d'Examens, pour en dériver des questions d'examen adaptées aux objectifs d'apprentissage de chaque module dans la langue locale.
3. Aux fournisseurs de formation, pour concevoir les cours et déterminer les méthodes de formation appropriées.
4. Aux candidats à la certification, pour préparer l'examen (dans le cadre d'une formation ou indépendamment).
5. A la communauté internationale du logiciel et de l'ingénierie des systèmes, pour faire progresser la profession de testeur de logiciels et de systèmes, et comme base pour des livres et articles.

L'ISTQB® peut autoriser d'autres entités à utiliser ce syllabus pour d'autres objectifs à condition qu'elles demandent et obtiennent une autorisation écrite préalable.

0.2 Vue générale

Le Niveau Avancé est composé de trois syllabi séparés:

- Test Manager
- Analyste de Test
- Analyste Technique de Test

Le document Niveau Avancé Vue Générale [ISTQB_AL_OVIEW] contient les informations suivantes:

- Gains métier pour chaque syllabus
- Résumé pour chaque syllabus
- Relations entre les syllabi
- Description des niveaux cognitifs (Niveaux K)
- Annexes

0.3 Objectifs d'Apprentissage Examinables

Les objectifs d'apprentissage correspondent aux gains métier et sont utilisés pour créer l'examen de passage de la certification Test Manager avancé. Toutes les parties de ce syllabus sont examinables au niveau K1, c.à.d. que le candidat devra reconnaître, se souvenir et mémoriser un terme ou un concept. Les objectifs d'apprentissage significatifs aux niveaux K2, K3 et K4 sont fournis au début de chaque chapitre.

1. Processus de Test - 300 mn.

Mots clé

cas de test concret, critère de sortie, cas de test de haut niveau, cas de test logique, cas de test de bas niveau, contrôle du test, conception des tests, exécution des tests, implémentation des tests, gestion des tests

Objectifs d'Apprentissage pour Processus de Test

1.2 Le Test dans le Cycle de Vie de Développement Logiciel

AT-1.2.1 (K2) Expliquer comment et pourquoi le timing et le niveau d'implication d'un Analyste de Test varient selon les différents modèles de cycles de vie

1.3 Pilotage, Gestion et Contrôle des tests

AT-1.3.1 (K2) Résumer les activités réalisées par l'Analyste de Test en soutien à la gestion et au contrôle du test

1.4 Analyse des tests

AT-1.4.1 (K4) Analyser un scénario donné, incluant la description d'un projet et d'un modèle de cycle de vie, pour déterminer les tâches dédiées à l'Analyste de Test lors des phases d'analyse et conception

1.5 Conception des tests

AT-1.5.1 (K2) Expliquer pourquoi les conditions de test doivent être comprises par toutes les parties prenantes

AT-1.5.2 (K4) Analyser le scénario d'un projet pour déterminer l'usage le plus adapté de cas de test de bas niveau (concrets) et de haut niveau (logiques)

1.6 Implémentation des tests

AT-1.6.1 (K2) Décrire les critères de sortie typiques pour l'analyse et la conception des tests et expliquer comment la satisfaction à ces critères affectera l'effort d'implémentation des tests.

1.7 Exécution des tests

AT-1.7.1 (K3) Pour un scénario donné, déterminer les étapes et les précautions à prendre lors de l'exécution des tests

1.8 Évaluation des Critères de Sortie et Reporting

AT-1.8.1 (K2) Expliquer pourquoi des informations précises sur le statut d'exécution des tests sont importantes

1.9 Activités de Clôture des tests

AT-1.9.1 (K2) Fournir des exemples de livrables devant être produits par l'Analyste de Test lors des activités de clôture de tests

1.1 Introduction

Dans le syllabus ISTQB® Niveau Fondation, le processus fondamental de test comprend les activités suivantes :

- Planification et contrôle
- Analyse et conception
- Implémentation et exécution
- Évaluation des critères de sortie et information
- Activités de clôture des tests

Pour les syllabi du Niveau Avancé, certaines de ces activités sont abordées séparément, afin de fournir des précisions complémentaires et une optimisation des processus, mieux intégrés avec le cycle de vie de développement logiciel, et pour augmenter l'efficacité du pilotage et contrôle des tests. Les activités sont désormais les suivantes:

- Gestion, pilotage et contrôle
- Analyse
- Conception
- Implémentation
- Exécution
- Évaluation des critères de sortie et information
- Activités de clôture des tests

Ces activité peuvent être mises en œuvre de façon séquentielle ou, pour certaines, être mise en œuvre en parallèle, p. ex., la conception peut se faire en parallèle de l'implémentation (p. ex., test exploratoire). Déterminer les bons tests et cas de test, les concevoir et les exécuter sont les principales activités de l'Analyste de Test. Même s'il est important de comprendre les autres activités du processus de test, la majeure partie du travail de l'Analyste de Test est faite lors des activités d'analyse, de conception, d'implémentation et d'exécution dans le projet de test.

Les testeurs avancés devront faire face à un nombre important de challenges lors de la mise en œuvre des différents aspects du test décrits dans ce syllabus, dans le contexte de leurs propres organisations, équipes et tâches. Les différents cycles de vie de développement logiciel, de même que le type de système testé, doivent être pris en compte car ils peuvent influencer l'approche de test.

1.2 Le Test dans le Cycle de Vie de Développement Logiciel

Il est nécessaire de considérer et de définir dans la stratégie de test l'approche de test qui sera appliquée au cycle de vie sur le long terme. Le moment auquel intervient l'Analyste de Test est différent d'un cycle de vie à un autre et son degré d'implication, le temps passé, l'information disponible et les attentes varient également beaucoup. Comme les processus de test ne sont pas isolés, l'Analyste de Test doit connaître les différentes sources d'information concernant les autres domaines comme:

- L'ingénierie et la gestion des exigences – les revues d'exigences
- La gestion de projet – entrée pour l'organisation du planning
- La gestion de configuration et du changement – test de vérification des livraisons, contrôle de version
- Le développement logiciel – anticipation de ce qui arrive à des moments précis
- La maintenance logicielle – gestion des défauts, temps de redressement (c.à.d. le temps écoulé entre la découverte et la résolution d'un défaut)
- Le support technique – documentation précise des solutions de contournement

- La production de la documentation technique (p. ex., les spécifications de conception de bases de données) – les éléments en entrée pour ces documents de même que la revue technique des documents

Les activités de test doivent être cohérentes avec le cycle de vie de développement logiciel choisi, qu'il soit séquentiel, itératif, ou incrémental. Par exemple, dans le modèle séquentiel en V, le processus de test fondamental de l'ISTQB® pourraient s'organiser de la façon suivante:

- La gestion des tests système se déroule en même temps que la gestion de projet, et le contrôle du test continue jusqu'à ce que l'exécution des tests système et la clôture soient terminés.
- L'analyse et la conception des tests système se déroulent en même temps que la spécification des exigences, la conception du système et de l'architecture (à haut niveau), et la conception des composants (de bas niveau).
- L'implémentation de l'environnement de test système (p. ex., environnements d'exécution, bancs de test) peut commencer lors de la conception du système, même si elle sera essentiellement organisée en parallèle du codage et du test de composant, avec le travail d'implémentation des tests système, se prolongeant souvent jusqu'aux derniers jours avant le démarrage de l'exécution des tests système.
- L'exécution des tests système commence lorsque les critères d'entrée pour le test système sont tous satisfaits (ou écartés), ce qui signifie généralement qu'au moins le test de composant et souvent aussi les tests d'intégration de composant sont terminés. L'exécution des tests système se poursuit jusqu'à ce que les critères de sortie du test système soient satisfaits
- L'évaluation des critères de sortie et la communication des résultats du test système se font tout au long de l'exécution des tests système, généralement avec une fréquence et un degré d'urgence plus élevé à l'approche de la fin du projet.
- Les activités de clôture des tests système se produisent une fois que les critères de sortie des tests système sont satisfaits et que l'exécution des tests système est terminée, même si elles peuvent parfois être repoussées jusqu'à ce que les tests d'acceptation et toutes les activités du projet soient terminés.

Des modèles itératifs et incrémentaux peuvent ne pas suivre le même ordre dans l'enchaînement des tâches et peuvent en exclure certaines. Par exemple, un modèle itératif peut utiliser à chaque itération un sous-ensemble des processus de test standard. L'analyse et la conception, l'implémentation et l'exécution, l'évaluation et le reporting peuvent être faits pour chaque itération, alors que la gestion est organisée au début du projet et que le reporting de clôture est fait à la fin. Dans un projet Agile, il est fréquent d'utiliser un processus moins formel et plus proche des relations qui facilitent les changements dans le projet. Comme Agile est un processus "allégé" il y a moins de documentation de test, mais des méthodes de communication rapide comme les "stand up meetings" journaliers (appelé "stand up" car ils sont très rapide, généralement 10-15 minutes, de façon à ce que personne de s'assoit et à ce que tout le monde reste impliqué).

Les projets Agile, parmi tous les modèles de cycle de vie, demandent l'implication le plus tôt possible de l'Analyste de Test. L'Analyste de Test devrait s'attendre à être impliqué dès le début du projet, en travaillant avec les développeurs au moment où ils font l'architecture et la conception initiales. Les revues peuvent ne pas être formalisées mais sont permanentes au cours de l'évolution du logiciel. Une implication tout au long du projet est nécessaire et l'Analyste de Test doit être disponible pour l'équipe. Grâce à cette immersion, les membres d'une équipe Agile sont, en général, affectés à un unique projet et entièrement impliqués dans tous les aspects du projet.

Les modèles itératifs/incrémentaux vont de l'approche Agile, où le changement est prévu avec l'évolution du logiciel, à des modèles de développement itératifs/incrémentaux existant au sein d'un modèle en V (parfois appelés itératifs intégrés). Dans le cas d'un modèle itératif intégré, l'Analyste de Test devrait s'attendre à être impliqué dans le planning standard et dans les aspects liés à la

conception, mais il aura ensuite un rôle plus interactif lorsque le logiciel sera développé, testé, modifié et déployé.

Quel que soit le cycle de vie de développement logiciel utilisé, l'Analyste de Test doit comprendre les attentes relatives à son implication et au temps qu'il y consacrera. Beaucoup de modèles hybrides sont utilisés, tels que le modèle itératif inclus dans un modèle en V comme indiqué ci-dessus. L'Analyste de Test doit souvent définir le rôle le plus efficace et s'orienter vers lui pour intervenir au meilleur moment, plutôt que de rester dépendant d'un modèle particulier.

1.3 Gestion, Pilotage et Contrôle des tests

Cette section aborde les processus de gestion, pilotage et contrôle des tests.

1.3.1 Gestion des tests

La gestion des tests se produit principalement au commencement de l'effort de test et demande l'identification et la gestion de toutes les activités et ressources requises pour satisfaire à la mission et aux objectifs identifiés dans la stratégie de test. Pendant la gestion des tests, il est important que l'Analyste de Test travaille avec le Test Manager, pour aborder et prévoir les points suivants:

- S'assurer que les plans de test ne soient pas réduits à du test fonctionnel. Tous les types de test doivent être considérés dans le plan de test et planifiés en conséquence. Par exemple, en plus du test fonctionnel, l'Analyste de Test peut être responsable des tests d'utilisabilité. Ce type de test doit aussi être traité dans le plan de test.
- Revoir les estimations de test avec le Test Manager et assurer le temps et le budget adéquats à la fourniture et à la validation de l'environnement de test.
- Prévoir des tests de configuration. Si plusieurs types de processeurs, de systèmes d'exploitation, de machines virtuelles, de navigateurs et différents périphériques peuvent être combinés en de nombreuses configurations, il faut prévoir d'appliquer les techniques de test qui apporteront la meilleure couverture de ces combinaisons.
- Prévoir de tester la documentation. Le logiciel est fourni aux utilisateurs avec de la documentation. Celle-ci doit être précise pour être utile. L'Analyste de Test doit allouer du temps à la vérification de la documentation et peut être amené à travailler avec des rédacteurs techniques pour préparer les données à utiliser pour des copies d'écran et clips vidéo.
- Prévoir de tester les procédures d'installation. Les procédures d'installation, de même que les procédures de sauvegarde et de restauration, doivent être suffisamment testées. Ces procédures peuvent être plus critiques que le logiciel qui, s'il ne peut pas être installé, ne sera pas utilisé. Cela peut être difficile à organiser puisque l'Analyste de Test réalise souvent les premiers tests sur un système ayant été préconfiguré sans que les processus d'installation ne soient en place.
- Prévoir le test de façon cohérente avec le cycle de vie logiciel. Une exécution séquentielle des tâches ne convient pas à la plupart des plannings. De nombreuses tâches ont souvent besoin d'être réalisées (au moins en partie) de façon concurrente. L'Analyste de Test doit connaître le cycle de vie sélectionné et les attentes relatives à son implication durant la conception, le développement et l'implémentation du logiciel. Cela inclut aussi l'attribution de temps aux tests de régression et de confirmation.
- Allouer le temps nécessaire à l'identification et à l'analyse des risques avec l'équipe fonctionnelle transverse. Même si en général, il n'est pas responsable de l'organisation de sessions de gestion des risques, l'Analyste de Test devrait s'attendre à y être fortement impliqué.

Des relations complexes peuvent exister entre les bases de test, les conditions de test et les cas de test telles que des relations n-n peuvent exister entre ces livrables. Ces relations doivent être

comprises pour permettre une mise en œuvre efficace de la gestion et du contrôle des tests. L'Analyste de Test est en général la meilleure personne pour définir ces relations et séparer autant que possible les dépendances.

1.3.2 Pilotage et Contrôle des tests

Même si l'activité de pilotage et de contrôle des tests appartient en général au Test Manager, l'Analyste de Test contribue aux mesures permettant le contrôle.

Un certain nombre de données quantitatives devraient être collectées durant le cycle de vie de développement logiciel (p. ex., le pourcentage d'activités de gestion des tests terminées, le pourcentage de couverture atteint, le nombre de cas de test passés avec succès ou en échec). Dans tous les cas, une référence (c.à.d., une référence standard) doit être définie et ensuite le progrès sera mesuré par rapport à cette référence. Alors que le Test Manager sera concerné par la compilation et le reporting de l'information mesurée pour les métriques, l'Analyste de Test collecte l'information pour chaque métrique. Chaque cas de test finalisé, chaque défaut rapporté par écrit, chaque jalon atteint, apparaîtront dans les métriques globales du projet. L'information saisie dans les différents outils de suivi devra être aussi précise que possible afin que les métriques reflètent la réalité.

Des métriques précises permettent aux responsables de gérer un projet (piloter) et d'amorcer les changements nécessaires (contrôle). Par exemple, un grand nombre de défauts rapportés pour un domaine du logiciel peut signifier qu'un effort de test supplémentaire est requis sur ce domaine. L'information relative à la couverture des exigences et des risques (traçabilité) peut être utilisée pour prioriser le travail restant et allouer des ressources. L'information relative à la cause racine est utilisée pour identifier des domaines d'amélioration dans les processus. Si les données enregistrées sont précises, le projet peut être contrôlé et une information précise sur le statut peut être diffusée aux parties prenantes. Les projets à venir peuvent être planifiés plus efficacement lorsque les données issues des projets précédents sont prises en compte. Les utilisations possibles de données précises sont extrêmement nombreuses. Il appartient à l'Analyste de Test de s'assurer que les données sont précises, délivrées à temps et objectives.

1.4 Analyse des tests

Pendant la gestion des tests, le périmètre du projet de test est défini. L'Analyste de Test utilise cette définition du périmètre pour:

- Analyser les bases de test
- Identifier les conditions de test

Pour permettre à l'Analyste de Test de mener efficacement l'analyse des tests, les critères d'entrée suivants devraient être satisfaits:

- Il existe un document décrivant l'objet de test qui pourra servir de bases de test
- Ce document a fait l'objet d'une revue avec des résultats satisfaisants et a été mis à jour comme cela est nécessaire, à la suite de la revue
- Un budget et un planning raisonnables sont définis pour accomplir le travail de test restant pour cet objet de test

Les conditions de tests sont identifiées en général par l'analyse des bases de test et des objectifs de test. Dans certaines situations, où la documentation peut être ancienne ou inexistante, les conditions de test peuvent être identifiées en parlant aux parties prenantes concernées (p. ex., lors de réunions de travail ou lors du « sprint planning »). Ces conditions sont ensuite utilisées pour déterminer quoi tester, en utilisant les techniques de conception des tests identifiées dans la stratégie de test et/ou dans le plan de test.

Même si les conditions de test sont en général spécifiques à l'élément à tester, l'Analyste de Test doit faire les considérations suivantes :

- Il est en général recommandé de définir des conditions de test à différents niveaux de détail. Initialement, des conditions de haut-niveau sont définies pour définir les principales cibles du test, telles que "fonctionnalités de l'écran x". Par la suite, des conditions plus détaillées sont identifiées comme base de cas de test spécifiques, telles que "l'écran x rejette un numéro de compte dont la longueur est d'un caractère inférieure à la longueur correcte". Utiliser ce type d'approche hiérarchique pour définir des conditions de test peut aider à assurer que la couverture est suffisante pour des éléments de haut niveau.
- Si des risques produit ont été définis, alors les conditions de test nécessaires pour traiter chaque risque produit devront être identifiées et reliées au risque.

A la fin des activités d'analyse des tests l'Analyste de Test doit savoir quels tests spécifiques doivent être conçus pour satisfaire aux besoins des différents domaines du projet de test.

1.5 Conception des tests

Toujours en cohérence avec le périmètre défini lors de la gestion des tests, le processus de test se poursuit avec la conception, par l'Analyste de Test, des tests qui seront développés et exécutés. Le processus de conception des tests comprend les activités suivantes:

- Déterminer pour quel domaine de test des cas de test de bas niveau (concrets) ou de haut niveau (logiques) seront le plus adaptés
- Déterminer la ou les technique(s) de conception des tests qui permettront la couverture de test nécessaire
- Créer les cas de test qui solliciteront les conditions de test identifiées

Les critères de priorisation identifiés lors de l'analyse de risque et lors de la gestion des tests doivent être appliqués tout au long du processus, de l'activité d'analyse et de conception à l'activité d'implémentation et d'exécution.

Selon les types de test à concevoir, l'un des critères d'entrée pour la conception des tests peut être la disponibilité des outils qui seront utilisés pendant le travail de conception.

Lors de la conception des tests, il est important de se souvenir des points suivants :

- Certains éléments de test seront mieux couverts en définissant simplement des conditions de test plutôt qu'en allant plus loin dans la définition de scripts de test. Dans ce cas, les conditions de test doivent être définies pour être utilisées comme un guide lors de l'exécution de tests sans script.
- Les critères de passage/échec doivent être clairement identifiés.
- Les tests doivent être conçus de façon à être compréhensibles par d'autres testeurs, et pas seulement par leur auteur. Si l'auteur n'est pas la personne qui exécute le test, d'autres testeurs auront besoin de lire et de comprendre les tests spécifiés pour comprendre les objectifs de test et l'importance relative du test.
- Les tests doivent aussi être compréhensibles par d'autres parties prenantes comme les développeurs, qui devront faire des revues de tests, et les auditeurs, qui pourront avoir à approuver les tests.
- Les tests doivent être conçus pour couvrir toutes les interactions du logiciel avec les acteurs (p. ex., les utilisateurs finaux, d'autres systèmes), et pas uniquement les interactions se produisant par l'intermédiaire de la partie visible de l'interface utilisateur. Les communications entre processus, l'exécution de batchs et autres interruptions interagissent également avec le logiciel et peuvent contenir des défauts. C'est pourquoi l'Analyste de Test doit concevoir des tests pour réduire ces risques.
- Les tests doivent être conçus pour tester les interfaces entre les différents objets de test.

1.5.1 Cas de test Concrets et cas de test Logiques

L'une des tâches de l'Analyste de Test est de déterminer les types de cas de test les plus adaptés à une situation donnée. Des cas de test concrets apportent toute l'information et les procédures nécessaires au testeur pour l'exécution du cas de test (y compris les données à utiliser) et la vérification des résultats. Les cas de test concrets sont utiles lorsque les exigences sont bien définies, quand l'équipe de test a peu d'expérience et quand une vérification externe des tests, comme lors d'un audit, est requise. Les cas de test concrets sont tout à fait reproductibles (c.à.d. qu'un testeur différent obtiendra les mêmes résultats), mais nécessiteront aussi un important effort de maintenance et pourront avoir tendance à limiter l'ingéniosité du testeur pendant l'exécution.

Les cas de test logiques apportent des directives sur ce qui doit être testé, mais permettent à l'Analyste de Test de faire varier les données prévues et même la procédure à suivre lors de l'exécution du test. Les cas de test logiques peuvent apporter une meilleure couverture que les cas de test concrets puisqu'ils varieront d'une certaine façon à chaque exécution. Cela ne les rend pas reproductibles. Les cas de test logiques seront intéressants lorsque les exigences sont mal définies, quand l'Analyste de Test qui exécutera les tests a de l'expérience avec le test et aussi avec le produit, et quand il n'est pas nécessaire d'avoir une documentation formelle (p. ex., aucune réalisation d'audit). Les cas de test logiques peuvent être définis tôt dans le processus d'exigences, avant même que les exigences ne soient définies. Ces cas de test peuvent être utilisés pour développer des cas de test concrets quand les exigences sont mieux définies et plus stables. Dans le cas, la création des cas de test est faite de façon séquentielle, allant du logique au concret, bien que seuls les cas de test concrets soient exécutés.

1.5.2 Création des Cas de Test

Les cas de test sont conçus par l'élaboration progressive et le raffinement des conditions de test identifiées, en utilisant des techniques de conception des tests (voir chapitre 3) identifiées dans la stratégie de test et/ou dans le plan de test. Les cas de test doivent être répétables, vérifiables et reliés aux bases de test (p. ex., les exigences), comme cela est indiqué dans la stratégie de test utilisée.

La conception d'un cas de test comprend l'identification des éléments suivants :

- Objectif
- Pré-conditions, telles que des exigences pour un environnement de test projet ou localisé et le plan pour sa livraison, l'état du système, etc.
- Les exigences en données de test (à la fois pour les données en entrée des cas de test et pour les données qui doivent être présentes dans le système pour permettre l'exécution des tests)
- Les résultats attendus
- Les post-conditions, comme les données modifiées, l'état du système, les "triggers" (déclencheurs) pour les traitements suivants, etc.

Le niveau de détail des cas de test, qui impacte le coût de développement et le niveau de répétabilité, devraient être précisés avant la création effective des cas de test. Des cas de test moins détaillés laisseront à l'Analyste de Test plus de flexibilité lors de l'exécution des cas de test tout en lui donnant l'opportunité d'enquêter sur des domaines potentiellement intéressants. Avec peu de détail, cependant, les tests sont moins reproductibles.

La définition du résultat attendu d'un test constitue souvent une difficulté particulière. Le faire manuellement est souvent pénible et sujet à l'erreur. Si possible, il est préférable de trouver ou d'identifier un oracle de test automatisé. En identifiant les résultats attendus, les testeurs se préoccupent non seulement des sorties visibles à l'écran, mais aussi des données et des post-conditions pour l'environnement. Si les bases de test sont clairement identifiées, définir le bon résultat devrait en théorie être simple. Cependant, les bases de test sont souvent vagues, contradictoires, insuffisantes pour couvrir les domaines clés, ou totalement inexistantes. Dans ce cas, un Analyste de

Test doit posséder la bonne expertise métier ou y avoir accès . De plus, même lorsque les bases de test sont bien spécifiées, des interactions complexes relatives à des stimuli et réponses complexes peuvent rendre difficile la définition des résultats attendus. Par conséquent, il est essentiel d'avoir un oracle de test. Exécuter des cas de test sans un moyen de déterminer l'exactitude des résultats présente une valeur ajoutée et un bénéfice très faibles, et est souvent à l'origine de faux rapports d'erreur ou d'une confiance infondée dans le système.

Les activités décrites ci-dessus peuvent être appliquées à tous les niveaux de test, même si les bases de test varient. Par exemple, les tests d'acceptation utilisateur peuvent être initialement basés sur la spécification des exigences, sur les cas d'utilisation, et sur les définitions de processus métier, tandis que les tests de composant seront plutôt basés sur des spécifications de conception de bas niveau, des « user stories » et le code lui-même. Il est important de se souvenir que ces activités se déroulent sur tous les niveaux de test, même si la cible du test peut varier. Par exemple, le test fonctionnel au niveau unitaire est conçu pour garantir qu'un composant en particulier fournisse la fonctionnalité telle que spécifiée dans la conception détaillée de ce composant. Le test fonctionnel au niveau intégration, consiste à vérifier que les composants interagissent les uns avec les autres et apportent la fonctionnalité par le biais de leur intégration. Au niveau système, les fonctionnalités de bout-en-bout doivent être l'objet du test. Lors de l'analyse et de la conception des tests, il est important de se souvenir du niveau visé par les tests et de l'objectif du test. Cela aide à déterminer le niveau de détail nécessaire, de même que tout outil pouvant être utile (p. ex., pilotes et bouchons au niveau composant).

Pendant le développement des conditions de test et des cas de test, une certaine quantité de documentation est créée, sous forme de livrables. En pratique, le degré de documentation des livrables varie considérablement. Cela peut être affecté par les éléments suivants:

- Risques projet (ce qui doit/ne doit pas être documenté)
- La "valeur ajoutée" que la documentation apporte au projet
- Les standards à suivre et/ou les régulations à satisfaire
- Le modèle de cycle de vie utilisé (p. ex., une approche Agile vise le « juste assez » en termes de documentation)
- Les exigences relatives à la traçabilité des bases de test par l'intermédiaire de l'analyse des tests et de la conception.

Selon le périmètre du test, l'analyse des tests aborde les caractéristiques de qualité pour le (ou les) objet(s) de test. Le standard ISO 25000 [ISO25000] (qui remplace ISO 9126) constitue une référence très utile. D'autres caractéristiques peuvent s'appliquer lors du test de systèmes matériels/logiciels.

Les processus d'analyse et de conception des tests peuvent être améliorés si on les couple à des revues et à de l'analyse statique. En fait, les activités d'analyse et de conception des tests sont souvent une forme de test statique parce que des problèmes peuvent être trouvés dans des documents de base pendant ces activités. L'analyse et la conception des tests basées sur la spécification des exigences est une très bonne façon de se préparer à une réunion de revue des exigences. Le fait de lire les exigences pour créer des cas de test nécessite une bonne compréhension des exigences et une capacité à déterminer une façon d'évaluer la satisfaction de celles-ci. Cette activité permet souvent de découvrir des exigences imprécises, ou non-testables, ou même sans critère d'acceptation. Des produits comme les cas de test, les analyses de risques et les plans de test peuvent aussi faire l'objet de revues.

Certains projets, comme ceux qui suivent un cycle de vie Agile, peuvent avoir des exigences très peu documentées. Elles peuvent prendre la forme de « user stories »qui décrivent des parties de fonctionnalités, petites mais démontrables. Une « »user story» » devrait inclure une définition des critères d'acceptation. Si le logiciel permet de démontrer que les critères d'acceptation sont satisfaits, il est en général considéré comme prêt pour l'intégration avec les autres fonctionnalités développées, à moins qu'il n'ait déjà été intégré pour démontrer sa fonctionnalité.

Pendant la conception des tests, les exigences relatives à l'infrastructure de test peuvent être définies, même si en pratique, elles ne peuvent être finalisées avant l'implémentation des tests. Il est utile de rappeler que l'infrastructure des tests ne se limite pas aux objets de test et au « testware ». Par exemple, les exigences d'infrastructure peuvent inclure des salles, des équipements, du personnel, du logiciel, des outils, des périphériques, des équipements de communications, des autorisations utilisateurs, et tout autre élément nécessaire à l'exécution des tests.

Les critères de sortie pour l'analyse des tests et la conception des tests vont varier en fonction des paramètres du projet mais tous les éléments discutés dans ces deux sections devraient être pris en compte par rapport aux critères de sortie définis. Il est important que les critères soient mesurables et qu'ils permettent d'assurer que tous les éléments et la préparation requise pour les étapes suivantes aient été fournis.

1.6 Implémentation des tests

L'implémentation des tests est l'aboutissement de la conception des tests. Cela comprend la création de tests automatisés, l'organisation des tests (à la fois manuels et automatisés) dans un ordre d'exécution, la finalisation des données et des environnements de test, et la création d'un planning d'exécution des tests comprenant l'affectation des ressources, pour permettre le démarrage de l'exécution des tests. Cela implique également de vérifier, par rapport à des critères d'entrée explicites et implicites pour le niveau de test en question, et de s'assurer que les critères de sortie de la précédente étape dans le processus ont été satisfaits. Si les critères de sortie n'ont pas été satisfaits, pour le niveau de test ou pour une étape dans le processus de test, l'effort d'implémentation des tests sera probablement affecté par des retards, par une qualité insuffisante ou par un effort supplémentaire imprévu. Il est important de s'assurer que tous les critères de sortie ont été satisfaits avant de commencer l'implémentation des tests.

Plusieurs éléments peuvent être pris en compte dans la définition de l'ordre d'exécution. Dans certains cas, il est intéressant d'organiser les cas de test en suites de test (c.à.d., un groupe de cas de test). Cela permet d'organiser le test de façon à ce que certains cas de test soient exécutés ensemble. Si une stratégie de test basée sur les risques est en place, la priorité des risques peut conditionner l'ordre d'exécution des cas de test. D'autres facteurs peuvent déterminer l'ordre d'exécution, par exemple, la disponibilité des bonnes personnes, de l'équipement, des données et des fonctionnalités à tester. Le code est fréquemment livré en différentes lots et l'effort de test doit être coordonné avec l'ordre dans lequel les différentes parties du code sont disponibles pour le test. Dans des modèles de cycle de vie incrémental en particulier, il est important que l'Analyste de Test se synchronise avec l'équipe de développement pour s'assurer que le logiciel soit livré dans un ordre permettant le test. Lors de l'implémentation des tests, Les Analystes de Test devraient définir et confirmer l'ordre dans lequel les tests manuels et automatisés devront être exécutés, en vérifiant avec attention les contraintes pouvant impliquer un ordre d'exécution particulier des tests. Les dépendances doivent être documentées et vérifiées.

Le niveau de détail et la complexité du travail fait lors de l'implémentation des tests peuvent être influencés par le détail des cas de test et les conditions de test. Dans certains cas, des contraintes règlementaires sont applicables, et les tests doivent apporter des preuves du respect des standards applicables, tels que les standards DO-178B « United States Federal Aviation Administration's » /ED 12B [RTCA DO-178B/ED-12B].

Comme indiqué plus haut, le test a besoin de données de test et, dans certains cas, ces ensembles de données peuvent être très grands. Lors de l'implémentation, les Analystes de Test créent des entrées et des données d'environnement pour alimenter les bases de données et autres référentiels du même type. Les Analystes de Test créent aussi les données à utiliser dans le cadre d'une automatisation pilotée par les données, de même que dans le cadre du test manuel.

L'implémentation des tests concerne aussi l'environnement de test(s). A ce stade, le ou les environnement(s) doivent être entièrement en place et vérifiés avant l'exécution des tests. Il est essentiel d'avoir un environnement de test "adapté au besoin", (c.à.d., que l'environnement de test doit permettre de mettre en évidence les défauts présents lors du test, de fonctionner normalement en l'absence d'erreurs, et de reproduire de façon adéquate l'environnement de produit ou de l'utilisateur final pour les niveaux de test les plus hauts). Il peut être nécessaire de modifier l'environnement de test pendant l'exécution des tests en fonction de changements anticipés ou non, des résultats de test ou d'autres considérations. Si l'environnement change pendant l'exécution, il est important d'évaluer l'impact des changements sur les tests ayant déjà été exécutés.

Pendant l'implémentation des tests, les testeurs doivent s'assurer que les personnes responsables de la création et de la maintenance de l'environnement de test soient identifiées et disponibles, et que tous le testware , les outils de support au test et les processus associés soient prêts à l'usage. Cela comprend, la gestion de configuration, l'enregistrement des tests et leur gestion. De plus, les Analystes de Test doivent vérifier les procédures de collecte des données permettant l'évaluation des critères de sortie et le reporting des résultats de test.

Il est prudent d'utiliser une approche équilibrée pour l'implémentation des tests, telle qu'elle aura été définie lors de la gestion des tests. Par exemple, des stratégies de test analytiques basées sur les risques sont souvent associées à des stratégies de test dynamiques. Dans ce cas, un certain pourcentage de l'effort de test est affecté à du test qui ne suit pas des scripts de test prédéfinis (test « non-scriptés»).

Le test non-scripté ne doit pas être utilisé par hasard ou sansbut précis car il ne pourra être contrôlé en termes de durée et de couverture que s'il est règlementé et planifié. Au fil des années, les testeurs ont développé différentes techniques basées sur l'expérience, telles que les attaques, l'estimation d'erreur [Myers79], et le test exploratoire. L'analyse des tests, la conception des tests, et l'implémentation des tests ont toujours lieu mais se produisent durant l'exécution des tests. Avec de telles stratégies de test dynamiques, le résultat de chaque test influence l'analyse, la conception et l'implémentation des tests suivants. Même si ces stratégies sont simples et souvent efficaces pour trouver des défauts, il y a des inconvénients. Ces techniques nécessitent l'expertise de l'Analyste de Test, leur durée est difficile à prévoir, la couverture est difficile à suivre et la répétabilité peut ne pas être assurée en l'absence de bonne documentation et d'un outil de support.

1.7 Exécution des tests

L'exécution des tests commence lorsque l'objet de test est livré et que les critères d'entrée pour l'exécution des tests sont satisfaits (ou abandonnés). Les tests devraient être exécutés conformément à ce qui a été prévu lors de l'implémentation des tests, mais l'Analyste de Test devrait disposer du temps nécessaire pour assurer la couverture de scénarios de test supplémentaires intéressants et des comportements pouvant être observés lors du test (toute défaillance observée doit être décrite avec les éventuelles divergences par rapport au scénario de test initialement décrit et tout élément nécessaire à sa reproduction). Cette intégration de techniques pour des tests scriptés et non-scriptés (p. ex., exploratoire) permet de se prémunir de certains problèmes dus à des divergences entre le script et le test à effectuer en pratique. Cela aide à contourner le paradoxe du pesticide.

Au cœur de l'activité d'exécution des tests se trouve la comparaison entre les résultats obtenus et les résultats attendus. Les Analystes de Test doivent être attentifs et rester concentrés sur ces tâches, sans quoi tout le travail de conception et d'implémentation des tests peut être perdu si des défaillances ne sont pas relevées (faux-négatif) ou si des comportements corrects sont classés comme incorrects (faux-positif). Si les résultats attendus et obtenus ne correspondent pas, c'est qu'un incident est survenu. Les incidents doivent être attentivement examinés pour déterminer leur cause

(qui peut être ou ne pas être un défaut dans l'objet de test) et pour collecter les données nécessaires pour aider à la résolution de l'incident (voir Chapitre 6 pour plus de détails sur la gestion des défauts).

Quand une défaillance est identifiée, la documentation des tests (spécification de test, cas de test, etc.) devrait être revue avec attention pour être vérifiée. Un document de test peut être incorrect pour différentes raisons. S'il est incorrect, il devrait être corrigé et le test ré-exécuté. Comme des changements dans les bases et objet de test peuvent rendre un cas de test incorrect, même après plusieurs exécutions passantes, les testeurs devraient toujours avoir en tête que les résultats observés peuvent être dus à un test incorrect.

Pendant l'exécution des tests, les résultats de test doivent être enregistrés correctement. Les tests ayant été exécutés mais sans enregistrement des résultats peuvent avoir à être exécutés à nouveau pour identifier le bon résultat, ce qui entraînera de l'inefficacité et des retards. (Noter qu'un enregistrement/log adéquat peut intégrer l'information relative aux problèmes de couverture et de répétabilité associés à des techniques de test comme les tests exploratoires). Comme l'objet de test, le « testware », et l'environnement de tests peuvent évoluer, l'enregistrement doit préciser la version spécifique testée, de même que la configuration précise de l'environnement. L'inscription des tests (Test Logging) apporte un enregistrement chronologique de détails pertinent sur l'exécution des tests.

L'enregistrement des résultats (Results logging) s'applique à la fois aux tests individuels et aux activités ou évènements de test. Chaque test devrait être identifié de façon unique et son statut enregistré au moment de l'exécution des tests. Tout évènement affectant l'exécution des tests devrait aussi être enregistré. Une information suffisante devrait être enregistrée pour permettre la mesure de la couverture et la documentation des motifs de retard ou d'interruption dans le test. De plus l'information doit être enregistrée pour permettre le contrôle du test, le reporting sur l'avancement, la mesure des critères de sortie, et l'amélioration du processus de test.

L'inscription/enregistrement (Logging) varie en fonction du niveau de test et de la stratégie. Par exemple, avec des tests de composant automatisés, les tests automatisés devront générer la plupart de l'information à enregistrer. Avec du test manuel, l'Analyste de Test enregistrera l'information relative à l'exécution des tests, en général dans un outil de gestion des tests qui permet de suivre l'information sur l'exécution des tests. Dans certains cas, comme avec l'implémentation des tests, la quantité d'information enregistrée sur l'exécution des tests dépend d'exigences règlementaires ou propres à un audit.

Dans certains cas, les utilisateurs ou les clients peuvent participer à l'exécution des tests. Cela peut contribuer à augmenter leur confiance dans le logiciel, même si cela présuppose que ces tests trouvent peu de défauts. Une telle supposition n'est en général pas applicable aux premiers niveaux de test mais le sera pour les tests d'acceptation.

Les éléments spécifiques qui suivent doivent être pris en compte lors de l'exécution des tests:

- Releveret investiguer des éléments étranges non-significatifs. Des observations ou résultats pouvant sembler étranges sont souvent indicateurs de défauts qui, comme des icebergs, menacent sous la surface.
- Vérifier que le produit ne fait pas ce qu'il n'est pas censé faire. Vérifier que le produit fait ce qu'il est censé faire correspond à une préoccupation classique de test, mais l'Analyste de Test doit aussi être sûr que le produit n'adopte pas un mauvais comportement en faisant quelque chose qu'il ne devrait pas faire (par exemple, une fonction supplémentaire non souhaitée).
- Construire la suite de test et veiller à sa mise à jour et à son évolution dans le temps. Le code va évoluer et des tests supplémentaires seront nécessaires pour couvrir ces nouvelles fonctionnalités, de même que pour vérifier les régressions dans les autres parties du logiciel. Des manques dans le test sont souvent découverts pendant l'exécution. Construire la suite de test est un processus continu.

- Prendre des notes pour le prochain effort de test. Les tâches de test ne se terminent pas lorsque le logiciel est fourni aux utilisateurs ou distribué sur le marché. Une nouvelle version ou livraison du logiciel sera probablement produite, donc, la connaissance doit être conservée et transférée aux testeurs en charge du prochain effort de test.
- Ne pas s'attendre à rejouer tous les tests manuels. Il est irréaliste de penser que tous les tests manuels seront rejoués. Si un problème semble survenir, l'Analyste de Test doit l'étudier et le répertorier plutôt que de penser qu'il sera détecté par une prochaine exécution des cas de test.
- Utiliser les données de l'outil de suivi des défauts pour des cas de test supplémentaires. Penser à créer des cas de test pour des défauts découverts lors de tests non-scriptés ou exploratoires et les ajouter à la suite de test de régression.
- Trouver les défauts avant les tests de régression. Le temps pour les tests de régression est souvent limité et trouver des défauts pendant le test de régression peut entraîner des retards de planning. Les tests de régression, en général, ne trouvent pas la majeure partie des défauts, principalement car il s'agit de tests qui ont déjà été exécutés (p. ex., pour une version précédente du logiciel) et des défauts devraient avoir été détectés lors de ces précédentes exécutions. Cela ne signifie pas que les tests de régression sont inutiles mais seulement que l'efficacité des tests de régression, en terme de capacité de détection de nouveaux défauts, est plus basse que celle des autres tests.

1.8 Évaluation des Critères de Sortie et Reporting

Selon le point de vue du processus de test, le suivi de l'avancement du test consiste à collecter le bon ensemble d'information permettant de faire des rapports relatifs aux exigences. Cela implique la mesure de l'avancement jusqu'à l'achèvement. Quand les critères de sortie sont définis au moment de la planification, il peut y avoir une séparation des critères «doit obligatoirement » et « devrait ». Par exemple, le critère peut indiquer « aucun défaut de « priorité-1 » ou de « priorité 2 » et, 95% de taux de succès sur tous les cas de test ».Dans ce cas, une défaillance devant satisfaire au critère « doit obligatoirement » pourra faire échouer le critère de sortie même si 93% de taux de passage pourrait permettre au projet de passer au niveau suivant. Le critère de sortie doit être clairement défini afin de pouvoir être évalué objectivement.

L'Analyste de Test est responsable de la fourniture de l'information utilisée par le Test Manager pour évaluer la satisfaction des critères de sortie et pour assurer la précision des données. Si, par exemple, le système de gestion des tests fournit les statuts suivants pour la complétude des cas de test :

- Passé
- En échec
- Passé avec exception

Alors l'Analyste de Test doit être très clair sur la signification de chacun de ces statuts afin de les utiliser à bon escient. "Passé avec exception" veut-il dire qu'un défaut a été trouvé mais qu'il n'affecte pas les fonctionnalités de système? Comment classer un défaut d'utilisabilité qui perturbe l'utilisateur? Si le taux de succès est un critère de sortie impératif, alors, classer un défaut « En échec » plutôt que « Passé avec exception » sera un facteur critique. Une attention particulière doit aussi être accordée aux cas de test ayant le statut « En échec » mais dont la cause de la défaillance n'est pas un défaut (p. ex., l'environnement de test n'a pas été correctement configuré) S'il y a le moindre risque de confusion avec les métriques suivies ou l'utilisation des différents statuts, l'Analyste de Test doit clarifier cela avec le Test Manager afin que l'information puisse être suivie de façon précise et consistante tout au long du projet.

Il est fréquent de demander à l'Analyste de Test un rapport durant les cycles de test de même qu'une contribution à l'élaboration du rapport final à la fin du test. Cela peut demander la collecte de métriques à partir des systèmes de gestion des défauts et de test, de même que l'évaluation de la couverture et de l'avancement global. L'Analyste de Test doit être capable d'utiliser les outils de

reporting et de fournir l'information demandée au Test Manager pour lui permettre d'extraire l'information dont il a besoin.

1.9 Activités de Clôture des tests

Une fois l'exécution des tests terminée, les sorties principales de l'effort de test devraient être collectées et soit transmises à la bonne personne, soit archivées. L'ensemble constitue les activités de clôture des tests. L'Analyste de Test devrait s'attendre à être impliqué dans la remise des livrables à ceux qui en ont besoin. Par exemple, les défauts connus, reportés ou acceptés devraient être communiqués à ceux qui en auront besoin. Par exemple, les défauts connus, reportés ou acceptés devraient être communiqués à ceux qui utiliseront le système ou en feront le support. Les tests et environnements de tests devraient être donnés aux responsables des tests de maintenance. Un autre livrable peut être un ensemble de tests de régression (automatisé ou manuel). L'information sur les livrables du test doit être clairement documentée, avec les liens nécessaires et une gestion appropriée des droits d'accès.

L'Analyste de Test devrait aussi participer aux réunions de rétrospective ("lessons learned") où les leçons importantes (tirées du projet de test ou plus généralement du cycle de vie de développement logiciel) peuvent être documentées, accompagnées de plans pour renforcer les bonnes pratiques et éliminer, ou au moins contrôler, les mauvaises. L'Analyste de Test est une source importante d'information et de savoir pour ces réunions et doit y participer si des éléments destinés à l'amélioration du processus doivent être collectés. Si seul le Test Manager participe à la réunion, l'Analyste de Test doit lui fournir la bonne information afin qu'il puisse présenter une image précise du projet.

Il est également nécessaire d'archiver dans le système de gestion de configuration, les résultats, les enregistrements (logs), les rapports et tout autre document ou livrable. Cette tâche revient souvent à l'Analyste de Test et constitue une importante activité de clôture des tests, en particulier si un projet futur a besoin d'utiliser l'information.

Même si c'est en général le Test Manager qui détermine l'information qui doit être archivée, l'Analyste de Test devrait aussi se demander quelle information serait nécessaire si le projet devait recommencer dans le futur. Penser à cela à la fin du projet peut permettre d'économiser des mois d'effort si le projet recommence plus tard ou avec une autre équipe.

2. Gestion des Tests: Responsabilités de l'Analyste de Test - 90 mn.

Mots clé

risque-produit, analyse de risque, identification des risques, niveau de risque, gestion des risques, réduction des risques, test basé sur les risques, supervision des tests, stratégie de test

Objectifs d'apprentissage pour Gestion des Tests: Responsabilités de l'Analyste de Test

2.2 Contrôle et Supervision de l'Avancement des tests

AT-2.2.1 (K2) Expliquer les types d'information devant être suivis durant le test pour permettre une supervision et un contrôle adéquat du projet

2.3 Tests Distribués, Externalisés et Internalisés

AT-2.3.1 (K2) Fournir des exemples de bonnes pratiques de communication dans un environnement de test 24h/24

2.4 Tâches de l'Analyste de Test dans le Test Basé sur les Risques

AT-2.4.1 (K3) Pour un projet particulier, participer à l'identification des risques, faire une évaluation des risques et proposer des actions de réduction des risques adaptées

2.1 Introduction

Bien qu'il y ait de nombreux domaines dans lesquels l'Analyste de Test interagit avec le Test Manager et lui fournit des données, cette section se concentre sur les domaines du processus de test dans lesquels l'Analyste de Test joue un rôle majeur. Le Test Manager ira chercher auprès de l'Analyste de Test l'information dont il a besoin.

2.2 Contrôle et Supervision de l'Avancement des tests

L'avancement des tests peut être supervisé selon cinq dimensions primaires:

- Risques produit (qualité)
- Défauts
- Tests
- Couverture
- Confiance

Les risques produit, les défauts, les tests, et la couverture peuvent être mesurés et sont souvent mesurés et rapportés de différentes façons pendant le projet ou l'opération par l'Analyste de Test. La confiance, bien qu'étant mesurable au travers d'études, est en général rapportée de façon subjective. Collecter les informations nécessaires pour alimenter ces métriques fait partie du travail quotidien de l'Analyste de Test. Il est important de ce souvenir que l'exactitude de ces données est critique puisque des données inexactes génèreront une information inexacte et pourront aboutir à des conclusions inexactes. Dans le pire des cas, des données inexactes provoqueront de mauvaises décisions des responsables et nuiront à la crédibilité de l'équipe de test.

S'il utilise une approche de test basée sur les risques, l'Analyste de Test devrait suivre:

- Les risques ayant été réduits par le test
- Les risques considérés comme non-réduits

Suivre la réduction des risques se fait souvent avec un outil qui suit aussi la réalisation des tests (p. ex., outils de gestion des tests). Cela demande à ce que les risques identifiés soient liés aux conditions de test, elles-mêmes liées aux cas de test qui réduiront les risques (si les cas de test sont exécutés avec succès). Ainsi, l'information sur la réduction des risques est mise à jour automatiquement quand les cas de test sont mis à jour. Cela peut se faire pour les tests manuels et pour les tests automatisés.

Le suivi des défauts se fait en général avec un outil de suivi des défauts. Quand les défauts sont enregistrés, l'information pour classifier chaque défaut est aussi enregistrée. Cette information est utilisée pour produire des tendances et des graphiques indiquant l'avancement du test et la qualité du logiciel. L'information de classification est discutée de façon plus détaillée dans le chapitre « Gestion des Défauts ». Le cycle de vie utilisé peut avoir un effet sur la quantité de documentation enregistrée et sur les méthodes utilisées pour enregistrer l'information.

Pendant le test, l'information sur le statut des cas de test doit être enregistrée. Cela est, en général, fait avec un outil de gestion des tests mais peut aussi se faire avec des moyens manuels si nécessaire. L'information sur les cas de test peut inclure :

- Le statut de création des cas de test (p. ex., conçu, revu)
- Le statut d'exécution des cas de test (p. ex., passé avec succès, en échec, bloqué, non-exécuté)
- L'information d'exécution des cas de test (p. ex., date et heure, nom du testeur, données utilisées)

- Les artéfacts d'exécution des cas de test (p. ex., copies d'écran, logs)

Comme avec les risques, les cas de test devraient être liés aux exigences qu'ils testent. Il est important pour l'Analyste de Test de se souvenir que si un cas de test A est lié à une exigence A, et que c'est le seul cas de test lié à cette exigence, alors si le cas de test A est exécuté avec succès, l'exigence A sera considérées comme satisfaite. Cela peut être juste ou non. Dans beaucoup de cas, plusieurs cas de test sont nécessaires au test rigoureux d'une exigence mais, à cause du manque de temps, seul un sous-ensemble de ces tests est en général créé. Par exemple, si 20 cas de test étaient nécessaires pour bien tester l'implémentation d'une exigence, mais que seulement 10 ont été créés et exécutés, alors la couverture de l'exigence sera de 100% alors qu'en réalité seulement 50% de couverture a été atteint. Un suivi exact de la couverture et du statut des exigences elles-mêmes peut être utilisé comme une mesure de confiance.

La quantité d'information à enregistrer (et son niveau de détail) dépend de plusieurs facteurs, comme le modèle de cycle de vie de développement logiciel. Par exemple, sur des projets Agile l'information enregistrée sur le statut sera moins importante, avec l'interaction forte de l'équipe et davantage de communication en face-à-face.

2.3 Tests Distribués, Externalisés et Internalisés

Dans beaucoup de cas, seule une partie de l'effort de test est fourni par une même équipe composée de collègues appartenant au reste de l'équipe du projet et située au même et unique que le reste de l'équipe du projet. Si l'effort de test est mené à différents endroits, cet effort de test peut être appelé « distribué ». S'il est mené dans un endroit unique il peut être qualifié de « centralisé ». Si l'effort de test est mené à plusieurs endroits par des personnes n'étant pas employées de l'entreprise et qui ne sont pas co-localisées avec l'équipe projet, l'effort de test peut être appelé « outsourcé / externalisé ». Si l'effort de test est mené par des personnes qui ne sont pas co-localisées avec l'équipe projet mais qui ne sont pas des employés de l'entreprise, cet effort de test peut être appelé internalisé.

S'il travaille dans un projet ayant une partie de l'équipe répartie à différents endroits, voire dans différentes sociétés, l'Analyste de Test doit porter particulièrement attention à l'efficacité de la communication et au transfert d'information. Certaines organisations travaillent sur un modèle « 24 heures de test » dans lequel l'équipe située dans une zone horaire est censée passer le travail à l'équipe d'une autre zone horaire pour permettre un test continu sur 24 heures. Transmettre et recevoir le travail demande une planification spéciale de la part de l'Analyste de Test. Si bonne gestion est importante pour comprendre les responsabilités, il est aussi primordial de s'assurer que la bonne information est disponible.

Lorsque la communication verbale n'est pas possible, la communication écrite doit suffire. Cela signifie que le courriel, les rapports de statut et une utilisation efficace des outils de gestion des tests et de suivi des défauts doivent être en place et utilisés. Si l'outil de gestion des tests permet d'affecter les tests à des personnes, il peut aussi fonctionner comme un outil de planification et d'organisation du travail entre les personnes. Des défauts enregistrés de façon précise peuvent être transmis à des collègues pour être suivis, si nécessaire. Une utilisation efficace de ces systèmes de communication est vitale pour une organisation qui ne peut pas s'appuyer sur les interactions quotidiennes entre les personnes.

2.4 Tâches de l'Analyste de Test dans le Test Basé sur les Risques

2.4.1 Vue générale

Le Test Manager porte souvent la responsabilité d'établir et de gérer une stratégie de test basée sur les risques. Le Test Manager demandera en général l'implication de l'Analyste de Test pour s'assurer que l'approche basée sur les risques est correctement mise en œuvre.

L'Analyste de Test devrait être impliqué activement dans les tâches suivantes du test basé sur les risques:

- L'identification des risques
- L'évaluation des risques
- La réduction des risques

Ces tâches sont réalisées de façon itérative tout au long du cycle de vie du projet pour traiter les risques émergents, modifier les priorités, régulièrement évaluer les risques et communiquer sur leurs statuts.

Les Analystes de Test devraient travailler dans le cadre du test basé sur les risques défini pour le projet par le Test Manager. Ils devraient utiliser leur connaissance des risques métier inhérents au projet tels que les risques liés à la sécurité, aux aspects financiers et économiques ainsi qu'aux facteurs politiques.

2.4.2 Identification des Risques

En faisant appel au plus large ensemble possible de parties prenantes, le processus d'identification des risques pourra détecter le plus grand nombre de risques réels. Comme les Analystes de Test possèdent souvent un savoir unique sur le domaine métier du système sous test, ils sont particulièrement bien armés pour mener des interviews d'experts du domaine et d'utilisateurs, pour mener des évaluations indépendantes, pour utiliser ou faire utiliser des templates de risques, organiser des réunions de travail sur les risques, organiser des réunions de réflexion (« brainstorming ») avec des utilisateurs actuels et futurs, définir des check-lists de test et faire appel à l'expérience passée sur des projets ou systèmes similaires. L'Analyste de Test devrait notamment travailler de façon rapprochée avec les utilisateurs et des experts du domaine pour déterminer les zones de risques métier qui devront être traitées pendant le test. L'Analyste de Test peut aussi être particulièrement utile pour déterminer les effets potentiels des risques sur les utilisateurs et les parties prenantes.

Comme exemples de risques pouvant être identifiés dans un projet, on peut citer:

- Des problèmes d'exactitude avec une fonctionnalité logicielle, (p. ex., des calculs incorrects)
- Des problèmes d'utilisabilité, (p. ex., des raccourcis clavier insuffisants)
- Des problèmes d'apprentissage, (p. ex., le manque d'instructions pour les utilisateurs à des points de décisions importants)

Des considérations relatives au test des caractéristiques-qualité spécifiques sont couvertes au chapitre 4 de ce syllabus.

2.4.3 Evaluation des Risques

Alors que l'identification des risques consiste à identifier le plus possible de risques pertinents, l'évaluation des risques est l'étude de ces risques. En particulier, elle consiste à catégoriser chaque risque et déterminer la probabilité et l'impact associés à celui-ci.

Déterminer le niveau de risque consiste généralement à évaluer, pour chaque risque, la probabilité d'occurrence et l'impact de l'occurrence. La probabilité d'occurrence est généralement interprétée

comme la probabilité qu'a le problème potentiel de se produire dans le système sous test et d'être observé quand le système sera en production. En d'autres termes, cela est lié à un risque technique. L'Analyste Technique de Test devrait contribuer à chercher et à comprendre le niveau de risque technique potentiel pour chaque risque alors que l'Analyste de Test contribue à comprendre l'impact métier potentiel si le problème se produit.

L'impact de l'occurrence est souvent interprété comme la sévérité de l'effet sur les utilisateurs, les clients, ou les autres parties prenantes. En d'autres termes cela est lié au risque métier. L'Analyste de Test devrait contribuer à identifier et à évaluer, pour chaque risque, l'impact potentiel sur le métier ou l'utilisateur. Comme facteurs influençant le risque métier, on peut citer :

- La fréquence d'utilisation de la fonctionnalité touchée
- La perte financière
- Les éventuelles pertes ou responsabilités financières, écologiques ou sociales
- Les sanctions légales, civiles ou pénales
- Les problèmes de sécurité
- La perte de licenses
- Le manque de contournements raisonnables
- La visibilité sur la fonctionnalité
- La visibilité de l'échec conduisant à une publicité négative et à une dégradation possible de l'image de marque
- La perte de clients

A partir de l'information disponible sur les risques, l'Analyste de Test doit établir les niveaux de risques métier selon les consignes données par le Test Manager. Ils peuvent être classés avec des attributs (p. ex.., faible, moyen, haut) ou des chiffres. A moins qu'il existe une façon objective de mesurer le risque sur une échelle définie, cela ne peut pas être une mesure quantitative. Il est en général très difficile de réaliser une mesure précise de la probabilité et des conséquences en termes de coût.

Des nombres peuvent être affectés comme valeurs quantitative mais cela ne donnera pas une mesure quantitative exacte. Par exemple, le Test Manager peut décider que les risques métier seront classés avec une valeur allant de 1 à 10, avec 1 correspondant à l'impact métier le plus haut et donc le plus risqué. Une fois que la probabilité (l'évaluation du risque technique) et l'impact (l'évaluation du risque métier) ont été affectés, ces valeurs peuvent être multipliées pour déterminer une valeur globale pour chaque risque. Cette valeur globale est alors utilisée pour prioriser les activités de réduction des risques. Certains modèles de test basé sur les risques, tels que PRISMA® [vanVeenendaal12], ne combinent pas les valeurs de risques, ce qui permet à l'approche de test de traiter les risques techniques et métier séparément.

2.4.4 Réduction des Risques

Pendant le projet, les Analystes de Test devraient se charger de:

- Réduire les risques en utilisant des cas de test bien conçus démontrant de façon claire que les tests passent avec succès ou non, et en participant à des revues d'éléments logiciels tels que les exigences, les conceptions et la documentation utilisateur
- Mettre en œuvre de façon efficace les activités de réduction des risques identifiées dans la stratégie de test et le plan de test
- Réévaluer les risques connus à partir d'informations additionnelles collectées lors de l'avancement du projet, en révisant la probabilité, l'impact, ou les deux, selon l'intérêt.
- Prendre en compte de nouveaux risques identifiés à partir de l'information issue du test

Quand on parle de risques-produit (qualité), on considère que le test est une façon de réduire ces risques. En trouvant des défauts, les testeurs réduisent le risque car ils apportent de la visibilité sur les défauts et les façons de les traiter avant la livraison. Si les testeurs ne trouvent pas de défaut, le test réduit alors le risque en assurant que, sous certaines conditions (c.à.d., les conditions testées), le

système fonctionne correctement. Les Analystes de Test aident à définir les options de réduction des risques en recherchant les façons de collecter des données exactes, en créant et en testant des scénarios utilisateurs réalistes et en menant des études sur l'utilisabilité.

2.4.4.1 Prioriser les Tests

Le niveau de risque est aussi utilisé pour prioriser les tests. Un Analyste de Test peut préciser qu'il y a un risque élevé dans le domaine de l'exactitude des transactions dans un système de comptabilité. Comme conséquence, pour réduire le risque, le testeur pourrait travailler avec d'autres experts métier pour collecter un nombre important de données pouvant être traitées afin de vérifier leur exactitude. De même, un Analyste de Test pourrait préciser que des problèmes d'utilisabilité constituent un risque important pour un nouveau produit. Plutôt que d'attendre les tests d'acceptation pour découvrir des problèmes, l'Analyste de Test pourrait prioriser du test d'utilisabilité afin qu'il soit fait pendant le niveau intégration et ainsi aider à identifier et résoudre les problèmes d'utilisabilité tôt dans le test. Cette priorisation doit être envisagée le plus tôt possible dans l'étape de gestion des tests afin que le planning puisse prévoir le test nécessaire au bon moment.

Dans certains cas, tous les tests sur les risques les plus élevés sont exécutés avant les tests associés à des risques de niveau inférieur et les tests sont exécutés strictement selon l'ordre de risques (souvent appelé "parcours en profondeur"); dans d'autres cas, une méthode d'échantillonnage est utilisée pour sélectionner un échantillon de tests parmi tous les risques identifiés en utilisant le poids des risques pour pondérer la sélection et en cherchant à couvrir chaque risque au moins une fois (souvent appelé "parcours en largeur ").

Que le test basé sur les risques soit mené en profondeur ou en largeur, il est possible que le temps alloué au test soit consommé avant que tous les tests n'aient été exécutés. Le test basé sur les risques permet aux testeurs de délivrer aux responsables des rapports en termes de niveau de risque restant à un certain moment, et permet aux responsables de décider de prolonger le test ou de reporter le risque restant sur les utilisateurs, clients, help desk/support technique, et/ou sur l'équipe opérationnelle.

2.4.4.2 Ajuster le Test pour les Cycles de Tests suivants

L'évaluation des risques n'est pas une activité réalisée une seule fois avant le début de l'implémentation des tests; c'est un processus continu. Chaque cycle de test prévu doit faire l'objet d'une nouvelle analyse de risque pour prendre en compte des facteurs tels que:

- Toute modification ou tout ajout dans les risques produit
- Les domaines instables ou sujets aux défauts, découverts durant le test
- Les risques par rapport aux défauts corrigés
- Les défauts typiques trouvés pendant le test
- Les domaines ayant été sous-testé (faible couverture de test)

Si du temps additionnel est alloué au test, il est envisageable d'étendre la couverture des risques aux domaines ayant un niveau de risque plus bas.

3. Techniques de test - 825 mn.

Mots clé

analyse des valeurs limites (BVA), graphe de cause à effet, test basé sur les check-lists , classification arborescente, test combinatoire, table de décision, taxonomie de défauts, technique basée sur les défauts , analyse par domaine, estimation d'erreur, partitions d'équivalence, technique basée sur l'expérience, test exploratoire, tableaux orthogonaux, test par tableaux orthogonaux , test par paires, test basé sur les exigences, technique basée sur les spécifications, test de transition d'état, agrément de test, test de cas d'utilisation, test de « »user story» »

Objectifs d'apprentissage pour Techniques de test

3.2 Techniques basées sur les Spécifications

AT-3.2.1 (K2) Expliquer l'utilisation des graphes de cause à effet

AT-3.2.2 (K3) Ecrire des cas de test à partir d'une spécification donnée en appliquant la technique de conception des tests par partitions d'équivalence pour atteindre un certain niveau de couverture

AT-3.2.3 (K3) Ecrire des cas de test à partir d'une spécification donnée en appliquant la technique de conception des tests par analyse des valeurs limites pour atteindre un certain niveau de couverture

AT-3.2.4 (K3) Ecrire des cas de test à partir d'une spécification donnée en appliquant la technique de conception des tests par table de décision pour atteindre un certain niveau de couverture

AT-3.2.5 (K3) Ecrire des cas de test à partir d'une spécification donnée en appliquant la technique de conception des tests par transition d'états pour atteindre un certain niveau de couverture

AT-3.2.6 (K3) Ecrire des cas de test à partir d'une spécification donnée en appliquant la technique de conception des tests par test par paires pour atteindre un certain niveau de couverture

AT-3.2.7 (K3) Ecrire des cas de test à partir d'une spécification donnée en appliquant la technique de conception des tests par classification arborescente pour atteindre un certain niveau de couverture

AT-3.2.8 (K3) Ecrire des cas de test à partir d'une spécification donnée en appliquant la technique de conception des tests par cas d'utilisation pour atteindre un certain niveau de couverture

AT-3.2.9 (K2) Expliquer comment les « user stories » sont utilisées pour orienter le test dans un projet Agile

AT-3.2.10 (K3) Ecrire des cas de test à partir d'une spécification donnée en appliquant la technique de conception des tests de l'analyse par domaine pour atteindre un certain niveau de couverture

AT-3.2.11 (K4) Analyser un système ou la spécification de ses exigences afin de déterminer les types of défauts susceptibles d'être trouvés et sélectionner la ou les technique(s) de conception des tests basées sur les spécifications appropriée(s)

3.3 Techniques Basées sur les Défauts

AT-3.3.1 (K2) Décrire l'application des techniques de test basées sur les défauts faire la différence avec l'usage des techniques basée sur les spécifications

AT-3.3.2 (K4) Analyser une taxonomie de défauts donnée pour son applicabilité dans un contexte donnée en utilisant les critères d'une bonne taxonomie

3.4 Techniques Basées sur l'Expérience

AT-3.4.1 (K2) Expliquer les principes des techniques basées sur l'expérience, et les bénéfices et inconvénients par rapport aux techniques basées sur les spécifications et aux techniques basées sur les défauts

AT-3.4.2 (K3) Pour un scénario donné, décrire les tests exploratoires et expliquer comment les résultats peuvent être documentés et transmis

AT-3.4.3 (K4) Pour un contexte projet donné situation, déterminer quelle technique appliquer, parmi les techniques basées sur les spécifications, sur les défauts ou sur l'expérience , afin d'atteindre un but spécifique

3.1 Introduction

Les techniques de conception des tests abordées dans ce chapitre sont réparties dans les catégories suivantes:

- Basée sur les spécifications (ou basée sur le comportement ou encore boîte noire)
- Basée sur les défauts
- Basée sur l'expérience

Ces techniques sont complémentaires et peuvent être utilisées de façon adaptée à une activité de test particulière, indépendamment du niveau de test en cours.

Noter que ces trois catégories peuvent être utilisées à la fois pour le test de caractéristiques qualité fonctionnelles ou non-fonctionnelles. Le test des caractéristiques non-fonctionnelles est discuté dans le chapitre suivant.

Les techniques de conception des tests, abordées dans les sections suivantes, se concentrent sur l'identification des données de test optimales (p. ex., partitions d'équivalence) ou sur l'obtention de séquences de test (p. ex., modèles d'état). Il est fréquent de combiner ces techniques pour créer des cas de test complets.

3.2 Techniques basées sur les Spécifications

Les techniques basées sur les spécifications sont appliquées aux conditions de test pour obtenir des cas de test à partir d'une analyse des bases de test pour un composent ou système, sans référence à sa structure interne.

Les caractéristiques principales des techniques basées sur les spécifications incluent:

- Les modèles (p. ex.. diagrammes de transition d'état et table de décisions) sont créés pendant la conception des tests selon la technique de test
- Les conditions de tests sont dérivées systématiquement de ces modèles

Certaines techniques apportent aussi des critères de couverture pouvant être utilisés pour mesurer les activités de conception des tests et d'exécution des tests. Satisfaire entièrement un critère de couverture ne signifie pas que tous les tests ont été exécutés, mais plutôt que le modèle ne requiert pas de tests supplémentaires pour augmenter la couverture sur la base de cette technique.

Les tests basés sur les spécifications s'appuient en général sur les documents d'exigences système. Comme les spécifications d'exigences devraient préciser le comportement du système, en particulier d'un point de vue fonctionnel, le test du comportement du système s'appuie souvent sur la création de cas de test à partir des exigences. Dans certains, il n'existe pas d'exigences documentées mais des exigences implicites telles le remplacement des fonctionnalités d'un système préexistant.

Il existe de nombreuses techniques de test basées sur les spécifications. Ces techniques concernent différents types de logiciels et de scénarios. Les sections suivantes présentent la mise en œuvre de chaque technique, ainsi que les limitations ou difficultés que l'Analyste de Test peut rencontrer, et la méthode par laquelle la couverture est mesurée et les différents types de défauts sont détectés.

3.2.1 Partitions d'Equivalence

La technique des partitions d'équivalence est utilisée pour réduire le nombre de cas de test nécessaire pour tester un ensemble d'entrées, de sorties, d'intervalles de valeurs ou de temps. Le découpage en partitions est utilisé pour créer des classes d'équivalence (souvent appelées partitions d'équivalence) qui sont constituées d'ensembles de valeurs qui sont traités de la même manière. En sélectionnant dans chaque partition une valeur représentative, la couverture de tous les éléments de la même partition est assurée.

Application
Cette technique est applicable à tous les niveaux de test et est adaptée lorsque tous les éléments d'un même ensemble de valeurs sont supposés être traités de la même façon et lorsque les ensembles de valeurs utilisés par l'application n'interagissent pas. La sélection des ensembles de valeur est applicable aux partitions valides et invalides (c.à.d., partitions contenant des valeurs devant être considérées comme invalides pour le logiciel testé). Il est fortement recommandé d'utiliser cette technique en la combinant avec l'analyse des valeurs limites qui étend les valeurs de test pour inclure celles situées aux extrémités des partitions. C'est une technique fréquemment utilisé pour les tests fumigatoires (« smoke test ») sur une nouvelle version ou une nouvelle livraison afin de rapidement déterminer si les fonctionnalités de base fonctionnent.

Limitations/Difficultés
Si la supposition est incorrecte et que les valeurs dans la partition ne sont pas traitées exactement de la même façon, cette technique peut laisser passer des défauts. Il est donc important de sélectionner avec attention les partitions d'équivalence. Par exemple, un champ de saisie qui accepte des valeurs positives et des valeurs négatives devrait être testé avec deux partitions valides, l'une pour les nombres positifs, l'autre pour les nombres négatifs, car ces deux ensembles seront probablement gérés de façon différente. En fonction de la possibilité ou non d'utiliser le zéro, celui-ci peut devenir une autre partition. Il est important pour l'Analyste de Test de bien comprendre le processus sous-jacent afin de déterminer les meilleures partitions de valeurs.

Couverture
La couverture est calculée en prenant le nombre de partitions pour lesquelles une valeur a été testée et en le divisant pour le nombre de partitions définies. Utiliser plusieurs valeurs pour une même partition n'augmente pas le pourcentage de couverture.

Types de Défauts
Cette technique trouve des défauts fonctionnels dans la gestion des différentes données.

3.2.2 Analyse des Valeurs Limites

L'analyse des valeurs limites est utilisée pour tester les valeurs aux limites de partitions d'équivalence. Il y a deux façons d'aborder l'analyse des valeurs limites : prendre deux ou trois valeurs pour le test. Avec deux valeurs, la valeur limite (sur la limite) et la valeur qui est juste au-delà (par le plus petit incrément possible) sont utilisées. Par exemple, si la partition inclut les valeurs 1 à 10 avec un incrément de 0,5, les deux valeurs de test pour la limite supérieure seront 10 et 10,5. Les valeurs de test pour la limite inférieure seront 1 et 0,5. Les limites sont définies par les valeurs maximum et minimum des partitions d'équivalence.

Avec trois valeurs, les valeurs inférieure, égale et supérieure à la limite sont utilisées. Selon l'exemple précédent, les valeurs pour la limite supérieure seraient 9.5, 10 et 10.5. Les valeurs pour la limite

inférieure seraient 1.5, 1 et 0.5. La décision d'utiliser deux ou trois valeurs doit se baser sur les risques associés à l'élément à tester, sachant que trois valeurs seront utilisées pour les éléments associés aux risques les plus élevés.

Application
Cette technique est applicable à tous les niveaux de test et peut être utilisée quand des partitions d'équivalence ordonnées existent. L'ordre est important à cause des concepts consistant à être sur un limite ou au-dessus. Par exemple, une plage de nombres est une partition ordonnée. Une partition correspondant à tous les objets rectangulaires ne serait pas une partition ordonnée et ne pourrait pas avoir de valeurs limites. En plus des plages de nombres, l'analyse des valeurs limites peut être appliquée aux cas suivants :

- Ensembles de variables non-numériques (p. ex., longueur)
- Boucles, y compris dans les cas d'utilisation
- Structures de données entregistrées
- Objets physiques (y compris la mémoire)
- Activités limitées en temps

Limitations/Difficultés
Comme la précision de cette technique dépend de la justesse de l'identification des partitions d'équivalence, elle est sujette aux mêmes limitations et difficultés. L'Analyste de Test devrait ainsi comprendre les incréments des partitions valides et invalides pour être capable de déteminer précisément les valeurs à tester. Seules des partitions ordonnées peuvent être utilisées pour l'analyse des valeurs limites mais cela ne se limite pas à un ensemble d'entrées valides. Par exemple, lors du test du nombre de cellules gérées par une feuille de calcul, il y aura une partition contenant le nombre de cellules jusqu'à un maximum inclus dans la partition (sur la limite) et une autre partition commençant avec une cellule au-dessus du maximum (au-delà de la limite).

Couverture
La couverture est déterminée en prenant le nombre de limites testées et en divisant par le nombre de limites identifiées (en utilisant la méthode des deux ou des trois valeurs). Cela donnera un pourcentage de couverture pour le test des limites.

Types de Défauts
L'analyse des valeurs limites permet de trouver des déplacements ou des omissions de limites, et peut détecter des cas de dépassement des limites. Cette technique trouve des défauts relatifs à la gestion des valeurs limites, en particulier des erreurs dues à trop peu ou trop de logique (c.à.d., déplacement). Elle peut aussi être utilisée pour trouver des défauts non-fonctionnels, par exemple de tolérance ou de limites de charge (p. ex., le système gère 10,000 utilisateurs simultanés).

3.2.3 Tables de Décision

Les tables de décision sont utilisées pour tester les interactions entre des combinaisons de conditions. Les tables de décision apportent une méthode claire pour vérifier les tests de toutes les combinaisons de conditions possibles et pour vérifier que toutes les combinaisons possibles sont gérées par le logiciel testé. Le but du test par table de décision est d'assurer que toutes les combinaisons de conditions sont testées. En essayant de tester toutes les combinaisons possibles, les tables de décisions peuvent devenir très grandes. Une méthode appelée « test par table de décision réduite » consiste à réduire intelligemment les nombre de combinaisons à partir de toutes celles possibles vers celles qui sont « intéressantes ». Avec cette technique, les combinaisons sont réduites à celles produisant des sorties différentes, et tous les tests redondants ou irréalistes sont supprimés. La décision d'utiliser des tables de décision entières ou réduites est en général basée sur les risques. [Copeland03]

Application
Cette technique est en général appliquée aux niveaux de test intégration, système et acceptation. En fonction du code, elle peut aussi être appliquée au test de composant, lorsqu'un composant est chargé d'un ensemble de décisions logiques. Cette technique est particulièrement utile quand les exigences sont représentées sous la forme de digrammes de flux ou de tables de règles métier. Les tables de décision sont également une technique de définition des exigences et certaines spécifications d'exigences peuvent être faites directement dans ce format. Même lorsque les exigences ne sont pas représentées sous forme de tableau ou de diagrammes de flux, des combinaisons de conditions sont en général présentes dans le texte. Lors de la création des tables de décision, il est important de prendre en compte les combinaisons de conditions définies mais aussi celles qui ne sont pas expressément définies mais qui existeront. Pour concevoir une table de décision valide, le testeur doit être capable de dériver toutes les sorties attendues pour toutes les combinaisons de conditions à partir de la spécification ou de l'oracle de test. C'est seulement lorsque toutes les conditions d'interactions sont prises en compte que la table de décision constituera un bon outil de conception des tests.

Limitations/Difficultés
Trouver toutes les conditions d'interaction peur s'avérer difficile, en particulier lorsque les exigences sont mal formulées ou n'existent pas. Il n'est pas inhabituel de s'apercevoir, lors de la préparation d'un ensemble de conditions que le résultat attendu est inconnu.

Couverture
La couverture minimale pour une table de décision est d'avoir un cas de test pour chaque colonne. Cela suppose qu'il n'y a pas de conditions composées et que toutes les combinaisons de conditions possibles ont été saisies dans une colonne. Lors de la définition de test à partir d'une table de décision, il est aussi important de considérer toutes les conditions aux limites devant être testées. Ces conditions aux limites peuvent entraîner une augmentation du nombre de cas de tests nécessaires pour tester le logiciel de façon adéquate. L'analyse des valeurs limites et les partitions d'équivalence sont complémentaires à la technique des tables de décision.

Types de Défauts
Des défauts typiques peuvent être un traitement incorrect pour des combinaisons de conditions particulières générant des résultats inattendus. Pendant la création d'une table de décisions, des défauts peuvent être trouvés dans le document de spécification. Les types de défauts les plus communs sont des omissions (aucune information sur ce qui doit réellement se passer dans une situation précise). Le test peut également trouver des problèmes avec des combinaisons de conditions non gérées ou mal gérées.

3.2.4 Graphes de Cause à Effet

Les graphes de cause à effet peuvent être obtenus à partir de toute source décrivant la logique fonctionnelle (c.à.d., les "règles") d'un programme, telle que les « user stories » ou les diagrammes de flux. Ils peuvent être utiles pour obtenir une vision globale graphique de la structure logique d'un programme et sont notamment utilisés pour créer les tables de décision. Documenter des décisions sous forme de graphes de cause à effet et/ou table de décisions permet d'atteindre une couverture de test systématique de la logique du programme.

Application
Les graphes de cause à effet s'appliquent dans les mêmes situations que les tables de décisions et s'appliquent aux mêmes niveaux de test. En particulier, un graphe de cause à effet montrera des combinaisons de conditions aboutissant à des résultats (causalité), des combinaisons de conditions excluant des résultats (not), des conditions multiples devant obligatoirement être vraies pour causer certains résultats (and) et des conditions alternatives pouvant être vraies pour causer un résultat

particulier (or). Ces relations, sont en général plus faciles à voir avec un graphe de cause à effet qu'avec une description textuelle.

Limitations/Difficultés
Les graphes de cause à effet demandent, en comparaison avec d'autres techniques de conception des tests, du temps et un effort supplémentaires pour leur apprentissage. Ils nécessitent également le soutien d'un outil. Les graphes de cause à effet ont leur propre notation qui doit être comprise par le créateur et par le lecteur du graphe.

Couverture
Chaque branche de cause à effet doit être testée, y compris les conditions de combinaisons, pour atteindre la couverture minimale. Les graphes de cause à effet fournissent un moyen de définir des contraintes sur les données et des contraintes dans un flux logique.

Types de Défauts
Ces graphes trouvent les mêmes types de défauts combinatoires que les tables de décisions. De plus, la création des graphes aide à définir le bon niveau de détail dans les bases de test, et par conséquent, aide à améliorer le détail et la qualité de ces bases de test, et aide le testeur à identifier des exigences manquantes.

3.2.5 Test de Transition d'Etat

Le test de transition d'état permet de tester la capacité d'un système à entrer dans des états définis et à en sortir par des transitions valides et invalides. Des évènements particuliers font passer le système d'un état à un autre et déclenchent certaines actions. Les évènements peuvent être associés à des conditions qui influencent le chemin de transition à prendre. Par exemple, l'évènement correspondant à une connexion avec un nom d'utilisateur et un mot de passe valides donnera lieu à une transition différente de celle déclenchée par une connexion avec un mauvais mot de passe.

Les transitions d'état sont suivies soit dans un diagramme de transition d'état montrant toutes les transitions valides entre les états, de façon graphique ; soit avec une table d'états montrant toutes les transitions possibles, valides ou invalides.

Application
Le test de transition d'état peut s'appliquer à tout logiciel ayant des états définis et étant sensible à des évènements qui déclencheront des transitions entre ces états (p. ex., un changement d'écrans). Le test de transition d'état peut être utilisé à tous les niveaux de test. Les logiciels embarqués, les logiciels web, et tout type de logiciels transactionnels sont de bons candidats à ce type de test. Les systèmes de contrôle, (c.à.d., les contrôleurs de feux de circulation), sont aussi de bons candidats pour ce type de test.

Limitations/Difficultés
La détermination des états est souvent la chose la plus difficile dans la définition des tables ou diagrammes d'état. Lorsque le logiciel dispose d'une interface utilisateur, les différents écrans affichés sont souvent utilisés pour définir les états. Pour les logiciels embarqués, les états peuvent dépendre des états que le matériel prendra.

En plus des états en eux-mêmes, l'unité de base d'un test de transition d'état est la transition seule, aussi appelé aiguillage-0 (0-switch). Le simple test de toutes les transitions trouvera certains défauts de transition d'état, mais davantage de défauts seront trouvés par les tests de séquences de transactions. Une séquence de deux transactions successives sera appelée aiguillage-1, une séquence de trois transactions successives sera appelée aiguillage-2, et ainsi de suite (ces aiguillages sont parfois appelés aiguillages N-1 , où N représente le nombre de transactions qui seront traversées. Une simple transaction, par exemple un aiguillage-0, est désignée par application de la formule aiguillage 1-1. [Bath08]

Couverture
Comme avec d'autres techniques de test, il y a une hiérarchie des niveaux de couverture de test. Le degré minimum de couverture acceptable est d'avoir visité chaque état et traversé chaque transition. Une couverture des transitions à 100% (aussi appelée 100% de couverture d'aiguillage-0 ou 100% de couverture logique des branches) garantira que chaque état est visité et chaque transition traversée, à moins que la conception du système ou le modèle de transition d'état (diagramme ou table) soit incorrect. En fonction des relations entre les états et les transitions, il peut être nécessaire de traverser certaines transitions plusieurs fois afin d'exécuter d'autres transitions au moins une fois.

Le terme "aiguillage-n " correspond au nombre de transitions couvertes. Par exemple, atteindre 100% de couverture d'aiguillage-1 demande à ce que toute séquence valide de deux transitions successives soit testée au moins une fois. Ce test pourrait faire apparaître certains types d'erreur de couverture que 100% d'aiguillage-0 manquerait.

Une "couverture Aller-Retour" correspond aux situations dans lesquelles les séquences de transitions forment des boucles. 100% de « couverture Aller-Retour » est obtenu lorsque toutes les boucles partant d'un état et revenant vers ce même état ont été testées. Cela doit être testé pour tous les états inclus dans des boucles.

Pour chacune de ces approches, un degré de couverture encore plus important pourra être atteint en essayant d'inclure toutes les transitions invalides. La couverture des exigences et la couverture des différents jeux pour le test de transition d'état doit identifier si les transitions invalides sont incluses.

Types de Défauts
Les principaux défauts trouvés peuvent être des traitements incorrects dans l'état courant à cause des traitements faits dans un état précédent, des transitions incorrectes ou non supportées, des états qui n'ont pas de sortie et le besoin d'états ou de transitions supplémentaires. Pendant la création du modèle de machine à états, des défauts peuvent être trouvés dans le document de spécification. Les types de défauts fréquents sont les omissions (aucune information sur ce qui doit réellement se passer dans une certaine situation) et les contradictions.

3.2.6 Techniques de Test Combinatoire

Le test combinatoire est utilisé lors du test de logiciel avec plusieurs paramètres, chacun avec plusieurs valeurs, ce qui aboutit à davantage de combinaisons que ce qui est faisable dans le temps imparti. Les paramètres doivent être indépendants et compatibles dans la mesure où il doit être possible de combiner toute option d'un facteur avec n'importe quelle fonction d'un autre facteur. La classification arborescente permet à certaines combinaisons d'être exclues, si certaines options ne sont pas compatibles. Cela ne veut pas dire que les facteurs combinés ne s'influenceront pas, au contraire, mais qu'ils s'influenceront de façon acceptable.

Le test combinatoire offre une façon d'identifier un sous-ensemble adéquat de ces combinaisons pour atteindre un certain niveau de couverture. Le nombre d'éléments à inclure dans les combinaisons peut être sélectionné par l'Analyste de Test, y compris des éléments seuls, des paires, des ensembles de trois éléments et plus [Copeland03]. Différents outils existent pour aider l'Analyste de Test dans cette tâche (voir www.pairwise.org pour des exemples). Ces outils demandent à ce que les paramètres et leurs valeurs soient listés (test par paires et tests par tableaux orthogonaux) ou graphiquement représentés (classification arborescentes) [Grochtmann94]. Le test par paires est une méthode appliquée pour tester des paires de variables combinées. Les tableaux orthogonaux sont des tableaux prédéfinis, mathématiquement justes qui permettent à l'Analyste de Test de remplacer les éléments à tester pour les variables dans un tableau, en produisant un ensemble de combinaisons qui atteindront un certain niveau de couverture lors du test [Koomen06]. Des outils de classification arborescente permettent à l'Analyste de Test de définir la taille des combinaisons à tester (c.à.d., combinaison de deux valeurs, trois valeurs, etc.).

Application
Le problème d'un nombre de combinaisons trop important se manifeste dans au moins deux situations liées au test. Certains cas de test contiennent plusieurs paramètres, chacun avec plusieurs valeurs possibles, par exemple un écran avec plusieurs champs de saisie. Dans ce cas, les combinaisons de valeurs constituent les données d'entrée pour les cas de test. De plus, certains systèmes se configurent selon un nombre important de dimensions, ce qui donne un très grand nombre de configurations possibles. Dans ces deux situations, le test combinatoire peut être utilisé pour identifier un sous-ensemble de combinaisons de taille raisonnable.

Pour les paramètres ayant un grand nombre de valeurs, les partitions de classes d'équivalence, ou un autre mécanisme de sélection, peuvent être appliqués en premier lieu, à chaque paramètre individuellement, pour réduire le nombre de valeur pour chaque paramètre, avant que le test combinatoire ne soit appliqué pour réduire l'ensemble des combinaisons obtenues.

Ces techniques sont en général appliquées aux niveaux de test intégration, système et intégration système.

Limitations/Difficultés
La principale limitation avec ces techniques est l'hypothèse que les résultats de quelques tests seront représentatifs de tous les tests et que ces quelques tests représentent l'utilisation cible. Si une interaction inattendue entre certaines variables se produit, elle ne sera pas forcément détectée par ce type de test, si cette combinaison particulière n'est pas testée. Ces techniques peuvent être difficiles à expliquer à un public non technique pour qui la réduction logique des tests pourra être difficile à comprendre.

Il est parfois difficile d'identifier les paramètres et leurs valeurs respectives. Il est difficile d'obtenir à la main le plus petit ensemble possible permettant d'atteindre un certain niveau de couverture. En général des outils sont utilisés pour trouver cet ensemble minimal de combinaisons. Certains outils permettent de forcer l'inclusion ou l'exclusion de (sous-) combinaisons dans la sélection finale. Cette possibilité peut être utile à l'Analyste de Test pour accentuer ou réduire certains facteurs selon la connaissance du domaine ou les informations sur l'usage du produit.

Couverture
Il y a différents niveaux de couverture. Le plus bas niveau de couverture est appelé « 1-wise » ou couverture de singleton. Il demande à ce que chaque valeur de chaque paramètre soit présente dans au moins l'une des combinaisons choisies. Le niveau de couverture suivant est appelé « 2-wise » couverture de paires. Il demande à ce que chaque paire de valeurs possible soit incluse dans au moins une combinaison. Cette idée peut être généralisée à la couverture « n-wise », qui demandera à ce que toute combinaison de valeurs possible pour n paramètres soit incluse dans l'ensemble de combinaisons sélectionnées. Plus le n est élevé, plus le nombre de combinaisons nécessaires pour atteindre 100% de couverture est important. La couverture minimale avec des techniques est d'avoir un cas de test pour chaque combinaison produite par l'outil.

Types de Défauts
Les défauts les plus communs trouvés avec ce type de test sont les défauts liés aux combinaisons de valeurs avec plusieurs paramètres.

3.2.7 Test de Cas d'Utilisation

Le test de cas d'utilisation fournit des tests de transactions basés sur les scénarios qui simulent l'usage du système. Des cas d'usage sont définis en termes d'interactions entre les acteurs et le système qui effectue des actions. Les acteurs peuvent être des utilisateurs ou des systèmes extérieurs.

Application
Le test des cas d'utilisation est en général appliqué aux niveaux de test système et acceptation. Il peut être utilisé pour le test d'intégration en fonction du niveau d'intégration et même au niveau composant en fonction du comportement du composant. Les cas d'utilisation servent souvent de base au test de performance parce qu'ils représentent un usage réaliste du système. Les scénarios décrits dans les cas d'utilisation peuvent être affectés à des utilisateurs virtuels pour créer une charge réaliste sur le système.

Limitations/Difficultés
Pour être valides, les cas d'utilisation doivent représenter des transactions réalistes avec l'utilisateur. L'information doit provenir d'un utilisateur ou d'un représentant des utilisateurs. La valeur d'un cas d'utilisation est diminuée si celui-ci ne représente pas les activités d'un utilisateur réel. Pour que la couverture de test puisse être rigoureuse, il est important de disposer d'une description précise des différents chemins (flux) possibles. Les cas d'utilisation doivent être considérés comme un guide, mais pas comme une définition exhaustive de ce qui doit être testé car ils ne fournissent pas systématiquement une définition claire de l'ensemble des exigences. Il peut également être intéressant de créer d'autres modèles, tels que des diagrammes de flux, à partir de la description des cas d'utilisation, pour améliorer la précision du test et pour vérifier le cas d'utilisation lui-même.

Couverture
La couverture minimale d'un cas d'utilisation consiste à avoir au moins un cas de test pour le principal chemin possible, et un cas de test pour chaque chemin ou flux alternatif. Les chemins alternatifs incluent les chemins concernant les exceptions et les défaillances. Les chemins alternatifs sont parfois considérés comme des extensions du chemin principal. Le pourcentage de couverture est déterminé en prenant le nombre de chemins testés et en divisant par le nombre total de chemins principaux et alternatifs.

Types de Défauts
Les défauts pourront être la mauvaise gestion de scénarios définis, l'absence de prise en compte de chemins alternatifs, le mauvais traitement de certaines conditions et une gestion des erreurs imprécise ou incorrecte.

3.2.8 Test de « »User Story» »

Dans certaines méthodes Agile , telles que Scrum, les exigences sont préparées sous la forme de « user stories » décrivant des petites unités fonctionnelles pouvant être conçues, testées et montrées dans une même itération [Cohn04]. Ces « user stories » comprennent une description de la fonctionnalité à développer, d'éventuels critères non-fonctionnels, et aussi des critères d'acceptation devant être satisfaits pour que la « »user story» » soit considérée terminée.

Application
Les « user stories » sont d'abord utilisées en environnement Agile et dans tout autre environnement itératif et incrémental du même type. Elles sont utilisées à la fois pour les tests fonctionnels et pour les tests non-fonctionnels. Les «user stories» sont utilisées à tous les niveaux de test avec comme attente, la démonstration par le développeur de la fonctionnalité développée pour la «user story» avant la remise du code aux membres de l'équipe en charge des tâches de test des niveaux suivants. (p. ex., intégration, performance).

Limitations/Difficultés
Comme les «user stories» sont de petits incréments de fonctionnalité, il peut être nécessaire de développer des pilotes et des bouchons pour réellement tester la partie de fonctionnalité qui est délivrée. Cela nécessite en général de la programmation et l'utilisation d'outils qui aideront à tester, comme des outils de test d'IHM (Interface Homme Machine). La création de « drivers » et de

bouchons est en général sous la responsabilité du développeur, même si un Analyste Technique de Test peut aussi être impliqué dans la production de code et l'utilisation d'outils de test d'IHM. Si un modèle d'intégration continue est utilisé, comme dans le cas de la plupart des projets Agiles, les besoins de pilotes et bouchons sont réduits.

Couverture
La couverture minimale d'une «user story» consiste à vérifier que chacun des critères d'acceptation spécifiés a été satisfait.

Types de Défauts
Les défauts sont en général fonctionnels lorsque le logiciel ne parvient pas à fournir la fonctionnalité spécifiée. D'autres défauts trouvés sont des problèmes d'intégration de la fonctionnalité de la nouvelle «user story» avec la fonctionnalité déjà existante. Comme les «user stories» peuvent être développées indépendamment, des défauts de performance, d'interface et de gestion des erreurs peuvent être trouvées. Il est important pour l'Analyste de Test de réaliser non seulement le test de la fonctionnalité livrée individuellement mais aussi le test d'intégration à chaque fois qu'une nouvelle «user story» est testée.

3.2.9 Analyse par Domaine

Un domaine est un ensemble défini de valeurs. Ce domaine peut être défini comme une plage de valeur pour une simple variable (un domaine à une dimension, p. ex., "les hommes âgés de 24 à 66 ans"), ou comme plusieurs plages de valeurs pour des variables associées (un domaine à plusieurs dimensions, p. ex., "les hommes âgés de 24 à 66 ans ET ayant un poids compris entre 69 et 90 kg "). Pour un domaine à plusieurs dimensions, chaque cas de test doit inclure les valeurs nécessaires pour chaque variable concernée.

L'analyse par domaine pour un domaine à une dimension utilise en général les partitions d'équivalence et l'analyse des valeurs limites. Une fois les partitions définies, l'Analyste de Test sélectionne pour chaque partition des valeurs correspondant à une valeur qui est dans la partition (IN), une valeur à l'extérieur de la partition (OUT), une valeur sur la limite de la partition (ON) et une valeur juste sous la limite de la partition (OFF). Avec ces valeurs, chaque partition est testée avec ses conditions aux limites. [Black07]

Avec des domaines à plusieurs dimensions, le nombre de cas de test générés par ces méthodes augmente de façon exponentielle avec le nombre de variables concernées, alors qu'une approche par domaine débouche en théorie sur une croissance linéaire. De plus, comme l'approche formelle intègre une théorie sur les défauts (un modèle de fautes), qui manque aux partitions d'équivalence et à l'analyse des valeurs limites, cet ensemble de tests réduit pourra trouver des défauts dans des domaines à plusieurs dimensions, que les plus grands ensembles de test heuristique ne trouveraient pas forcément. Avec des domaines à plusieurs dimensions, le modèle de test peut être construit comme une table de décision (ou "matrice de domaine"). L'identification des valeurs des cas de test pour des domaines à plusieurs dimensions nécessitera en général un support outillé.

Application
L'analyse par domaine combine les techniques utilisées pour les tables de décisions, les partitions d'équivalence et l'analyse des valeurs limites pour créer un ensemble réduit de tests qui couvre néanmoins les domaines importants et les domaines sujets aux défaillances. Elle est souvent appliquée dans les cas où les tables de décisions seraient inappropriées à cause du trop grand nombre de variables pouvant interagir. L'analyse par domaine peut être utilisée à tous les niveaux de test mais est le plus souvent utilisée aux niveaux intégration et système.

Limitations/Difficultés

Réaliser une analyse par domaine demande une bonne compréhension du logiciel afin de pouvoir identifier les différents domaines et les interactions possibles entre les domaines. Si un domaine est oublié, il manquera des tests, mais il est probable que le domaine soit détecté parce que des variables OFF et OUT pourraient se trouver dans le domaine non détecté. L'analyse par domaine est une technique puissante à utiliser lors du travail de définition des domaines de test avec le développeur.

Couverture
La couverture minimale pour l'analyse par domaine est d'avoir un test pour chaque valeur IN, OUT, ON et OFF dans chaque domaine. Lorsqu'il y a un chevauchement de valeurs (par exemple, la valeur OUT d'un domaine est la valeur IN d'un autre domaine), il n'est pas utile de dupliquer les tests. Grâce à cela, les tests réellement nécessaires sont souvent moins de quatre par domaine.

Types de Défauts
Les défauts peuvent être des problèmes fonctionnels au sein du domaine, la gestion des valeurs limites, des problèmes d'interaction de variables et de gestion d'erreur (en particulier pour les valeurs n'appartenant pas à un domaine valide).

3.2.10 Combiner les Techniques

Il arrive que des techniques soient combinées pour créer des cas de test. Par exemple, les conditions identifiées dans une table de décision peuvent donner lieu à des partitions d'équivalence pour découvrir différentes façon de satisfaire une condition. Les cas de test couvriront alors non seulement toutes les combinaisons de conditions, mais aussi, pour les conditions qui auront donné lieu à des partitions, les partitions d'équivalence couvertes par les cas de tests supplémentaires. Lors de la sélection de la technique particulière à appliquer, l'Analyste de Test devrait se demander si la technique est applicable, quelles sont les limitations et difficultés, et quels sont les objectifs du test en termes de couverture et de défauts à détecter. Il n'y aura pas forcément « LA » technique pour une situation donnée. Une combinaison de technique offrira souvent la couverture la plus large à condition qu'il y ait suffisamment de temps et les bonnes compétences pour correctement appliquer les techniques.

3.3 Techniques Basées sur les Défauts

3.3.1 Utiliser les Techniques Basées sur les Défauts

Une technique de conception des tests basée sur les défauts utilise le type de défaut recherché comme base pour la conception des tests, avec des tests dérivés systématiquement de ce que l'on sait des types de défauts. A la différence du test basé sur les spécifications, qui dérive ses tests des spécifications, le test basé sur les défauts dérive les tests de taxonomies de défauts (c.à.d., des listes par catégorie) pouvant être totalement indépendantes du logiciel testé. Les taxonomies peuvent inclure des listes de types de défauts, de causes racines, de symptôme de défaillances et de toute autre donnée relative à des défauts. Le test basé sur les défauts peut aussi utiliser, pour cibler le test, des listes de risques identifiés et de scénarii de risque. Cette technique de test permet que testeur de cibler un type spécifique de défaut ou de travailler de façon systématique avec une taxonomie de défauts regroupant les défauts connus et répandus d'un type particulier. L'Analyste de Test utilise des données de taxonomie pour déterminer l'objectif du test, qui est de trouver un type de défaut spécifique. A partir de ces éléments, l'Analyste de Test pourra créer les cas de test et conditions de test provoquant le défaut, s'il existe.

Application
Le test basé sur les défauts peut être appliqué à tous les niveaux de test mais est en général utilisé lors du test système. Des taxonomies standard existent pour les différents types de logiciels. Ce type de test indépendant du produit aide à améliorer la connaissance des standards industriels utilisés

pour dériver des cas de test. En suivant des taxonomies spécifiques à l'industrie, des métriques relatives à l'occurrence des défauts peuvent être suivies tout au long des projets et même au sein des organisations.

Limitations/Difficultés
Il existe de nombreuses taxonomies des défauts spécifiques à un type de test particulier, l'utilisabilité par exemple. Il est important de choisir la taxonomie applicable au logiciel testé, si elle existe. Par exemple, il n'y aura pas forcément de taxonomie propre au logiciel innovant. Certaines organisations pourront avoir compilé leurs propres taxonomies pour le logiciel innovant. Certaines organisations pourront avoir compilé leurs propres taxonomies de défauts probables ou fréquemment rencontrés. Quelle que soit la taxonomie utilisée, il est important de définir la couverture attendue avant de commencer à tester.

Couverture
Cette technique apporte des critères de couverture qui sont utilisés pour déterminer à quel moment tous les cas de test utiles auront été définis. En pratique, le critère de couverture pour une technique basée sur les défauts aura tendance à être moins systématique que pour une technique basée sur les spécifications puisque seules des règles générales de couverture sont données et puisque l'appréciation du degré de couverture nécessaire reste personnel. Comme avec d'autres techniques, le critère de couverture ne signifie pas que tous les tests ont été exécutés, mais plutôt que les défauts étudiés ne permettent plus de créer des tests supplémentaires utiles avec cette technique.

Types de Défauts
Les types de défauts découverts dépendent en général de la taxonomie utilisée. Si une taxonomie propre aux IHM est utilisée, la majeure partie des défauts trouvés portera sur l'IHM, mais d'autres défauts peuvent être découverts comme résultats de cette technique de test.

3.3.2 Taxonomie de Défauts

Les taxonomies de défauts sont des listes de types de défauts organisées par catégories. Ces listes peuvent être très générales et utilisées pour servir de directives à haut niveau ou peuvent être très spécifiques. Par exemple, une taxonomie utilisée pour des IHM pourrait contenir des termes généraux comme : fonctionnalité, gestion des erreurs, affichage graphique et performance. Une taxonomie détaillée pourrait inclure la liste de tous les objets graphiques possibles (en particulier pour une interface graphique utilisateur) et pourrait décrire une mauvaise gestion de ces objets comme :

- Zone de texte
  - Une donnée valide n'est pas acceptée
  - Une donnée valide est acceptée
  - La longueur d'une entrée n'est pas vérifiée
  - Des caractères spéciaux ne sont pas détectés
  - Des messages d'erreur pour les utilisateurs ne sont pas clairs
  - L'utilisateur ne peut pas corriger une erreur de saisie
  - Aucune règle n'est appliquée
- Champ de saisie de la date
  - Des dates valides ne sont pas acceptées
  - Des dates invalides ne sont pas rejetées
  - Les plages de dates ne sont pas vérifiées
  - La précision des données n'est pas gérée correctement (p. ex.., hh:mm:ss)
  - L'utilisateur ne peut pas corriger une donnée erronée
  - Aucune règle n'est appliquée (p. ex.., la date de fin doit être supérieure à la date de début)

Il existe beaucoup de taxonomies de défauts, allant de taxonomies formelles, pouvant être achetées, à des taxonomies sur mesure, faites pour les besoins spécifiques d'une organisation. Des taxonomies de défauts développées en interne peuvent aussi être utilisées pour cibler des défauts spécifiques

trouvés dans l'organisation. Lors de la création d'une nouvelle taxonomie de défauts ou lors de l'adaptation d'une existante, il est important de définir les buts et objectifs de la taxonomie. Par exemple, le but peut être d'identifier des problèmes d'interface utilisateur ayant été découverts sur des systèmes en production ou d'identifier des problèmes relatifs à la gestion des champs de saisie d'entrées.

Pour créer une taxonomie:
1. Créer un but et définir le niveau de détail désiré
2. Sélectionner une taxonomie existante pour l'utiliser comme base
3. Définir les valeurs et les défauts le plus souvent rencontrés dans l'organisation et/ou dans des pratiques extérieures

Plus la taxonomie est détaillée, plus elle sera longue à développer et à maintenir, mais cela doit permettre d'augmenter la productivité dans les résultats de test. Des taxonomies détaillées peuvent être redondantes mais elles permettent à l'équipe de test de se répartir le test sans perte d'information ou de couverture.

Une fois que la taxonomie la plus adaptée a été sélectionnée, elle peut être utilisée pour créer des conditions de test et des cas de test. Une taxonomie basée sur les risques peut aider le test à se concentrer sur un domaine spécifique. Des taxonomies peuvent également être utilisées pour des domaines non fonctionnels tels que l'utilisabilité, la performance, etc. Des listes de taxonomies sont offertes par différentes publications de l'IEEE, et sur Internet.

3.4 Techniques Basées sur l'Expérience

Les tests basés sur l'expérience utilisent les compétences et l'intuition des testeurs, avec leur expérience sur des applications ou technologies similaires. Ces tests permettent de trouver des défauts mais ne sont pas aussi appropriés que d'autres techniques pour atteindre certains niveaux de couverture de test, ou pour produire des procédures de test réutilisables. Lorsque la documentation du système est pauvre, que le temps consacré au test est très réduit ou que l'équipe de test a une forte expertise dans le système à tester, le test basé sur l'expérience peut être une bonne alternative à des approches plus structurées. Le test basé sur l'expérience peut être inadapté à des systèmes demandant une documentation de test détaillée, une parfaite répétabilité ou la capacité à mesurer précisément la couverture de test.

En utilisant des approches dynamiques ou heuristiques, les testeurs s'appuient normalement sur les tests basés sur l'expérience, et le test est plus réactif aux évènements que des approches de test planifiées à l'avance. De plus l'exécution et l'évaluation sont des tâches concurrentes. Certaines approches structurées des tests basés sur l'expérience ne sont pas entièrement dynamiques, (c.à.d., que les tests ne sont pas entièrement créés au moment où le testeur exécute les tests).

Même si certaines notions de couverture sont présentées pour les techniques discutées, les techniques basées sur l'expérience n'ont pas de critères de couverture formels.

3.4.1 Estimation d'Erreur

Avec la technique de l'estimation d'erreur, l'Analyste de Test utilise l'expérience pour deviner les erreurs potentielles pouvant avoir été faites lors de la conception et du développement du code. Une fois les erreurs attendues identifiées, l'Analyste de Test détermine les meilleures méthodes à utiliser pour découvrir les défauts associés. Par exemple, si l'Analyste de Test prévoit des erreurs du logiciel lors de l'introduction d'un mot de passe invalide, des tests seront conçus pour entrer un nombre important de valeurs différentes dans le champ mot de passe afin de voir si l'erreur a réellement été faite et a produit une défaillance visible lors de l'exécution des tests.

En plus de son utilisation comme technique de test, l'estimation d'erreur est aussi utile lors de l'analyse de risque pour identifier les différents modes de défaillance possibles. [Myers79]

Application
L'estimation d'erreur est utilisée principalement lors des tests d'intégration et des tests système, mais peut être utilisée à tous les niveaux de test. Cette technique est souvent utilisée conjointement à d'autres techniques et permet d'élargir le périmètre des tests existant. L'estimation d'erreur peut aussi être utilisée efficacement lors du test de la nouvelle version d'un logiciel, afin de commencer à tester les défauts et erreurs les plus fréquents avant de commencer des tests plus rigoureux et sous formes de script. Des check-lists et taxonomies peuvent aider à guider le test.

Limitations/Difficultés
La couverture est difficile à évaluer et varie beaucoup en fonction des capacités et de l'expérience de l'Analyste de Test. Elle sera plus efficace avec un testeur expérimenté habitué aux différents types de défauts les plus souvent introduits dans le code testé. L'estimation d'erreur est souvent utilisée mais rarement documentée et, par conséquent, peut être moins reproductible que d'autres formes de test.

Couverture
Lorsque l'on utilise une taxonomie, la couverture est déterminée selon les erreurs de données et les types de défauts correspondants. Sans taxonomie, la couverture est limitée par l'expérience et la connaissance du testeur, ainsi que par le temps disponible. L'intérêt de cette technique sera variable et fonction de la capacité du testeur à bien cibler les domaines problématiques.

Types de Défauts
Les principaux défauts sont en général ceux issus de la taxonomie retenue, ou "devinés" par l'Analyste de Test, et qui n'ont pas été trouvés par les tests basés sur les spécifications.

3.4.2 Test basé sur les check-lists

Avec la technique du test basé sur les check-lists, l'Analyste de Test expérimenté, utilise une liste de haut niveau d'éléments à noter, à vérifier, à mémoriser, ou un ensemble de règles et critères par rapport auxquels un produit doit être vérifié. Ces check-lists sont construites sur la base d'un ensemble de standards, de l'expérience et d'autres considérations. On peut citer comme exemple de test basé sur les check-lists, une check-list relative aux interfaces graphiques utilisée comme base pour les tests.

Application
Le test basé sur les check-lists est le plus efficace dans les projets disposant d'une équipe de test expérimentée, connaissant bien le logiciel à tester ou le domaine couvert par la check-list (p. ex., pour appliquer efficacement une check-list portant sur l'interface utilisateur, l'Analyste de Test peut être habitué au test d'interface utilisateur, mais pas spécifiquement au logiciel testé). Comme les check-lists sont en général de haut niveau et ne fournissent habituellement pas les étapes détaillées que l'on trouve dans les cas de test et les procédures de test, la connaissance du testeur est utilisée pour combler les manques. Sans étapes détaillées, les check-lists sont faciles à maintenir et peuvent être appliquées à de nombreuses livraisons du même type. Les check-lists peuvent être utilisées à tous les niveaux de test. Elles sont également utilisées pour le test de régression et le test fumigatoire.

Limitations/Difficultés
Le caractère de haut niveau des check-lists peut limiter la possibilité de reproduire les résultats de test. Des testeurs différents pourront interpréter différemment une même check-list et suivre des approches différentes pour satisfaire les éléments de la check-list. Cela peut entraîner des résultats différents, malgré l'utilisation d'une même check-list. Cela peut aboutir à une couverture plus large, mais parfois au détriment de la possibilité de reproduire les résultats de test. Les check-lists peuvent aussi conduire à une confiance excessive par rapport au niveau de couverture atteint, puisque le test

reste à l'appréciation du testeur. Les check-lists peuvent être dérivées de cas de tests ou de listes détaillées et ont tendance à grossir avec le temps. Leur maintenance est nécessaire pour assurer qu'elles couvrent les principaux aspects du logiciel testé.

Couverture
La couverture dépend de la check-list mais, compte-tenu de la nature de haut niveau des check-lists, les résultats seront différents en fonction de l'Analyste de Test qui appliquera la check-list.

Types de Défauts
Les principaux défauts trouvés avec cette technique seront des défaillances dues à une variation des données, de l'ordre des étapes ou du workflow général lors du test. L'utilisation de check-lists peut aider à garder un test renouvelé puisque de nouvelles combinaisons de données et de processus sont possibles durant le test.

3.4.3 Test Exploratoire

Le Test Exploratoire se caractérise par la réalisation simultanée par le testeur de la découverte du produit et de ses défauts, de la planification du travail de test à faire, de la conception et de l'exécution des tests, et du reporting des résultats. Le testeur ajuste dynamiquement les objectifs de test lors de l'exécution et ne conçoit qu'une documentation réduite. [Whittaker09]

Application
Un bon test exploratoire est géré, interactif et créatif. Il demande peu de documentation sur le système à tester et est souvent utilisé lorsque la documentation n'est pas disponible ou n'est pas adaptée à d'autres techniques de test. Le test exploratoire est souvent utilisé pour compléter d'autres techniques de test et pour servir de base au développement de cas de test supplémentaires.

Limitations/Difficultés
Le test exploratoire peut être difficile à gérer et à programmer. Sa couverture est sporadique et il est difficile à reproduire. L'une des méthodes utilisée pour gérer le test exploratoire est d'utiliser des chartes pour identifier les domaines à couvrir lors d'une session de test, et de limiter le temps pour préciser celui à consacrer au test. A la fin d'une session ou d'un ensemble de sessions de test, le Test Manager peut organiser une réunion de "débriefing" pour collecter les résultats des tests et définir des chartes pour les sessions futures. Les réunions de "débriefing" sont difficiles à organiser pour les grandes équipes ou les grands projets.

Une autre difficulté avec les sessions de test exploratoire est de les suivre précisément dans un système de gestion des tests. Cela se fait parfois en créant des cas de test qui sont en fait des sessions de test exploratoire. Cela permet de suivre le temps alloué au test exploratoire, ainsi que la couverture prévue, avec les autres efforts de test.

Le test exploratoire étant peu reproductible, des problèmes peuvent survenir lorsqu'il faut se souvenir des étapes nécessaires pour reproduire une défaillance. Certaines organisations utilisent les fonctionnalités de capture/rejeu d'un outil d'automatisation des tests pour enregistrer les étapes suivies par le testeur. Cela apporte un enregistrement complet de toutes les activités suivies lors de la session de test exploratoire (ou lors de toute autre session de test basé sur l'expérience). Rechercher dans les petits détails pour trouver la cause réelle d'une défaillance peut être fastidieux mais il y a au moins un enregistrement de toutes les étapes suivies.

Couverture
Des chartes peuvent être créées pour spécifier les tâches, les objectifs et les livrables. Des sessions exploratoires sont alors planifiées pour atteindre ces objectifs. La charte peut aussi préciser à quel endroit doit être concentré l'effort de test, ce qui fait ou ne fait pas partie de la session de test, et

quelles ressources devraient être engagées pour accomplir les tests prévus. Une session peut aussi se concentrer sur un type particulier de défauts et sur d'autres domaines pouvant être sources de problèmes et que l'on peut traiter sans utiliser des scripts formels de test.

Types de Défauts
Les principaux types de défauts trouvés avec le test exploratoire sont liés à des problèmes de scenarii non détectés lors des tests fonctionnels basés sur des scripts, à des problèmes se produisant entre des valeurs limites, et à des problèmes liés à des « workflows ». Des problèmes de performance et de sécurité sont aussi parfois découverts lors du test exploratoire.

3.4.4 Appliquer la meilleure technique

Les techniques basées sur les défauts et sur l'expérience nécessitent la mise en œuvre d'une connaissance des défauts et d'une expérience dans le domaine du test pour cibler le test afin d'augmenter le taux de détection des défauts. Elles vont du « test rapide » dans lequel le testeur n'a pas formellement préparé les activités à réaliser, aux sessions documentées, en passant par des activités simplement planifiées. En général, elles sont toujours utiles, en particulier dans les contextes suivants:

- Aucune spécification disponible
- Le système sous test est très peu documenté
- Manque de temps consacré à la conception et à la création de tests détaillés
- Testeurs disposant d'une forte expérience dans le domaine et/ou la technologie
- Objectif de maximiser la couverture des tests avec des tests complémentaires aux tests documentés
- Nécessité d'analyser des défaillances opérationnelles

Les techniques basées sur les défauts et sur l'expérience sont également utiles lorsqu'elles sont utilisées avec des techniques basées sur les spécifications, car elles couvrent les trous dans la couverture de test dus aux faiblesses typiques de ces techniques. Comme avec les techniques basées sur les spécifications, il n'existe pas une seule technique parfaite pour toutes les situations. Il est important pour l'Analyste de Test de comprendre les avantages et les inconvénients de chaque technique et d'être capable de sélectionner la meilleure technique ou la meilleure combinaison de techniques pour une situation donnée, en prenant en compte le type de projet, le planning, l'accès à l'information, les compétences du testeur et tout autre facteur pouvant influencer la sélection.

4. Tester les Caractéristiques du logiciel - 120 mn.

Mots clé

test d'accessibilité, test d'exactitude, attractivité, évaluation heuristique, test d'interopérabilité, apprentissage, opérabilité, test d'aptitude à l'usage, inventaire des mesures de rentabilité (SUMI), intelligibilité, test d'utilisabilité, WAMMI

Objectifs d'apprentissage pour Tester les Caractéristiques du logiciel

4.2 Caractéristiques Qualité pour les tests Métier

AT-4.2.1 (K2) Expliquer avec des exemples, quelles techniques de test sont adaptées aux tests d'exactitude, d'aptitude à l'usage, d'interopérabilité et quelles sont les critères de choix.

AT-4.2.2 (K2) Pour les caractéristiques d'exactitude, d'aptitude à l'usage et d'interopérabilité, identifier les principaux défauts à rechercher

AT-4.2.3 (K2) Pour les caractéristiques d'exactitude, d'aptitude à l'usage et d'interopérabilité, déterminer à quel moment les caractéristiques devraient être testées dans le cycle de vie

AT-4.2.4 (K4) Pour un contexte de projet donné, identifier les approches qui seraient adaptées à la fois pour vérifier et valider l'implémentation des exigences d'utilisabilité et pour vérifier la satisfaction des attentes des utilisateurs

4.1 Introduction

Alors que le chapitre précèdent décrivait les techniques particulières disponibles pour le testeur, ce chapitre aborde l'application de ces techniques en évaluant les principales caractéristiques utilisées pour décrire la qualité des applications logicielles ou des systèmes.

Ce syllabus aborde les caractéristiques de la qualité pouvant être évaluées par un Analyste de Test. Les attributs à évaluer par l'Analyste Technique de Test sont abordés dans le syllabus de Niveau Avancé Analyste Technique de Test. La description des caractéristiques de la qualité du produit se base sur l'ISO 9126, utilisé comme guide. D'autres standards, comme l'ISO 25000 [ISO25000] (qui a englobé l'ISO 9126) peuvent également être utilisés. Les caractéristiques de la qualité décrits par ISO sont divisées en caractéristiques de la qualité du produit (attributs), chacune pouvant avoir des sous-caractéristiques (sous-attributs). Elles sont rapportées dans le tableau ci-dessous, avec l'identification du syllabus dans lequel elles sont abordées(Analyste de Test ou Analyste Technique de Test):

Caractéristiques	**Sous-caractéristiques**	**Analyste de Test**	**Analyste Technique de Test**
Fonctionnalité	Exactitude, aptitude à l'usage, interopérabilité, conformité	X	
	Sécurité		X
Fiabilité	Maturité (robustesse), tolérance aux fautes, récupérabilité, conformité		X
Utilisabilité	Intelligibilité, apprentissage, opérabilité, attractivité, conformité	X	
Rendement	Performance (comportement dans le temps), utilisation des ressources, conformité		X
Maintenabilité	Analysabilité, variabilité, stabilité, testabilité, conformité		X
Portabilité	Adaptabilité, installabilité, coexistence, remplaçabilité, conformité		X

L'Analyste de Test devrait se concentrer sur les caractéristiques de qualité logicielle suivantes : fonctionnalité et utilisabilité. Le test d'accessibilité devrait également être mené par l'Analyste de Test. Bien que non listée comme une sous-caractéristique, on considère souvent que l'accessibilité fait partie du test d'utilisabilité. On considère en général que le test des autres caractéristiques de la qualité relève de la responsabilité de l'Analyste Technique de Test. Bien que cette répartition du travail puisse varier dans différentes organisations, c'est celle qui est suivie dans ces syllabi de l'ISTQB.

La sous-caractéristique de conformité apparaît pour chacune des caractéristiques de la qualité. Dans le cas de certains environnements, à sécurité critique ou régulés, chaque caractéristique de la qualité peut avoir à satisfaire à des standards et régulations (p. ex., la conformité de fonctionnalité peut indiquer que la fonctionnalité satisfait à un standard particulier tel que l'utilisation d'un protocole de communication particulier devant permettre d'envoyer/recevoir des données à partir d'une puce). Comme ces standards peuvent fortement varier en fonction de l'industrie, ils ne seront pas discutés en détail ici. Si l'Analyste de Test travaille dans un environnement concerné par des exigences de conformité, il est important de comprendre ces exigences et de s'assurer qu'à la fois le test et la documentation du test satisferont à ces exigences de conformité.

Pour toutes les caractéristiques et sous-caractéristiques de la qualité discutées dans cette section, les risques typiques doivent être reconnus afin de pouvoir bâtir et documenter une stratégie de test

appropriée. Le test de caractéristique de la qualité nécessite une attention particulière au déroulement du cycle de vie, aux outils nécessaires, à la disponibilité du logiciel et de la documentation, ainsi qu'à l'expertise technique. Sans la mise en place d'une stratégie pour traiter chaque caractéristique avec ses besoins particuliers en test, le testeur risque de ne pas disposer d'assez de temps dans le planning pour la gestion, la préparation et l'exécution des tests. Une partie des tests, (p. ex., le test d'utilisabilité), peut nécessiter l'allocation de ressources humaines particulières, une extension du planning, des ateliers particuliers, des outils spécifiques, des compétences spécialisées en test et, dans la plupart des cas, un temps très important. Dans certains cas, le test d'utilisabilité peut être réalisé par un groupe spécifique à l'utilisabilité, sur la base de l'expérience d'utilisateurs ou d'experts.

Le test des caractéristiques et sous-caractéristiques de la qualité doit être intégré au planning global de test, avec l'affectation des ressources nécessaires à l'effort. Chacun de ces domaines a des besoins spécifiques, cible des problèmes particuliers et peut se produire à différents moments durant le cycle de vie de développement logiciel, comme cela sera discuté dans la section suivante.

Même si l'Analyste de Test peut ne pas être responsable des caractéristiques de la qualité demandant une approche plus technique, il est important que l'Analyste de Test soit au courant des autres caractéristiques et comprenne les domaines de recouvrement pour le test. Par exemple, un produit défaillant lors des tests de performance sera probablement défaillant aussi lors des tests d'utilisabilité s'il est trop lent pour que l'utilisateur puisse s'en servir efficacement. De même, un produit avec des problèmes d'interopérabilité avec certains composants n'est probablement pas prêt à subir des tests de portabilité car cela aura tendance à masquer les problèmes les plus basiques lorsque l'environnement est modifié.

4.2 Caractéristiques de la Qualité pour les tests du Métier

Le test fonctionnel est une préoccupation première de l'Analyste de Test. Le test fonctionnel est centré sur le « quoi », sur ce que le produit fait. Les bases de test pour le test fonctionnel sont généralement un document d'exigences ou de spécification, une expertise-métier spécifique ou un besoin induit.
Les test fonctionnels varient en fonction du niveau de test auquel ils sont menés et peuvent aussi être influencés par le cycle de vie de développement logiciel. Par exemple, un test fonctionnel mené pendant les tests d'intégration va tester la fonctionnalité d'interfaçage de modules qui implémentent une fonction unique définie. Au niveau système, les tests fonctionnels incluent le test de la fonctionnalité de l'application dans son ensemble. Pour les systèmes de systèmes, le test fonctionnel ciblera principalement le test bout-en-bout parmi les systèmes interfacés. Dans un environnement Agile, le test fonctionnel se limite en général à la fonctionnalité rendue disponible pour une certaine itération ou pour un « sprint » en particulier même si le test de régression d'une itération peut couvrir toutes les fonctionnalités livrées.

Une grande variété de techniques de test sont utilisées lors du test fonctionnel (voir chapitre 3).
Le test fonctionnel peut être réalisé par un testeur dédié, un expert métier, ou un développeur (en général au niveau des composants).

En complément du test fonctionnel abordé dans cette section, il existe deux autres caractéristiques de la qualité, du ressort de l'Analyste de Test, qui sont considérée comme non-fonctionnelles (se concentrant sur « comment » le produit fournit la fonctionnalité). Ces deux attributs non-fonctionnels sont l'utilisabilité et l'accessibilité.

Les caractéristiques de la qualité suivantes sont abordées dans cette section:

- Sous-caractéristiques fonctionnelles de la qualité
  - Exactitude
  - Aptitude à l'usage
  - Interopérabilité

- Caractéristiques non-fonctionnelles de la qualité
  - Utilisabilité
  - Accessibilité

4.2.1 Test d'exactitude

L'exactitude fonctionnelle implique le test de l'adhésion de l'application à des exigences spécifiées ou implicites et peut aussi inclure l'exactitude computationnelle. Le test d'exactitude utilise beaucoup des techniques de test expliquées dans le chapitre 3, et utilise souvent la spécification ou le système historique comme oracle de test. Le test d'exactitude peut être réalisé à tout moment dans le cycle de vie et cible la mauvaise gestion de données ou de situations.

4.2.2 Test d'aptitude à l'usage

Le test d'aptitude à l'usage implique l'évaluation et la validation de la justesse d'un ensemble de fonctions pour les tâches prévues spécifiées. Ce test peut se baser sur les cas d'utilisation. Le test d'aptitude à l'usage est en général mené pendant le test système mais peut aussi être mené ultérieurement lors du test d'intégration. Les défauts découverts durant ces tests sont des indications de l'incapacité du système à répondre d'une façon acceptable aux besoins des utilisateurs.

4.2.3 Test d'interopérabilité

Le test d'interopérabilité teste la faculté de plusieurs systèmes à échanger de l'information et à l'utiliser ensuite. Le test doit couvrir tous les environnements ciblés (y compris les variations dans le matériel, le logiciel, le middleware, le système d'exploitation, etc.) pour garantir que les données échangées fonctionneront correctement. Dans certains cas le test d'interopérabilité peut se limiter à un groupe d'environnements représentatifs. Spécifier des tests d'interopérabilité nécessite l'identification des différentes combinaisons possibles pour les environnements ciblés, en gérant leur configuration et leur disponibilité pour l'équipe de test. Ces environnements sont ensuite testés à partir d'une sélection de cas de test fonctionnels qui solliciteront les différents points d'échange de données présents dans l'environnement.

L'interopérabilité correspond à la façon avec laquelle différents systèmes logiciels interagissent les uns avec les autres. Un logiciel ayant de bonnes caractéristiques d'interopérabilité pourra être intégré avec un grand nombre d'autres systèmes sans nécessiter de changements importants. Le nombre de changements et l'effort nécessaire à la réalisation de ces changements peuvent être utilisés comme mesure de l'interopérabilité.

Le test de l'interopérabilité logicielle peut, par exemple, se concentrer sur les fonctionnalités de conception suivantes:

- Utilisation de standards de communication largement répandus dans l'industrie, tels que XML
- Capacité à détecter automatiquement les besoins en communication des systèmes avec lesquels il y a des interactions et à s'adapter en conséquence

Le test d'interopérabilité peut être particulièrement significatif pour les organisations développant des composants sur étagère (« Commercial Off The Shelf » (COTS)) et des outils, ainsi que pour les organisations développant des systèmes de systèmes.

Ce type de test est réalisé durant les tests d'intégration de composants et les tests système, en se concentrant sur l'interaction du système avec son environnement. Au niveau intégration système, ce type de test est mené pour déterminer dans quelle mesure le système dans son ensemble interagit avec les autres systèmes. Comme des systèmes peuvent interagir à différents niveaux, l'Analyste de Test doit comprendre ces interactions et être capable de créer les conditions permettant de produire les différentes interactions. Par exemple, si deux systèmes échangent des données l'Analyste de Test doit être en mesure de créer les données nécessaires ainsi que les transactions requises pour produire l'échange de données. Il est important de se souvenir que toutes les interactions ne sont pas

forcément clairement spécifiées dans les documents d'exigences. En réalité, beaucoup de ces interactions ne seront définies que dans l'architecture du système et dans les documents de conception. L'Analyste de Test doit être capable d'examiner ces documents et être préparé à le faire afin d'identifier les flux d'informations échangées entre systèmes et entre le système et son environnement, pour s'assurer qu'ils seront tous testés. Des techniques comme les tables de décisions, les diagrammes de transition d'état, les cas d'utilisation et le test combinatoire sont toutes applicables au test d'interopérabilité. Comme défauts typiquement découverts, on peut citer des échanges de données incorrects entre les composants qui interagissent.

4.2.4 Test d'Utilisabilité

Il est important de comprendre pourquoi les utilisateurs peuvent avoir des difficultés à utiliser le système. Pour arriver à le comprendre, il est, en premier lieu, nécessaire de comprendre que le terme « utilisateur » peut s'appliquer à beaucoup de personnes différentes, allant d'experts informatiques, à des enfants, en passant par des personnes ayant un handicap.

Certaines institutions nationales (p. ex., l'institut national royal pour les malvoyants (British Royal National Institute for the Blind)), recommandent des pages web accessibles pour les personnes ayant un handicap, les aveugles, les personnes à mobilité réduite, les malvoyants, les malentendants, ou les personnes ayant un handicap cognitif. Vérifier que les applications et les sites web soient utilisables pour ces personnes peut aussi améliorer l'utilisabilité pour toute autre personne. L'accessibilité est abordée plus en détail ci-après.

Le test d'utilisabilité teste l'aisance avec laquelle les utilisateurs peuvent utiliser ou apprendre à utiliser le système pour atteindre un but spécifique dans un contexte particulier. Le test d'utilisabilité permet de mesurer les caractéristiques suivantes:

- Rentabilité – capacité du produit à permettre aux utilisateurs de dépenser la quantité de ressources appropriée par rapport à l'efficacité atteinte dans un contexte d'usage particulier
- Efficacité – capacité du produit logiciel à permettre aux utilisateurs d'atteindre des objectifs spécifiques avec exactitude et complétude dans un contexte d'usage particulier
- Satisfaction – capacité du produit logiciel à satisfaire les utilisateurs dans un contexte d'usage particulier

Parmi les attributs pouvant être mesurés on peut citer:

- Intelligibilité – attributs du logiciel ayant un effet sur l'effort demandé à l'utilisateur pour repérer les concepts logiques et leur mise en pratique
- Apprentissage - attributs du logiciel ayant un effet sur l'effort demandé à l'utilisateur pour apprendre l'application
- Opérabilité - attributs du logiciel ayant un effet sur l'effort demandé à l'utilisateur pour accomplir des tâches de façon efficace et rentable
- Attractivité – capacité du logiciel à être apprécié par l'utilisateur

Le test d'utilisabilité est en général mené en deux étapes:

- Test d'utilisabilité de forme – test qui est mené de façon itérative lors des étapes de conception et de prototypage pour aider à orienter (ou "former") la conception en identifiant des défauts de conception d'utilisabilité
- Test d'utilisabilité de fond – test qui est mené après l'implémentation pour mesurer l'utilisabilité et identifier les problèmes avec le composant ou système complet

Les compétences du testeur d'utilisabilité devraient inclure de l'expertise ou de la connaissance dans les domaines suivants:

- Sociologie
- Psychologie

- Conformité à des standards nationaux (incluant des standards d'accessibilité)
- Ergonomie

4.2.4.1 Mener les tests d'utilisabilité

La validation de l'implémentation réelle devrait se faire dans des conditions aussi proches que possibles de celles dans lesquelles le système sera utilisé. Cela peut nécessiter le montage d'un atelier d'utilisabilité avec des caméras, des bureaux factices, des groupes de revue, des utilisateurs, etc., afin que l'équipe de développement puisse observer les effets du système sur de vraies personnes. Le test d'utilisabilité formel requiert souvent un travail de préparation des "utilisateurs" (ceux-ci peuvent être de vrais utilisateurs ou des représentants des utilisateurs) en leur fournissant soit des scripts, soit des instructions à suivre. Une autre façon de tester, plus libre, consiste à laisser l'utilisateur essayer le logiciel afin que des observateurs puissent déterminer la facilité ou la difficulté avec laquelle l'utilisateur parvient à réaliser ses tâches.

De nombreux tests d'utilisabilité peuvent être exécutés par l'Analyste de Test parmi d'autres tests, par exemple lors des tests systèmes fonctionnels. Pour arriver à une approche cohérente afin de détecter et rapporter des défauts d'utilisabilité à toutes les étapes du cycle de vie, des consignes d'utilisabilité peuvent être utiles. Sans ces consignes relatives à l'utilisabilité, il peut être difficile de déterminer ce qui n'est pas « acceptable » d'un point de vue utilisabilité. Par exemple, est-il raisonnable pour un utilisateur de devoir effectuer 10 clics de souris pour se connecter à une application ? Sans consignes spécifiques, l'Analyste de Test peut se retrouver dans une position difficile consistant à défendre des rapports de défauts que le développeur veut clôturer parce que le système fonctionne « tel qu'il a été conçu ». Il est très important de disposer de spécifications d'utilisabilité vérifiables, définies dans les exigences, et également d'avoir un ensemble de consignes d'utilisabilité applicables à tous les projets d'un même type. Les consignes devraient inclure des éléments comme l'accessibilité des instructions, la clarté des messages, le nombre de clics pour réaliser une action, les messages d'erreur, les indicateurs de traitement (indication pour l'utilisateur que le système travaille et ne peut pas accepter d'autres entrées au même moment), consignes sur l'apparence d'un écran, l'utilisation de couleurs et de sons et tout autre facteur ayant une influence sur le ressenti de l'utilisateur.

4.2.4.2 Spécification des tests d'Utilisabilité

Les principales techniques pour les tests d'utilisabilité sont:

- Inspecter, évaluer ou revoir
- Interagir dynamiquement avec des prototypes
- Vérifier et valider l'implémentation
- Organiser des sondages et des questionnaires

Inspecter, évaluer ou revoir

L'inspection ou la revue des spécifications d'exigences et de la conception d'un point de vue utilisabilité, en impliquant davantage l'utilisateur, peut permettre efficacement de découvrir tôt des problèmes. Une évaluation heuristique (inspection systématique de la conception d'une interface utilisateur par rapport à son utilisabilité) peut être utilisée pour trouver, dans la conception, des problèmes d'utilisabilité qui pourront être pris en compte dans un processus itératif de conception. Cela demande à ce qu'un petit groupe d'évaluateurs examine l'interface et juge de son respect des principes reconnus d'utilisabilité (les "heuristiques"). Les revues sont plus efficaces lorsque l'interface utilisateur est plus visible. Par exemple, des captures d'écrans sont en général plus faciles à comprendre et à interpréter que la description textuelle de la fonctionnalité offerte par un écran particulier. La possibilité de visualiser est importante pour une bonne revue d'utilisabilité de la documentation.

Interagir dynamiquement avec des prototypes

Quand des prototypes sont développés, l'Analyste de Test devrait travailler avec eux et aider les développeurs à les faire évoluer en intégrant les retours des utilisateurs dans la conception. De cette

façon, les prototypes peuvent être raffinés et l'utilisateur peut obtenir une vision plus réaliste de l'image et de l'utilisation du produit final.

Vérifier et valider l'implémentation réelle
Lorsque les exigences précisent les caractéristiques d'utilisabilité du logiciel (p. ex., le nombre de clics de souris pour effectuer une action spécifique), des cas de test devraient être créés pour vérifier que l'implémentation du logiciel a pris en compte ces caractéristiques.

Pour effectuer la validation de l'implémentation réelle, des tests spécifiés pour le test système fonctionnel peuvent être développés comme scénarii de test d'utilisabilité. Ces scénarios de test mesurent des caractéristiques spécifiques d'utilisabilité, telles l'apprentissage ou l'opérabilité, plutôt que des sorties fonctionnelles.

Les scénarios de test pour l'utilisabilité peuvent être développés pour tester particulièrement la syntaxe et la sémantique. La syntaxe est la structure ou la grammaire de l'interface (p. ex., ce qui peut être entré dans un champ de saisie) alors que la sémantique décrit la signification et l'objet (p. ex., des messages et sorties raisonnables fournis à l'utilisateur) de l'interface.

Les techniques boîtes noires (par exemple celles décrites en Section 3.2), en particulier celle des cas d'utilisation qui peut être définie de façon textuelle ou avec UML (Unified Modeling Language), sont parfois employées dans le test d'utilisabilité.

Les scenarii de test d'utilisabilité doivent aussi comprendre des instructions pour l'utilisateur, du temps alloué avant et après les tests pour donner des instructions, recevoir des retours et se mettre d'accord sur la façon de mener les sessions de test. Ce protocole comprend la description de la façon d'exécuter le test, les durées, la prise de notes et l'enregistrement des sessions, ainsi que les méthodes d'interrogation et d'enquête à utiliser.

Gérer des enquêtes et des questionnaires
Les techniques d'enquête et de questionnaire peuvent être appliquées pour collecter des observations et des retours relatifs au comportement de l'utilisateur avec le système. Des enquêtes standard et publiques telles que l'Inventaire des mesures de rentabilité (SUMI) (« Software Usability Measurement Inventory ») et WAMMI (« Website Analysis and MeasureMent Inventory ») permettent de se positionner par rapport à une base de mesures d'utilisabilité. De plus, comme l'inventaire des mesures de rentabilité (SUMI) apporte des mesures concrètes de l'utilisabilité, cela peut fournir un ensemble complet de critère de complétude/acceptation.

4.2.5 Test d'accessibilité

Il est important de se préoccuper de l'accessibilité du logiciel pour toutes les personnes ayant un besoin particulier, ou des restrictions dans l'usage de celui-ci. Cela comprend les personnes ayant un handicap. Le test d'accessibilité devrait s'appuyer sur les standards adéquats, tel que le « Web Content Accessibilité Guidelines », la législation comme le « Disability Discrimination Acts » (Royaume-Uni, Australie), ou encore comme la « Section 508 » (Etats-Unis). L'accessibilité, de même que l'utilisabilité, doivent être prise en compte pendant les phases de conception. Le test a lieu au niveau intégration et se poursuit aux niveaux système et acceptation. Des défauts sont en général mis en évidence lorsque le logiciel ne parvient pas à satisfaire les régulations et standards définis pour le logiciel.

5. Revues - 165 mn.

Mots clé

aucun

Objectifs d'apprentissage pour Revues

5.1 Introduction

AT-5.1.1 (K2) Expliquer pourquoi la préparation de revue est importante pour l'Analyste de Test

5.2 Utiliser des check-lists pour les Revues

AT-5.2.1 (K4) Analyser un cas d'utilisation ou une interface utilisateur et identifier les problèmes selon les informations de check-list fournies dans le syllabus

AT-5.2.2 (K4) Analyser une spécification d'exigence ou une «user story» et identifier les problèmes selon les informations de check-list fournies dans le syllabus

5.1 Introduction

Pour être réalisé avec succès, un processus de revue nécessite une gestion, de la participation et un suivi. Les Analystes de Test doivent participer activement au processus de revue, en apportant leur vision unique. Ils devraient suivre une formation spécifique aux revues pour mieux comprendre leurs rôles respectifs dans tout processus de revue. Tous les participants à une revue doivent s'engager à contribuer à permettre les bénéfices d'une revue bien menée. Bien réalisées, les revues peuvent constituer à elles seules le contributeur le plus important et le plus rentable à la qualité d'ensemble délivrée.

Quel que soit le type de revue en cours, l'Analyste de Test doit consacrer le temps nécessaire à la préparation. Cela comprend le temps pour revoir le livrable, le temps pour vérifier la cohérence des documents référencés, et le temps pour déterminer ce qui pourrait manquer au livrable. Sans un temps de préparation adéquat, l'Analyste de Test pourrait être réduit à simplement éditer ce qui est déjà dans le document, plutôt que de participer à une revue efficace maximisant l'utilisation du temps de l'équipe de revue et fournissant les meilleurs retours possibles. Une bonne revue inclut la compréhension de ce qui est écrit, l'identification de ce qui manque et la vérification de la cohérence du produit décrit avec d'autres produits pouvant soit déjà être développés, soit être en développement. Par exemple, lors de la revue d'un plan de test de niveau d'intégration, l'Analyste de Test doit aussi prendre en compte les éléments qui sont en train d'être intégrés. Quelles sont les conditions à satisfaire pour qu'ils soient prêts pour l'intégration ? Existe-t-il des dépendances qui doivent être documentées ? Des données sont-elles disponibles pour tester les points d'intégration ? Une revue ne se limite pas au livrable qui est revu, elle doit aussi prendre en compte les interactions entre cet élément et les autres éléments du système.

L'auteur du livrable revu peut facilement se sentir critiqué. L'Analyste de Test devrait s'assurer d'aborder tout commentaire de revue, dans un esprit de coopération avec l'auteur afin de créer le meilleur produit possible. En utilisant cette approche, les commentaires seront formulés de façon constructive et seront dirigés vers le livrable et non vers son auteur. Par exemple, si une formulation est ambigüe, il est préférable de dire "Je ne comprends pas ce que je devrai tester pour vérifier que cette exigence ait été implémentée correctement. Pouvez-vous m'aider à le comprendre ?" Plutôt que "Cette exigence est ambiguë et personne ne pourra la comprendre." Le travail de l'Analyste de Test dans une revue est de s'assurer que l'information fournie dans le livrable sera suffisante pour permettre l'activité de test. Si l'information n'est pas là, n'est pas claire, ou ne fournit pas un niveau de détail suffisant, alors il y a probablement un défaut à corriger par l'auteur. En maintenant une approche constructive positive, plutôt qu'une approche critique négative, les commentaires seront mieux reçus et les réunions seront plus productives.

5.2 Utiliser des check-lists pour les Revues

Des check-lists sont utilisées durant les revues pour rappeler aux participants de vérifier certains points lors des revues. Les check-lists peuvent aussi aider à dépersonnaliser la revue, (p. ex., "Cette check-list est celle que nous utilisons pour toutes les revues, nous ne visons pas votre produit en particulier "). Les check-lists peuvent être génériques et utilisées pour toutes les revues ou peuvent être focalisées sur des caractéristiques de qualité, des domaines ou des types de documents spécifiques. Par exemple, une check-list générique pourrait vérifier les propriétés générales d'un document telles qu'avoir un identifiant unique, ne pas avoir de références erronées, être correctement formaté et être homogène. Une check-list spécifique pour un document d'exigences pourrait contenir des vérifications sur le bon usage des termes "doit" et "devrait", des vérifications pour la testabilité de chaque exigence formulée et ainsi de suite. Le format des exigences peut aussi indiquer le type de check-list à utiliser. Un document d'exigences écrit dans un format textuel aura des critères de revue différents de ceux d'un document basé sur des diagrammes.

Les check-lists peuvent aussi être orientées par rapport aux compétences d'un programmeur/architecte ou d'un testeur. Dans le cas de l'Analyste de Test, la check-list des compétences en test sera la plus appropriée. Ces check-lists peuvent inclure les éléments mentionnés ci-après.

Des check-lists utilisées pour les exigences, les cas d'utilisation et les «user stories» ont en général un focus diffèrent de celles utilisées pour le code ou l'architecture. Ces check-lists orientées exigences pourraientinclure les éléments suivants:

- Testabilité de chaque exigence
- Critère d'acceptation pour chaque exigence
- Disponibilité d'un cas utilisation lorsque cela est applicable
- Identification unique de chaque exigence/cas d'utilisation/ « user story»
- Gestion de version de chaque exigence/cas d'utilisation/ « user story»
- Traçabilité pour chaque exigence à partir des exigences du métier/marketing
- Traçabilité entre exigences et cas d'utilisation

Les éléments ci-dessus ne sont mentionnés qu'à titre d'exemple. Il est important de se souvenir que si une exigence n'est pas testable, c'est-à-dire qu'elle est définie de telle façon que l'Analyste de Test ne peut pas déterminer comment la tester, alors il y a un défaut dans cette exigence. Par exemple, une exigence comme "Le logiciel devrait être très agréable à utiliser" n'est pas testable. Comment l'Analyste de Test pourrait-il déterminer si le logiciel est agréable à utiliser ou même très agréable à utiliser? Si, au lieu de cela, l'exigence disait: "Le logiciel doit satisfaire les standards d'utilisabilité mentionnés dans le document relatif aux standards d'utilisabilité", et si le document en question existe vraiment, alors c'est une exigence testable. C'est également une exigence transverse puisque elle s'applique à chaque élément de l'interface. Dans ce cas, cette unique exigence pourrait facilement engendrer de nombreux cas de test pour une application complexe. La traçabilité à partir de cette exigence, ou peut-être à partir du document relatif aux standards d'utilisabilité, vers les cas de test, est aussi critique car si la spécification d'utilisabilité doit changer, alors tous les cas de test devront également être revus et mis à jour en conséquence.

Une exigence ne sera pas non plus testable si le testeur n'est pas en mesure de déterminer si le test est passé avec succès ou bien est en échec ; ou encore s'il n'est pas capable de construire un test qui pourra passer avec succès ou échouer. Par exemple, "Le système doit être disponible 100% du temps, 24 heures par jour, 7 jours par semaine, 365 (ou 366) jours par an" n'est pas testable.
.
Une simple check-list pour la revue des cas d'utilisation pourrait inclure les questions suivantes:

- Le principal chemin (scenario) est-il clairement défini?
- Les chemins (scenarii) alternatifs sont-ils tous identifiés, complets, et avec une gestion d'erreur?
- Les messages de l'interface utilisateur sont-ils définis?
- N'y-a-t-il qu'un chemin principal (cela devrait être le cas, sinon, il y a plusieurs cas d'utilisation) ?
- Chaque chemin est-il testable?

Une simple check-list pour l'utilisabilité de l'interface utilisateur d'une application pourrait inclure:

- Chaque champ, et sa fonction, sont-ils définis?
- Les messages d'erreur sont-ils tous définis?
- Les messages pour l'utilisateur sont-ils tous définis et valables?
- L'ordre de tabulation des champs est-il défini ?
- Y-a-t-il des raccourcis clavier en alternative à la souris?
- Des raccourcis clavier ont-ils été définis pour l'utilisateur (p. ex., copier et coller)?

- Y-a-t-il des dépendances entre les champs (comme une certaine date devant être postérieure à une autre)?
- Y-a-t-il une charte graphique?
- La charte graphique est-elle en accord avec les exigences ?
- Un indicateur de traitement est-il affiché à l'utilisateur lorsque le système travaille?
- L'écran satisfait-il la contrainte relative au nombre de clics de souris (si elle est définie)?
- La navigation se fait-elle logiquement pour l'utilisateur conformément au cas d'utilisation?
- L'écran satisfait-il à toutes les exigences relatives à l'apprentissage?
- Une aide textuelle est-elle disponible pour l'utilisateur?
- Un affichage dynamique de texte, sur certaines zones survolées par la souris, est-il prévu ?
- Cela sera-t-il jugé agréable par l'utilisateur (évaluation subjective)?
- L'utilisation de couleur est-elle cohérente avec les autres applications et avec les standards de l'organisation ?
- Les effets de son sont-ils utilisés de façon appropriée et sont-ils configurables?
- L'écran répond-il aux exigences de localisation?
- L'utilisateur peut-il déterminer de façon intuitive ce qu'il a à faire (intelligibilité) (évaluation subjective)?
- L'utilisateur sera-t-il en mesure de se souvenir de ce qu'il doit faire (apprentissage) (évaluation subjective)?

Dans un projet Agile, les exigences prennent en général la forme de «user stories». Ces «user stories» représentent de petites unités de fonctionnalités démontrables. A la différence d'un cas d'utilisation correspondant à une transaction utilisateur et traversant plusieurs domaines fonctionnels, une «user story» est plus isolée et est généralement bornée par rapport à son temps de développement. Une check-list pour une «user story» pourrait inclure :

- La «user story» est-elle adaptée à l'itération et au « sprint » visés?
- Les critères d'acceptation sont-ils définis et testables?
- La fonctionnalité est-elle clairement définie?
- Y-a-t-il des dépendances entre cette «user story» et d'autres «user stories» ?
- La «user story» est-elle priorisée?
- La «user story» contient-elle un élément de fonctionnalité?

Si la «user story» définit une nouvelle interface, il faudra bien sûr utiliser une check-list générique pour les «user stories» (comme celle ci-dessus) et une check-list spécifique à une interface utilisateur.

Une check-list peut être adaptée sur la base des éléments suivants:

- Organisation (p. ex., prendre en compte les politiques, les standards et les conventions de la société)
- Effort de projet/développement (p. ex., standards techniques, risques)
- Objet revu (p. ex., des revues de code pourront être adaptées à des langages spécifiques)

De bonnes check-lists permettront de trouver des problèmes et aideront également à démarrer des discussions relatives à d'autres éléments n'ayant pas forcément été spécifiquement référencés dans la check-list. Combiner des check-lists est une bonne façon de s'assurer que la revue atteint la plus grande qualité de produit possible. Utiliser des check-lists standards, comme celles référencées dans le syllabus du Niveau Fondation, et développer des check-lists spécifiques à l'organisation, telles que celles montrées ci-dessus, aideront l'Analyste de Test à être efficace dans les revues.

Pour plus d'information sur les revues et les inspections, voir [Gilb93] et [Wiegers03].

6. Gestion des Défauts – 120 mn.

Mots clé

Taxonomie de défauts, maîtrise de phase, analyse des causes racines

Objectifs d'apprentissage pour Gestion des Défauts

6.2 Quand un Défaut peut-il être Détecté?

AT-6.2.1 (K2) Expliquer comment la maîtrise de phase peut réduire les coûts

6.3 Champs d'un Rapport de Défaut

AT-6.3.1 (K2) Expliquer quelle information peut être nécessaire pour documenter un défaut non-fonctionnel

6.4 Classification des Défauts

AT-6.4.1 (K4) Identifier, collecter et enregistrer les informations nécessaires à la classification d'un défaut donné

6.5 Analyse des Causes Racines

AT-6.5.1 (K2) Expliquer l'objectif de l'analyse des causes racines

6.1 Introduction

Les Analystes de Test évaluent le comportement du système en termes des besoins du métier et des besoins fonctionnels, (p. ex., l'utilisateur saurait-il quoi faire s'il est confronté à tel message ou comportement ?). En comparant le résultat obtenu avec le résultat attendu, l'Analyste de Test détermine si le système se comporte correctement. Une anomalie (également appelée incident) est un évènement non-attendu qui nécessite une investigation ultérieure. Une anomalie peut être une défaillance causée par un défaut. Une anomalie peut donner lieu à la création d'un rapport de défaut ou non. Un défaut est un problème réel qui doit être résolu.

6.2 Quand un Défaut peut-il être Détecté?

Un défaut peut être détecté par du test statique, et les symptômes du défaut, la défaillance, peuvent être détectés par du test dynamique. Chaque phase du cycle de vie de développement logiciel devrait fournir des méthodes pour détecter et éliminer les défaillances potentielles. Par exemple, lors de la phase de développement, des revues de code et de conception devraient être mises en œuvre pour détecter des défauts. Pendant le test dynamique, les cas de test permettent de détecter des défaillances.

Plus un défaut est détecté et corrigé tôt, moins le coût de la qualité pour le système dans son ensemble sera élevé. Par exemple, le test statique peut trouver des défauts avant que le test dynamique ne soit possible. C'est l'une des raisons pour laquelle le test statique est une approche rentable pour produire des logiciels de grande qualité.

Le système de suivi des défauts devrait permettre à l'Analyste de Test d'enregistrer la phase durant laquelle le défaut a été introduit et la phase durant laquelle il a été découvert. Si les deux phases sont les mêmes, alors une parfaite maîtrise de la phase a été atteinte. . Cela signifie que le défaut a été introduit et trouvé dans la même phase et ne s'est pas «échappé » vers une phase ultérieure. On peut citer comme exemple une exigence incorrecte identifiée et corrigée lors de la revue des exigences. Non seulement cela correspond à une mise en œuvre efficace de la revue, mais cela évite également à ce défaut d'engendrer un travail supplémentaire qui le rendrait cher pour la société. Si une exigence incorrecte « échappe » à la revue des exigences et se retrouve implémentée par le développeur, testée par l'Analyste de Test, et détecté par l'utilisateur lors des tests d'acceptation, alors tout le travail fait sur cette exigence aura été du temps perdu (sans préciser que l'utilisateur pourra en plus avoir perdu confiance dans le système)

La maîtrise de phase est une façon efficace de réduire le coût des défauts.

6.3 Champs d'un Rapport de Défaut

Les champs (paramètres) fournis pour un rapport de défaut ont pour objectif de fournir suffisamment d'information pour que le rapport soit utilisable. Un rapport de défaut utilisable est:

- Complet – toute l'information nécessaire se trouve dans le rapport
- Concis – aucune information superflue n'est présente dans le rapport
- Précis – l'information dans le rapport est correcte et précise clairement les résultats attendus et obtenus, de même que les étapes permettant la reproduction du défaut
- Objectif – le rapport est la description factuelle d'un événement écrite de façon professionnelle

L'information enregistrée dans un rapport de défaut devrait se répartir dans différents champs de données. Mieux les champs sont définis, plus il est facile de rapporter des défauts uniques et de

fournir des rapports montrant des tendances ou des synthèses. Lorsqu'un nombre défini d'options sont possibles pour un champ, une liste déroulante avec les différentes valeurs possibles peut permettre de réduire le temps nécessaire à l'enregistrement du défaut. Les listes déroulantes ne sont utiles que si le nombre d'options est limité et si l'utilisateur n'est pas obligé de parcourir une longue liste de valeurs pour trouver la bonne. Différents types de rapports de défauts nécessitent différentes informations et l'outil de gestion des défauts devrait être suffisamment flexible pour faire apparaître les champs correspondant à un type de défaut particulier. Des données doivent être enregistrées dans différents champs avec une validation permettant d'éviter des saisies incorrectes et d'assurer un reporting efficace.

Des rapports de défauts sont écrits pour des défaillances découvertes lors des tests fonctionnels et non-fonctionnels. L'information contenue dans un rapport de défaut devrait toujours offrir une description claire du scénario dans lequel le problème a été détecté, inclure les données et les étapes nécessaires à la reproduction de ce scénario, de même que les résultats attendus et obtenus. Des rapports de défauts non-fonctionnels peuvent nécessiter plus de détails relatifs à l'environnement, à d'autres paramètres de performance (p. ex. la quantité de charge), à l'ordre des étapes et aux résultats attendus. Lors de la documentation d'une défaillance d'utilisabilité, il est important de rappeler ce que l'utilisateur s'attendait à ce que le logiciel fasse. Par exemple, si le standard d'utilisabilité spécifie qu'une opération doit être réalisée en moins de quatre clics de souris, le rapport de défaut devrait expliquer combien de « clicks » ont été nécessaires au-delà du nombre autorisé. Dans les cas où aucun standard n'est disponible et où les exigences ne couvrent pas les aspects non-fonctionnels de la qualité logicielle, le testeur peut utiliser la « bonne personne » pour tester et pour déterminer si l'utilisabilité est acceptable. Dans ce cas, les attentes concernant la "bonne personne" doivent être précisées dans le rapport de défaut. Comme les exigences non-fonctionnelles manquent parfois dans la documentation des exigences, la documentation des défaillances non-fonctionnelles présente des difficultés pour le testeur lorsqu'il doit documenter le comportement obtenu par rapport au comportement attendu.

Bien que le principal objectif, lors de l'écriture d'un rapport de défaut, soit d'obtenir une correction du problème, l'information relative au défaut doit aussi être fournie pour permettre une classification précise, une analyse de risque et l'amélioration de processus.

6.4 Classification des Défauts

Un rapport de défaut peut avoir différents niveaux de classification tout au long de son cycle de vie. Une bonne classification des défauts fait partie intégrante d'un bon reporting des défauts. Des classifications sont utilisées pour regrouper des défauts, évaluer la rentabilité du test, évaluer la rentabilité du cycle de vie de développement et déterminer les tendances intéressantes.

On peut citer comme classification pour des défauts récemment identifiés:

- Activité du projet durant laquelle le défaut a été détecté – (p. ex., revue, audit, inspection, codage, test)
- Phase du projet dans laquelle le défaut a été introduit (si elle est connue) – (p. ex., exigences, conception, conception détaillés, codage)
- Phase du projet dans laquelle le défaut a été détecté – (p. ex., exigences, conception, conception détaillés, codage, revue de code, test unitaire, test d'intégration, test système, test d'acceptation)
- Cause probable du défaut – (p. ex., exigences, conception, interface, code, données)
- Répétabilité – (p. ex., unique, intermittent, reproductible)
- Symptôme – (p. ex., crash, interruption, erreur d'interface, erreur système, performance)

Une fois que le défaut a été analysé, plusieurs classifications peuvent être possibles:

- Cause racine - l'erreur commise qui a donné lieu à un défaut, (p. ex., un processus, une erreur de codage, une erreur de l'utilisateur, une erreur de test, un problème de configuration,

un problème de donnée, un logiciel tiers, un problème externe au logiciel, un problème de documentation)
- Source – le livrable dans lequel l'erreur a été commise, (p. ex., exigences, conception, conception détaillée, architecture, conception de la base de donnée, documentation utilisateur, documentation de test)
- Type – (p. ex., problème logique, problème de calcul, problème de temps, gestion de donnée, amélioration)

Quand un défaut est résolu (ou a été différé, ou n'a pas pu être confirmé), on peut disposer d'encore plus d'éléments de classification comme:
- La résolution – (p. ex., modification dans le code, modification de documentation, différé, comportement normal, doublon)
- Action de correction – (p. ex., revue d'exigences, revue de code, test unitaire, documentation de la configuration, préparation des données, aucun changement)

En plus de ces catégories de classification, les défauts sont aussi souvent classés selon leur sévérité et leur priorité. De plus, selon le projet, il peut être intéressant de les classer en fonction de leur impact sur la sécurité, de leur impact sur le planning du projet, de leur impact sur les coûts du projet, de leur impact sur les risques du projet, et de leur impact sur la qualité du projet. Ces classifications peuvent être utilisées pour déterminer dans quels délais un correctif doit être livré.

Le dernier domaine de classification est la résolution finale. Les défauts sont souvent regroupés en fonction de leur résolution, (p. ex., corrigé/vérifié, clôturé/pas un problème, différé, ouvert/non-résolu). Cette classification est en général utilisée tout au long d'un projet car les défauts sont suivis tout au long du cycle de vie.

Les valeurs de classification utilisées par une organisation sont souvent adaptées sur mesure. Les éléments donnés plus haut ne sont que des exemples de certaines valeurs utilisées dans l'industrie. Il est important que ces valeurs de classification soient utilisées de façon consistante pour être utiles. Trop de champs de classification rendront l'ouverture et le traitement d'un défaut trop longs. Il est donc important, pour chaque défaut géré, de bien peser l'intérêt les données collectées par rapport à leur coût de traitement incrémental. La possibilité d'adapter les valeurs de classification collectées par un outil est souvent un important critère de choix.

6.5 Analyse des Causes Racines

L'objectif de l'analyse des causes racines est de déterminer ce qui a provoqué le défaut, et de fournir les données permettant des changements dans le processus pour supprimer les causes racines, responsables d'un nombre important de défauts. L'analyse de causes racines est en général menées par la personne qui étudie un défaut pour : soit le résoudre, soit pour déterminer que le problème ne peut pas ou ne doit pas être résolu. Il s'agit en général du développeur. L'Analyste de Test, parce qu'il a une perception acceptable de ce qui peut avoir provoqué le problème, affectera en général une première valeur à la cause racine. En vérifiant la correction, l'Analyste de Test va vérifier la cause racine indiquée par le développeur. Lorsque la cause racine est déterminée, il est fréquent de déterminer ou de confirmer la phase dans laquelle le défaut a été introduit.

On peut citer comme causes racines typiques:
- Exigence imprécise
- Exigence manquante
- Exigence erronée
- Implémentation incorrecte de la conception
- Implémentation incorrecte de l'interface
- Erreur de logique dans le code

- Erreur de calcul
- Erreur matérielle
- Erreur d'interface
- Données invalides

Cette information sur les causes racines est agrégée pour déterminer les principaux problèmes qui aboutissent à la création de défauts. Par exemple, si un grand nombre de défauts est causé par des exigences imprécises, il pourrait être utile de mettre un effort supplémentaire pour mener des revues d'exigences efficaces. De même, si l'implémentation d'interface est un problème parmi les différentes équipes de développement, il est possible qu'il soit nécessaire d'organiser des sessions communes.

Utiliser l'information issue des causes racines pour améliorer les processus aide l'organisation à surveiller les bénéfices de changements efficaces dans les processus et à quantifier le coût des défauts pouvant être rattachés à une cause racine particulière. Cela peut aider à financer des changements dans les processus qui nécessiteraient l'achat d'outils ou d'équipements supplémentaires, de même qu'à faciliter des changements de planning. Le Syllabus ISTQB niveau Expert "Améliorer le processus de test" [ISTQB_EL_ITP] aborde plus dans le détail l'analyse des causes racines.

7. Outils de Test - 45 mn.

Mots clé

test dirigé par les mots-clés, outils de préparation des données de test, outils de conception des tests, outil d'exécution des tests

Objectifs d'apprentissage pour Outils de Test

7.2 Outils de Test et Automatisation

AT-7.2.1 (K2) Expliquer les bénéfices de l'utilisation d'outils de préparation des données de tests, d'outils de conception des tests et d'outils d'exécution des tests

AT-7.2.2 (K2) Expliquer le rôle de l'Analyste de Tests dans l'automatisation dirigée par les mots-clés

AT-7.2.3 (K2) Expliquer les étapes pour résoudre une défaillance dans l'exécution d'un test automatique

7.1 Introduction

Les outils de test peuvent grandement améliorer l'efficacité et l'exactitude de l'effort de test, à condition de mettre en œuvre les bons outils de la bonne façon. Les outils de test doivent être gérés comme n'importe quel autre aspect d'une organisation de test bien gérée. La sophistication et l'applicabilité des outils de test varient grandement et le marché des outils change constamment. Les outils sont en général disponibles auprès d'éditeurs de logiciels commerciaux, de même qu'à partir des sites de différents outils libres ou partagés.

7.2 Outils de Test et Automatisation

Une grande partie du travail de l'Analyste de Test nécessite l'usage efficace d'outils. Savoir quel outil utiliser et quand, peut augmenter l'efficacité de l'Analyste de Test et peut aider à fournir une meilleure couverture de code dans le temps alloué.

7.2.1 Outils de Conception des Tests

Les outils de conception des tests sont utilisés pour aider à créer des cas de test et des données de test. Ces outils peuvent fonctionner à partir de documents d'exigences dans un format particulier, à partir de modèles (p. ex., UML), ou à partir d'entrées fournies par l'Analyste de Test. Les outils de conception des tests sont souvent conçus et construits pour fonctionner avec des formats et des produits particuliers tels que des outils de gestion d'exigences spécifiques.

Les outils de conception des tests peuvent apporter à l'Analyste de Test, de l'information utile pour déterminer les types de test nécessaires à l'obtention du niveau de couverture visé, de la confiance dans le système, ou des actions de mitigation des risques. Par exemple, les outils de classification arborescente génèrent (et affichent) l'ensemble des combinaisons nécessaires pour atteindre une couverture complète, sur la base du critère de couverture sélectionné. Cette information peut ensuite être utilisée par l'Analyste de Test pour déterminer les cas de test à exécuter.

7.2.2 Outils de préparation des données de tests

Les outils de préparation des données de tests apportent plusieurs bénéfices. Certains outils de préparation des données de tests sont capables d'analyser un document, comme un document de spécifications des exigences ou même du code source, pour déterminer les données nécessaires pour atteindre un certain niveau de couverture pendant le test. D'autres outils de préparation des données de tests peuvent prendre un ensemble de données à partir d'un système en production et l'anonymiser pour enlever toute information personnelle, tout en maintenant l'intégrité des données. Les données anonymisées peuvent ensuite être utilisées pour le test, sans risquer de fuite de sécurité ou une utilisation détournée des données. Ceci est en particulier important lorsque de grands volumes de données réalistes sont nécessaires. D'autres outils de génération de données peuvent être utilisés pour générer des données de test à partir d'ensembles de paramètres d'entrée (c.à.d., pour du test aléatoire). Certains de ces outils vont analyser la structure de la base de données pour déterminer quelles entrées seront nécessaires pour l'Analyste de Test.

7.2.3 Outils d'Exécution Automatique des tests

Les outils d'exécution des tests sont principalement utilisés par les Analystes de Test, à tous les niveaux de test, pour exécuter les tests et en vérifier les sorties. L'objectif de l'utilisation d'un outil d'exécution des tests consiste principalement à:

- Réduire les coûts (en termes d'efforts et/ou de temps)
- Exécuter davantage de tests

- Exécuter le même test dans plusieurs environnements
- Rendre l'exécution des tests plus répétable
- Exécuter des tests qu'il serait impossible d'exécuter manuellement (c.à.d., validation d'un très grand ensemble de données)

Ces objectifs s'ajoutent souvent à l'objectif principal d'augmentation de la couverture avec une réduction des coûts.

7.2.3.1 Mise en œuvre

Le retour sur investissement pour les outils d'exécution des tests est en général plus important avec l'automatisation des tests de régression, à cause du faible effort de maintenance attendu, et de l'exécution répétée des tests. Automatiser les tests fumigatoires ("smoke tests") peut également être une bonne application de l'automatisation à cause de l'usage répété de ces tests, du besoin de résultats rapides et, même si le coût de maintenance peut être plus élevé, de la possibilité de disposer d'une évaluation d'une nouvelle livraison (ou un nouveau "build"), d'une façon automatisée, dans un environnement d'intégration continue.

Les outils d'exécution des tests sont généralement utilisés aux niveaux de test système et intégration. Certains outils, en particulier les outils de test d'API, peuvent également être utilisés au niveau composant. Evaluer et appliquer les outils aux endroits où ils seront les plus applicables, contribuera à améliorer le retour sur investissement.

7.2.3.2 Principes des Outils d'Automatisation des Tests

Les outils d'exécution des tests fonctionnent en exécutant un ensemble d'instructions écrites dans un langage de programmation, souvent appelé langage de script. Les instructions pour l'outil sont d'un niveau très détaillé qui précise les entrées, l'ordre des entrées, les valeurs spécifiques utilisées pour les entrées, et les sorties attendues. Cela peut rendre les scripts sensibles aux changements dans le système testé, en particulier quand l'outil interagit avec l'interface utilisateur (IHM).

La plupart des outils d'exécution des tests comprennent un comparateur qui permet de comparer un résultat obtenu à un résultat attendu préalablement enregistré.

7.2.3.3 Mise en œuvre de l'Automatisation des Tests

La tendance avec l'automatisation de l'exécution des tests (comme dans la programmation) est de passer de langages de bas niveau ayant des instructions détaillées à des langages de plus haut niveau utilisant des librairies, des macros et des sous-programmes. Les techniques de conception comme celles dirigées par les mots-clés et celles dirigées par les mots-actions capturent une série d'instructions et les référencent avec un « mot-clé » ou un « mot-action » particulier. Cela permet à l'Analyste de Test d'écrire des cas de test dans un langage naturel, sans se préoccuper du langage de programmation sous-jacent et des fonctions de bas niveau. L'utilisation de cette technique modulaire pour écrire facilite la maintenance lors de changements de la fonctionnalité ou de l'interface du logiciel testé. [Bath08] L'utilisation de mots-clés dans les scripts automatisés est discutée plus en détail ci-après.

Des modèles peuvent être utilisés pour guider la création des mots-clés ou des mots-actions. En regardant la modélisation des processus du métier, qui est souvent incluse dans la documentation des exigences, l'Analyste de Test peut identifier les processus clé du métier qui doivent impérativement être testés. Les étapes pour ces processus peuvent être déterminées avec des points de décision pouvant être organisés durant les processus. Les points de décision peuvent devenir des mots-actions que l'automatisation des tests pourra utiliser à partir de tableaux contenant les mots-clés ou les mots-actions. La modélisation des processus du métier est une méthode de documentation de ces derniers et peut être utilisée pour identifier les processus clé et les points de décision. La modélisation

peut être faite manuellement ou en utilisant des outils qui se baseront sur des entrées correspondant aux règles du métier et aux descriptions des processus.

7.2.3.4 Améliorer le Succès de l'Effort d'Automatisation

Lors de l'identification des tests à automatiser, chaque cas de test ou suite de test candidats doivent être évalués pour voir si l'automatisation se justifie. Beaucoup de projets d'automatisation échouent car ils cherchent à automatiser les cas de test manuels disponibles, sans vérifier le bénéfice réel de l'automatisation. Pour un ensemble donné de cas de test (suite de test), l'idéal peut être d'avoir des tests manuels, des tests semi-automatisés et des tests entièrement automatisés.

Les aspects suivants devraient être pris en compte lors de la mise en œuvre d'un projet d'automatisation de l'exécution des tests:

Bénéfices possibles :

- Le temps d'exécution des tests automatisés est davantage prévisible
- Les tests de régression et la validation des défauts à partir de tests automatisés seront plus rapides et plus fiables tout au long du projet
- Le statut et la progression technique du testeur ou de l'équipe de test peuvent être améliorés par l'utilisation d'outils automatisés
- L'automatisation peut être particulièrement utile avec des cycles de vie de développement itératifs et incrémentaux pour permettre pour chaque « build » ou itération un meilleur test de régression.
- La couverture de certains types de test peut n'être possible qu'avec des outils d'automatisation (p. ex., effort de validation d'une grande quantité de données)
- L'automatisation de l'exécution des tests peut être plus rentable que du test manuel pour les efforts de test liés à de grandes quantités de données à recevoir en entrée, à convertir et à comparer, grâce à la fourniture d'entrées et de vérifications rapides et cohérentes.

Risques possibles:

- Des tests manuels incomplets, inefficaces ou incorrects peuvent être automatisés "tels quels"
- Le « testware » peut être difficile à maintenir, nécessiter de multiples changements lors de modifications du logiciel testé
- L'implication directe du testeur dans l'exécution des tests peut être réduite, ce qui aboutit à un taux de détection plus faible
- L'équipe de test peut manquer de compétences pour utiliser efficacement les outils d'automatisation
- Des tests inutiles qui ne contribuent pas à la couverture de test globale pourraient être automatisés simplement parce qu'ils existent et qu'ils sont stables.
- Les tests peuvent devenir improductifs au fur et à mesure que le logiciel se stabilise (paradoxe du pesticide)

Lors du déploiement d'un outil d'automatisation de l'exécution des tests, il n'est pas toujours judicieux d'automatiser les cas de test manuels tels quels.Il faut plutôt les redéfinir pour une meilleure automatisation. Cela consiste à formater les cas de test, à considérer des « patterns » (schémas) réutilisables ; à étendre les entrées en utilisant des variables plutôt que des valeurs codées en dur ; et en utilisant toutes les fonctionnalités intéressantes de l'outil de test. Les outils d'exécution des tests ont en général la possibilité de lancer de multiples tests, de grouper les tests, de répéter des tests et de changer l'ordre d'exécution, tout en fournissant des facilités de reporting et d'analyse.

Pour beaucoup d'outil d'automatisation de l'exécution des tests, des compétences en programmation sont nécessaires, pour créer des tests (scripts) et des suites de test efficaces et rentables. Il est fréquent que d'importantes suites de tests automatisés soient très difficiles à mettre à jour et à gérer si elles ne sont pas conçues avec précaution. Des formations particulières aux outils de test, à la

programmation et aux techniques de conception seront nécessaires pour garantir l'ensemble des bénéfices escomptés avec les outils.

Pendant l'activité de gestion des tests, il est important d'allouer du temps pour l'exécution manuelle des cas de test automatisés, afin de conserver la connaissance de leur façon de fonctionner, de vérifier le fonctionnement et la validité des données utilisées en entrée et leur couverture.

7.2.3.5 Automatisation dirigée par les Mots-Clés

Les mots-clés (parfois appelés mots-actions) sont principalement, mais pas exclusivement, utilisés pour représenter des interactions-métier de haut niveau avec un système (p. ex., "annulation de commande"). Chaque mot-clé est généralement utilisé pour représenter un certain nombre d'interactions détaillées entre un acteur et le système sous test. Des séquences de mots-clés (incluant des données de test adéquates) sont utilisées pour spécifier les cas de test. [Buwalda01]

Lors de l'automatisation des tests, un mot-clé correspond au développement d'un ou plusieurs scripts de test exécutables. Des outils lisent les cas de test écrits comme des séquences de mots-clés et appellent les scripts de tests correspondant à l'implémentation de la fonctionnalité des mots-clés. Les scripts sont implémentés de façon très modulaire pour permettre une association facile avec des mots-clés spécifiques. Des compétences en programmation sont nécessaires pour implémenter ces scripts modulaires.

Les avantages premiers de l'automatisation avec les tests dirigés par les mots-clés sont :

- Les mots-clés relevant d'une application ou d'un domaine particulier du métier peuvent être définis par des experts du domaine. Cela peut rendre la tâche de spécifications des cas de test plus efficace.
- Une personne ayant une expertise basique du domaine peut bénéficier de l'exécution automatique des cas de test (une fois que les mots-clés ont été implémentés en scripts) sans avoir à comprendre l'automatisation de code sous-jacente.
- Les cas de test écrits avec des mots-clés sont plus faciles à maintenir car ils sont moins sujets aux modifications si des détails dans le logiciel sous test changent.
- Les spécifications des cas de test sont indépendantes de leur implémentation. Les mots-clés peuvent être implémentés avec différents langages de script et outils.

Les scripts d'automatisation (le code correspondant à l'automatisation) utilisant l'information des mots-clés ou des mots-actions sont en général écrits par des développeurs ou des Analystes Techniques de Test ; alors que l'Analyste de Test sera en général chargé de créer et de maintenir les données relatives aux mots-actions/clés. Bien que l'automatisation dirigée par les mots-clés soit en général lancée durant la phase de test système, le développement du code peut débuter dès les phases d'intégration. Dans un environnement itératif, le développement des tests automatisés est un processus continu.

Une fois que les mots-clés et les données en entrée sont créés, l'Analyste de Test prend, en général, la responsabilité d'exécuter les cas de tests dirigés par les mots-clés et d'analyser les défaillances pouvant survenir. Quand une anomalie est détectée l'Analyste de Test doit rechercher la cause de la défaillance pour déterminer si le problème concerne les mots-clés, les données en entrée, le script d'automatisation lui-même ou l'application testée. En général, la première étape dans l'investigation est d'exécuter le même test, manuellement, avec les mêmes données, pour voir si la défaillance est dans l'application elle-même. Si une défaillance n'est pas observée alors, l'Analyste de Test devrait revoir la séquence de tests ayant conduit à la défaillance, pour déterminer si le problème s'est produit à une étape précédente (peut-être avec la production de données incorrectes), mais ne réapparaît que plus tard dans le traitement. Si l'l'Analyste de Test n'est pas capable de déterminer la cause de la défaillance, l'information correspondante devrait être transmise à l'Analyste Technique de Test ou à un développeur pour une analyse approfondie.

7.2.3.6 Causes d'Echec de l'Effort d'Automatisation

Les projets d'automatisation de l'exécution des tests parviennent rarement à atteindre leurs objectifs. Ces échecs peuvent être dus à une flexibilité insuffisante dans l'usage de l'outil de test, à des compétences en développement insuffisantes ou à des attentes irréalistes sur les problèmes pouvant être résolus grâce à l'automatisation de l'exécution des tests. Il est important de noter que toute automatisation de l'exécution des tests nécessite des efforts de gestion, des compétences et de l'attention, comme tout autre projet de développement logiciel. Il est nécessaire de consacrer du temps à la création d'une architecture fiable, au suivi de pratiques de conception adaptées, à la gestion de configuration et au suivi des bonnes pratiques de codage. Les scripts de test automatisés doivent être testés car ils peuvent contenir des défauts. Il peut être nécessaire de prendre en compte la performance de ces scripts. L'utilisabilité des outils doit aussi être prise en compte, non seulement pour le développeur, mais aussi pour les personnes qui devront utiliser l'outil pour exécuter les tests. Il peut être nécessaire de concevoir une interface entre l'outil et l'utilisateur pour permettre un accès aux cas de test, qui soit basé sur une organisation logique pour le testeur, tout en garantissant l'accessibilité de l'outil.

8. Références

8.1 Standards

- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE)
  Chapitres 1 et 4
- [ISO9126] ISO/IEC 9126-1:2001, Software Engineering - Software Product Quality,
  Chapitres 1 and 4
- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12B.1992.
  Chapitre1

8.2 Documents ISTQB

- [ISTQB_AL_OVIEW] ISTQB Advanced Level Overview, Version 1.0
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 1.0
- [ISTQB_ALTTA_SYL] ISTQB Advanced Level Technical Test Analyst Version 1.0
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 2011
- [ISTQB_GLOSSARY] Standard glossary of terms used in Software Testing, Version 2.2, 2012

8.3 Ouvrages

[Bath08] Graham Bath, Judy McKay, “The Software Test Engineer’s Handbook”, Rocky Nook, 2008, ISBN 978-1-933952-24-6

[Beizer95] Boris Beizer, “Black-box Testing”, John Wiley & Sons, 1995, ISBN 0-471-12094-4

[Black02]: Rex Black, “Managing the Testing Process (2nd edition)”, John Wiley & Sons: New York, 2002, ISBN 0-471-22398-0

[Black07]: Rex Black, “Pragmatic Software Testing”, John Wiley and Sons, 2007, ISBN 978-0-470-12790-2

[Buwalda01]: Hans Buwalda, “Integrated Test Design and Automation”, Addison-Wesley Longman, 2001, ISBN 0-201-73725-6

[Cohn04]: Mike Cohn, “User Stories Applied: For Agile Software Development”, Addison-Wesley Professional, 2004, ISBN 0-321-20568-5

[Copeland03]: Lee Copeland, “A Practitioner's Guide to Software Conception des tests”, Artech House, 2003, ISBN 1-58053-791-X

[Craig02]: Rick David Craig, Stefan P. Jaskiel, “Systematic Software Testing”, Artech House, 2002, ISBN 1-580-53508-9

[Gerrard02]: Paul Gerrard, Neil Thompson, “Risk-based e-business Testing”, Artech House, 2002, ISBN 1-580-53314-0

[Gilb93]: Tom Gilb, Graham Dorothy, “Software Inspection”, Addison-Wesley, 1993, ISBN 0-201-63181-4

[Graham07]: Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black “Foundations of Software Testing”, Thomson Learning, 2007, ISBN 978-1-84480-355-2

[Grochmann94]: M. Grochmann (1994), Test case design using Classification Trees, in: conference proceedings STAR 1994

[Koomen06]: Tim Koomen, Leo van der Aalst, Bart Broekman, Michiel Vroon "TMap NEXT, for result driven testing", UTN Publishers, 2006, ISBN 90-72194-80-2

[Myers79]: Glenford J. Myers, “The Art of Software Testing”, John Wiley & Sons, 1979, ISBN 0-471-46912-2

[Splaine01]: Steven Splaine, Stefan P. Jaskiel, “The Web-Testing Handbook”, STQE Publishing, 2001, ISBN 0-970-43630-0

[vanVeenendaal12]: Erik van Veenendaal, “Practical risk based testing – The PRISMA approach”, UTN Publishers, The Netherlands, ISBN 9789490986070

[Wiegers03]: Karl Wiegers, “Software Requirements 2”, Microsoft Press, 2003, ISBN 0-735-61879-8

[Whittaker03]: James Whittaker, “How to Break Software”, Addison-Wesley, 2003, ISBN 0-201-79619-8

[Whittaker09]: James Whittaker, “Exploratory Software Testing”, Addison-Wesley, 2009, ISBN 0-321-63641-4

8.4 Autres Références

Les références suivantes désignent de l’information disponible sur Internet et ailleurs. Même si ces références ont été vérifiées lors de la publication de ce syllabus du Niveau Avancé, l’ISTQB ne peut être tenue pour responsable si elles ne sont plus valides.

- Chapitre 3
  - Czerwonka, Jacek: www.pairwise.org
  - Bug Taxonomy: www.testingeducation.org/a/bsct2.pdf
  - Sample Bug Taxonomy based on Boris Beizer’s work: inet.uni2.dk/~vinter/bugtaxst.doc
  - Good overview of various taxonomies: testingeducation.org/a/bugtax.pdf
  - Heuristic Risk-Based Testing By James Bach
  - From “Exploratory & Risk-Based Testing (2004) www.testingeducation.org"
  - Exploring Tests Exploratoire , Cem Kaner and Andy Tikam , 2003
  - Pettichord, Bret, “An Tests Exploratoire Workshop Report”, www.testingcraft.com/exploratorypettichord
- Chapitre 4
  - www.testingstandards.co.uk

9. Index