Testeur Certifié

Extension Niveau Fondation

Testeur Agile

Version 2014

International Software Testing Qualifications Board



Note de Copyright
Ce document ne peut être copié en partie ou en intégralité, que si la source est reconnue.

Copyright © International Software Testing Qualifications Board (Appelée ci-dessous ISTQB®).

Groupe de travail “Extension du Syllabus niveau fondation Testeur Agile”: Rex Black (Chair), Bertrand Cornanguer (Vice Chair), Gerry Coleman (Leader du groupe objectifs d’apprentissage), Debra Freidenberg (Leader du groupe Examens), Alon Linetzki (Leader du groupe Objectifs Métier et marketing), Tauhida Parveen (Editeur), and Leo van der Aalst (Responsable du développement);

Auteurs Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh, and Stephan Weber;

Réviseurs internes Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael, and Erik van Veenendaal; 2013-2014.

© International Software Testing Qualifications Board

Historique des révisions

Version	Date	Commentaires
Syllabus v0.1	26JULY2013	Sections individuelles
Syllabus v0.2	16SEPT2013	Intégration des commentaires après la revue du groupe de travail v01
Syllabus v0.3	20OCT2013	Intégration des commentaires après la revue du groupe de travail v02
Syllabus v0.7	16DEC2013	Intégration des commentaires de l'Alpha revue v03
Syllabus v0.71	20DEC2013	Mises à jour du groupe de travail dans la v07
Syllabus v0.9	30JAN2014	Beta version
Syllabus v1.0	31MAI2014	Version pour assemblée générale

Traduction française : Comité Français des Tests Logiciels 2014 V1.1

© International Software Testing Qualifications Board

Table des Matières

Historique des révisions 3
Table of Contents 4
Remerciements 6
0. Introduction à ce syllabus 7
0.1 Objectif de ce document 7
0.2 Vue globale 7
0.3 Objectifs d'apprentissage examinables 7
1. Développement logiciel Agile 8
1.1 Les fondamentaux du développement logiciel Agile 9
1.1.1 Les développements logiciel Agile et le manifeste Agile 9
1.1.2 Approche d'équipe Intégrée 10
1.1.3 Un feedback au plus tôt et fréquent 11
1.2 Aspects des approches Agile 11
1.2.1 Approches de développement logiciel Agile 11
1.2.2 Création collaborative de User Story 13
1.2.3 Rétrospectives 14
1.2.4 Intégration Continue 15
1.2.5 Plannification de release et d'itérations 16
2. Principes, Pratiques, et Processus fondamentaux Agile– 105 mins. 19
2.1 Les différences des tests entre les approches traditionnelles et Agile 20
2.1.1 Activités de test et de développement 20
2.1.2 Produits d'activité des projets 21
2.1.3 Niveaux de Test 22
2.1.4 Tests et gestion de configuration 23
2.1.5 Options d'organisation avec des tests indépendants 24
2.2 Statuts du test dans les projets Agile 24
2.2.1 Communiquer les statuts du test, l'avancement, et la qualité Produit 24
2.2.2 Gérer les risques de régression en faisant évoluer les cas de test manuels et automatisés 25
2.3 Rôles et compétences d'un testeur dans une équipe Agile 27
2.3.1 Compétences d'un testeur Agile 27
2.3.2 Le Rôle d'un testeur dans une équipe Agile 28
3. Méthodes, Techniques, et outils pour les tests Agile – 480 mins. 29
3.1 Méthodes de test Agile 30
3.1.1 Développement piloté par les tests, Développement piloté par les tests d'Acceptation, et Développement piloté par les tests de Comportement 30
3.1.2 La Pyramide des tests 31
3.1.3 Quadrants de Test, Niveaux de Test, et type de Test 31
3.1.4 Le Rôle d'un Testeur 32
3.2 Evaluer les risques Qualité et estimer l'effort de Test 34
3.2.1 Evaluer les risques de qualité sur les Projets Agile 34
3.2.2 Estimer l'effort de test basé sur le contenu et les risques 35
3.3 Techniques dans les projets Agile 36
3.3.1 Critères d'acceptation, adéquation de la couverture, et autres informations pour les Tests 36
3.3.2 Appliquer le Développement piloté par les tests d'acceptation (ATDD) 39

© International Software Testing Qualifications Board

3.3.3 Conception des tests boite noire Fonctionnels et Non-Fonctionnels 39
3.3.4 Les tests exploratoires et les tests Agile 40
3.4 Outils dans les projets Agile 42
3.4.1 Outils de gestion des tâches et de suivi 42
3.4.2 Outils de communication et de partage d'information 42
3.4.3 Build de logiciel et outils de distribution 43
3.4.4 Outils de gestion de configuration 43
3.4.5 Outils de conception, d'implémentation, et d'exécution des tests 43
3.4.6 Outils de Cloud Computing et de virtualisation 44
4. Références 45
4.1 Standards 45
4.2 Documents ISTQB et CFTL 45
4.3 Livres 45
4.4 Terminologie Agile 46
4.5 Autres Références 46
5. Index 47

© International Software Testing Qualifications Board

Remerciements

Ce document a été produit par un groupe de travail de l'ISTQB pour le niveau fondation.

L'équipe en charge de cette extension Agile remercie l'équipe de réviseurs et tous les bureaux nationaux pour leurs suggestions et entrées.

Au moment où ce syllabus a été réalisé, les membres de l'équipe étaient: Rex Black (Chair), Bertrand Cornanguer (Vice Chair), Gerry Coleman (Leader du groupe objectifs d'apprentissage), Debra Freidenberg (Leader du groupe Examens), Alon Linetzki (Leader du groupe Objectifs Métier et marketing), Tauhida Parveen (Editeur), et Leo van der Aalst (Responsable du développement);

Auteurs: Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh, and Stephan Weber;

Reviseurs internes: Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael, and Erik van Veenendaal.

L'équipe remercie également les personnes suivantes, membres des bureaux nationaux et les experts de la communauté Agile, qui ont participé en révisant, en commentant et en discutant cette extension Agile:

Dani Almog, Richard Berns, Stephen Bird, Monika Bögge, Afeng Chai, Josephine Crawford, Tibor Csöndes, Huba Demeter, Arnaud Foucal, Cyril Fumery, Kobi Halperin, Inga Hansen, Hanne Hinz, Jidong Hu, Phill Isles, Shirley Itah, Martin Klonk, Kjell Lauren, Igal Levi, Rik Marselis, Johan Meivert, Armin Metzger, Peter Morgan, Ninna Morin, Ingvar Nordstrom, Chris O'Dea, Klaus Olsen, Ismo Paukamainen, Nathalie Phung, Helmut Pichler, Salvatore Reale, Stuart Reid, Hans Rombouts, Petri Säilynoja, Soile Sainio, Lars-Erik Sandberg, Dakar Shalom, Jian Shen, Marco Sogliani, Lucjan Stapp, Yaron Tsubery, Sabine Uhde, Stephanie Ulrich, Tommi Välimäki, Jurian Van de Laar, Marnix Van den Ent, António Vieira Melo, Wenye Xu, Ester Zabar, Wenqiang Zheng, Peter Zimmerer, Stevan Zivanovic and Terry Zuo.

Ce document a été formellement approuvé pour release par l'assemblée générale de l'ISTQB® le 31 Mai 2014.

© International Software Testing Qualifications Board

0. Introduction à ce syllabus

0.1 Objectif de ce document

Ce syllabus donne les bases de la qualification internationale en tests de logiciels au niveau fondation pour le testeur agile. L'ISTQB® fournit ce syllabus:

- Aux bureaux nationaux, pour traduction dans leur langue et pour accréditation des fournisseurs de formation. Les bureaux nationaux peuvent adapter le syllabus à leur langage et modifier les références pour les adapter à leurs publications.
- Aux bureaux d'examens, pour créer les questions d'examen dans leur langue, adaptées aux objectifs d'apprentissage de chaque syllabus.
- Aux Fournisseurs de formation, pour produire le matériel de formation et déterminer les méthodes appropriées d'apprentissage.
- Aux candidats à la certification, pour se préparer à l'examen (intégré à une formation ou pas).
- A la communauté internationale d'ingénierie logicielle et système, pour faire progresser la professionnalisation dans les tests de logiciels et des systèmes, et comme base pour des livres et des articles.

L'ISTQB® peut autoriser d'autres entités à utiliser ce syllabus pour d'autres sujets, en effectuant leur demande au préalable et en obtenant une autorisation écrite.

0.2 Vue globale

Le niveau fondation est composé de deux syllabus séparés:

- Testeur certifié
- Testeur Agile

Le document "Overview pour le testeur agile au niveau fondation » [ISTQB_FA_OVIEW] comprend les informations suivantes:

- Objectifs Métier pour chaque syllabus
- Résumé pour chaque syllabus
- Relations entre les syllabus
- Description des niveaux cognitifs (K-levels)
- Annexes

0.3 Objectifs d'apprentissage examinables

Les objectifs d'apprentissage instruisent les objectifs Métier et sont utilisés pour créer des examens pour obtenir le niveau testeur certifié au niveau fondation. En général, toutes les parties du syllabus sont examinables au niveau K1. Ce qui signifie que le candidat reconnaitra, se souviendra, et retreindra un terme ou un concept. Les objectifs d'apprentissage spécifiques, aux niveaux K1, K2, et K3, sont présentés au début de chaque chapitre.

© International Software Testing Qualifications Board

1. Développement logiciel Agile — 150 minutes

Termes

Manifeste Agile, Développement logiciel Agile, Modèle de développement incrémental, Modèle de développement itératif, cycle de vie logiciel, Automatisation des tests, Bases de test, Développement piloté par les tests, Oracle de test, User Story

Objectifs d'apprentissage pour le développement Agile

1.1 Les fondamentaux du développement logiciel agile

FA-1.1.1 (K1) Rappeler les concepts de base du développement logiciel Agile basé sur le manifeste Agile

FA-1.1.2 (K2) Comprendre les avantages de l'approche équipe intégrée

FA-1.1.3 (K2) Comprendre les bénéfices des feedback fréquents et au plus tôt

1.2 Aspects des approches Agile

FA-1.2.1 (K1) Rappel des approches de développement logiciel Agile

FA-1.2.2 (K3) Ecrire des user Story en collaboration avec le développement, les représentants Métier, et les Product Owners

FA-1.2.3 (K2) Comprendre comment les rétrospectives peuvent être utilisées comme mécanisme d'amélioration dans les projets Agile

FA-1.2.4 (K2) Comprendre l'utilisation et l'utilité de l'intégration continue

FA-1.2.5 (K1) Connaitre les différences entre planning d'itération et de release, et comment un testeur ajoute de la valeur dans chacune de ces activités

© International Software Testing Qualifications Board

1.1 Les fondamentaux du développement logiciel Agile

Un testeur sur un projet Agile travaille différemment que sur un projet traditionnel. Les testeurs doivent comprendre les valeurs et les principes qui sous-tendent les projets Agiles, et comment les testeurs font partie intégrante d'une approche intégrée avec les développeurs et les représentants des Métiers. Les membres d'un projet agile communiquent les uns avec les autres tôt et fréquemment ce qui aide à supprimer les défauts tôt et développer un produit de qualité.

1.1.1 Les développements logiciel Agile et le manifeste Agile

En 2001, un groupe de personnes, représentant le plus largement les méthodologies légères de développement logiciel utilisées, se sont mis d'accord sur un ensemble commun de valeurs et de principes qui sont connus sous le nom de Manifeste Agile [agilemanifesto.org]. Le manifeste Agile contient quatre déclarations de valeurs :

- Les individus et leurs interactions plus que les processus et les outils
- Des logiciels opérationnels plus qu'une documentation exhaustive
- La collaboration avec les clients plus que la négociation contractuelle
- L'adaptation au changement plus que le suivi d'un plan

Le manifeste Agile fait valoir que si les concepts de la partie de droite des concepts précédents ont de la valeur, ceux de gauche en ont davantage.

Les Individus et leurs interactions

Le développement Agile est très axé sur les personnes. Des équipes de personnes construisent des logiciels, et c'est grâce à une communication et des interactions continues plus que par un recours aux outils ou procédés, que les équipes peuvent travailler plus efficacement.

Des logiciels opérationnels

D'un point de vue client, un logiciel opérationnel est beaucoup plus utile et précieux qu'une documentation très détaillée et offre la possibilité de fournir des feedbacks rapides à l'équipe de développement. En outre, parce qu'un logiciel opérationnel, même avec des fonctionnalités réduites, est disponible beaucoup plus tôt dans le cycle de développement, le développement Agile peut conférer des avantages significatifs en time-to-market. Le développement Agile est donc particulièrement utile dans les environnements Métier qui évoluent très rapidement et/ou les problèmes et/ou les solutions ne sont pas clairs ou lorsque les Métiers souhaitent innover dans de nouveaux domaines.

La collaboration avec les clients

Les clients rencontrent souvent beaucoup de difficultés à exprimer le système dont ils ont besoin. Collaborer directement avec le client améliore les chances de comprendre exactement ce dont il a besoin. Bien qu'avoir des contrats avec les clients peut être important, travailler en collaboration régulièrement et étroitement avec eux peut rendre probable le succès des projets.

L'adaptation au changement

Les changements sont inévitables dans les projets logiciels. Le domaine Métier de l'entreprise, la législation, les activités des concurrents, les avancées technologiques, et autres facteurs peuvent avoir des influences majeures sur le projet et ses objectifs. Ces facteurs doivent être adaptés au

© International Software Testing Qualifications Board

processus de développement. À ce titre, avoir de la flexibilité dans la façon d'appréhender le changement est plus important que de simplement adhérer rigoureusement à un plan.

Principes

Les valeurs fondamentales du manifeste Agile sont traduites en douze principes :

- Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.
- Accueillez positivement les changements de besoins, même tard dans le projet. Les processus Agiles exploitent le changement pour donner un avantage compétitif au client.
- Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois avec une préférence pour les délais les plus courts.
- Les utilisateurs ou leurs représentants, et les développeurs doivent travailler ensemble tous les jours tout au long du projet.
- Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont ils ont besoin et faites-leur confiance pour atteindre les objectifs fixés.
- La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et entre les membres de celle-ci est le dialogue en face à face.
- Le logiciel opérationnel est la principale mesure d'avancement.
- Les processus Agiles encouragent un rythme de développement soutenable dans la durée. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.
- Une attention continue à l'excellence technique et à une bonne conception renforce l'Agilité.
- La simplicité – c'est-à-dire l'art de minimiser la quantité de travail inutile – est essentielle.
- Les meilleures architectures, spécifications et conceptions émergent d'équipes auto organisées.
- À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence.

Les différentes méthodologies Agiles fournissent des pratiques normatives pour mettre ces valeurs et principes en action.

1.1.2 Approche d'équipe Intégrée

L'approche d'équipe intégrée signifie impliquer chaque personne qui a les connaissances et les compétences nécessaires pour assurer la réussite du projet. L'équipe comprend des représentants des clients et d'autres intervenants Métier qui déterminent les caractéristiques du produit. L'équipe devrait être relativement petite : On a observé que les équipes qui réussissent ont de 3 à 9 personnes. Idéalement, toute l'équipe partage le même espace de travail, car cette co-location facilite fortement la communication et les interactions. L'approche équipe intégrée est facilitée par les réunions quotidiennes appelées Stand-up Meeting (voir chapitre 2.2.1) impliquant tous les membres de l'équipe, et où le travail en cours est communiqué et les obstacles à la progression sont mis en évidence. L'approche équipe intégrée facilite une dynamique d'équipe plus efficace et efficiente.

L'approche équipe intégrée pour produire un développement est un des bénéfices principaux des développements Agile. Ces bénéfices incluent:

- Améliore la communication et la collaboration au sein de l'équipe
- Permet aux diverses compétences au sein de l'équipe d'être exploitées au profit du projet
- Faire de la Qualité la responsabilité de chacun

© International Software Testing Qualifications Board

Dans les projets Agiles, toute l'équipe est responsable de la qualité. L'essence de l'approche équipe intégrée réside dans les testeurs, les développeurs et les représentants Métier travaillant ensemble à chaque étape du processus de développement. Les testeurs travailleront en étroite collaboration avec les développeurs et les représentants Métier pour s'assurer que les niveaux de qualité souhaités sont atteints. Cela inclus de soutenir et de collaborer avec les représentants des Métiers pour les aider à créer des tests d'acceptation pertinents, de travailler avec les développeurs pour s'entendre sur la stratégie de test, et de se décider sur les approches d'automatisation des test. Les testeurs peuvent ainsi transférer et étendre la connaissance du test aux autres membres de l'équipe et influencer le développement du produit.
Toute l'équipe participe à toutes les consultations ou réunions dans lesquelles les caractéristiques du produit sont présentées, analysées ou estimées. Le concept d'impliquer les testeurs, les développeurs et des représentants Métier dans tous les débats de la fonction est connu comme le "pouvoir des trois" [Crispin08].

1.1.3 Un feedback au plus tôt et fréquent

Les projets Agiles ont des itérations courtes afin que l'équipe projet puisse recevoir des feedbacks au plus tôt et continus sur la qualité des produits tout au long du cycle de développement. Une façon de fournir un feedback rapide est l'intégration continue (voir la Section 1.2.4).

Lorsqu'on utilise des approches de développement séquentielles, souvent le client ne voit pas le produit tant que le projet n'est pas pratiquement terminé. À ce stade, il est souvent trop tard pour l'équipe de développement de s'occuper efficacement des points que le client peut adresser. En recevant un feedback client fréquent au cours de la progression du projet, les équipes Agiles peuvent intégrer la plupart des nouveaux changements dans le processus de développement du produit. Les feedback tôt et fréquents aident à mettre l'accent sur les fonctionnalités qui ont la plus haute valeur Métier, ou leurs risques associés, et celles-ci sont livrées en premier au client. Il permet également de mieux manager l'équipe puisque l'aptitude de l'équipe est transparente à chacun. Par exemple, quelle quantité de travail peut-on faire dans un sprint ou une itération ? Qu'est-ce qui pourrait nous aider à aller plus vite ? Qu'est-ce qui nous empêche de le faire ?
Les avantages de recevoir des feedbacks tôt et fréquents sont les suivants :

- Eviter les incompréhensions sur les exigences qui n'auraient été décelées que plus tard dans le cycle de développement lorsqu'ils sont plus coûteux à corriger
- Clarifier les fonctionnalités que le client demande, les rendant disponibles au plus tôt pour les clients afin que e produit reflète mieux ce que veut le client
- Découvrir, isoler et résoudre tôt (via l'intégration continue) les problèmes
- Informer l'équipe Agile sur sa productivité et sa capacité à livrer
- Promouvoir une dynamique projet cohérente

1.2 Aspects des approches Agile

Un certain nombre d'approches Agile sont utilisées par les organisations. Dans ce syllabus, nous allons utiliser trois de ces techniques habituelles pour fournir des exemples: Extreme Programming, Scrum et Kanban. Les pratiques courantes des organisations Agile comprennent la création collaborative des user Story, les rétrospectives, l'intégration continue et la planification de chaque itération comme de la release globale. Cette sous-section décrit certaines de ces approches Agile.

1.2.1 Approches de développement logiciel Agile

Chacune des approches Agile implémente les valeurs et les principes du manifeste Agile de différentes manières. Dans ce syllabus, nous examinons trois implémentations représentatives de ces approches Agiles: Extreme Programming (XP), Scrum et Kanban.

© International Software Testing Qualifications Board

l'Extreme Programming

L'Extreme Programming (XP), introduit à l'origine par Kent Beck [Beck04], est une approche Agile du développement logiciel décrit par certaines valeurs, principes et pratiques de développement.

XP contient 5 valeurs pour guider le développement: Communication, simplicité, feedback, courage, et respect.

XP décrit un ensemble de principes comme lignes directrices supplémentaires: Humanité, économie, bénéfice mutuel, similitude, amélioration, diversité, reflet, flux, opportunité, redondance, défaillance, qualité, petites étapes, et responsabilité acceptée.

XP décrit treize pratiques primaires: S'asseoir ensemble, équipe intégrée, espace d'information, produit de travail énergique, programmation en binôme, Story, cycle hebdomadaire, cycle trimestriel, se relâcher, Build en dix minutes, intégration continue, programmation pilotée par les tests, et conception incrémentale.

De nombreuses approches de développement logiciel Agile en cours aujourd'hui sont influencées par XP et ses valeurs et principes. Par exemple, Les équipes Agile qui suivent Scrum intègrent souvent des pratiques XP.

Scrum

Scrum est un Framework de management Agile qui contient les actes constitutifs et les pratiques suivantes [Schwaber01]:

- Sprint: Scrum divise un projet en itérations (appelées sprints) de longueur fixe (habituellement de 2 à 4 semaines).
- Incrément Produit: Chaque résultat de sprint est un produit potentiellement livrable/déployable (appelé incrément).
- Backlog Produit: Le Product Owner gère une liste priorisée d'éléments de produits planifiés (appelés le backlog Produit). Le backlog produit évolue de sprint en sprint (appelé raffinement du backlog).
- Backlog de Sprint: Au début de chaque sprint, l'équipe Scrum sélectionne un ensemble d'éléments de priorité haute (appelés le backlog de sprint) à partir du backlog Produit. Puisque l'équipe Scrum, et non pas le Product Owner, sélectionne les éléments à réaliser dans le sprint, la sélection est dite faite suivant le principe de traction plutôt que de poussée
- Définition de Terminé: Pour s'assurer qu'il y a un produit potentiellement livrable à chaque fin de sprint, l'équipe Scrum discute et définit les critères appropriés pour la complétude du sprint. La discussion dépend de la compréhension par l'équipe des éléments du backlog et les exigences Produit.
- Timeboxing: Seules les tâches, les exigences, ou les fonctionnalités que l'équipe souhaite finir dans le sprint font partie du backlog de sprint. Si l'équipe de développement ne peut pas finir une tâche lors d'un sprint, la fonctionnalité du produit associée est supprimée du sprint et est replacée dans le backlog Produit. Le Timeboxing s'applique non seulement aux tâches, mais à d'autres situations (ex : renforcer les temps de départ et de fin des réunions).
- Transparence: L'équipe de développement rapporte et met à jour le statut du sprint sur une base journalière appelée le Daily Scrum. Cela permet de rendre visible le contenu et la progression du sprint en cours, en incluant les résultats de test, pour l'équipe de management et toutes les parties intéressées. Par exemple, l'équipe de développement peut montrer le statut du sprint sur un tableau blanc.

Scrum définit trois rôles:

- Le Scrum Master: assure que les pratiques et les règles Scrum sont implémentées et suivies, et résout toute violation, problème de ressource, ou tout autre blocage qui pourrait empêcher

© International Software Testing Qualifications Board

l'équipe de suivre les pratiques et les règles. Cette personne n'est pas le chef de l'équipe, mais un coach.
- Le Product Owner: représente le client, génère, maintient, et priorise le backlog produit. Cette personne n'est pas le chef de l'équipe.
- L'équipe de développement: développe et teste le produit. L'équipe est auto organisée: Il n'y a pas de chef d'équipe, donc l'équipe prend ses décisions. L'équipe est également inter fonctionnelle (voir Section 2.3.2 et Section 3.1.4).

Scrum (contrairement à XP) ne dicte pas de technique de développement spécifique (ex : programmation pilotée par les tests). De plus, Scrum ne fournit pas de guide sur comment le test doit être réalisé dans un projet Scrum.

Kanban

Kanban [Anderson13] est une approche de management qui est parfois utilisée dans les projets Agile. L'objectif général est de visualiser et d'optimiser le flux de travail dans une chaine de valeurs ajoutées. Kanban utilise trois instruments [Linz14]:

- Le Tableau Kanban: La chaine de valeurs qui doit être gérée est visualisée par un tableau Kanban. Chaque colonne montre un stade de développement, qui est une file d'attente d'activités connexes, ex: développement, ou tests. Les éléments devant être produits ou les tâches à réaliser sont symbolisées par des tickets se déplaçant de gauche à droite sur le tableau en fonction de leur stade de développement.
- Limite de file d'attente: Le nombre de tâches actives en parallèle est strictement limité. Cela est contrôlé par un nombre maximum de tickets autorisés pour la file d'attente d'un stade de développement et/ou globalement pour le tableau. Chaque fois qu'une file d'attente d'un stade de développement a de la disponibilité, il est tiré un ticket du stade de développement précédent.
- Délai d'exécution: Kanban est utilisé pour optimiser le flux continu de tâches réduisant au minimum le délai d'exécution (moyen) pour la chaîne de valeur complète.

Kanban comporte des similitudes avec Scrum. Dans les deux frameworks, visualiser les tâches actives (Ex: sur un tableau blanc public) donne de la transparence sur le contenu et la progression des tâches. Les tâches non encore planifiées sont en attente dans un backlog et déplacées sur le tableau Kanban dès qu'une nouvelle place (Capacité de production) est disponible.

Les Itérations ou les sprints sont optionnels en Kanban. Le processus Kanban permet de livrer élément par élément, plutôt que comme élément faisant partie de la release. Le Timeboxing en tant que mécanisme de synchronisation est donc optionnel, contrairement à Scrum qui synchronise toutes les tâches dans un sprint.

1.2.2 Création collaborative de User Story

Les spécifications sont souvent une des raisons majeures de l'échec de projets. Les problèmes de spécification peuvent résulter du manque de vision des utilisateurs dans leurs besoins réels, de l'absence de vision globale du système, des fonctionnalités redondantes ou contradictoires, et autres problèmes de communication. Dans un développement Agile, les user Story sont écrites pour capturer les exigences selon le point de vue des développeurs, des testeurs, et des représentants Métier. Dans un cycle de développement séquentiel, cette vision partagée des fonctionnalités est donnée au travers des revues formelles après que les exigences soient écrites. Dans un projet Agile, cette vision est donnée à travers des revues informelles fréquentes en même temps que les exigences sont écrites.

Les user Story doivent adresser à la fois les caractéristiques fonctionnelles et non-fonctionnelles. Chaque Story inclut des critères d'acceptation pour ces caractéristiques. Ces critères devraient être

© International Software Testing Qualifications Board

définis par une collaboration entre les représentants Métier, les développeurs, et les testeurs. Ils donnent une vision étendue de la fonctionnalité aux développeurs et aux testeurs, que les représentants Métier vont valider. Une équipe Agile considère une tâche terminée quand un ensemble de critères d'acceptation ont été satisfaits.

Typiquement, la vision du testeur va améliorer la User Story en identifiant des détails manquants ou des exigences non fonctionnelles. Un testeur peut contribuer en posant des questions ouvertes sur la User Story aux représentants Métier, en proposant des façons de tester la User Story, et de confirmer les critères d'acceptation.

Cette paternité collaborative de la User Story peut nécessiter de mettre en œuvre des techniques comme le brainstorming ou le mind-mapping. Le testeur peut utiliser la technique INVEST [INVEST]:

- Indépendante
- Négociable
- Apportant de la valeur
- Estimable
- Petite
- Testable

Selon le concept des 3C [Jeffries00], une User Story est la conjonction de trois éléments:

- Carte: La carte est le media physique décrivant la User Story et ses bénéfices. Elle identifie l'exigence, sa criticité, les temps de développement et de test, et les critères d'acceptation pour cette User Story. La description doit être précise, car elle sera utilisée dans le backlog produit.
- Conversation : La conversation explique comment le logiciel sera utilisé. La conversation peut être documentée ou verbale. Les testeurs ayant des points de vue différents des développeurs et des représentants Métier apportent des éléments de valeur pour échanger sur des idées, opinions, et expériences. La Conversation commence pendant la phase de planification de la release et continuera quand la User Story sera planifiée.
- Confirmation: Les critères d'acceptation, discutés dans la Conversation, sont utilisés pour confirmer que la User Story est terminée. Ces critères d'acceptation peuvent s'étendre sur plusieurs user Story. Des tests à la fois positifs et négatifs devraient être mis en œuvre pour couvrir les critères. Pendant la confirmation, différents participants jouent le rôle d'un testeur. Ils peuvent être développeurs ou spécialistes en performance, sécurité, interopérabilité, et autres caractéristiques qualité. Pour considérer une User Story comme terminée (Done), les critères d'acceptation définis devraient être testés et montrés pour être satisfaits.

Les équipes Agile varient sur leur façon de commenter les user Story. Indépendamment de l'approche mise en œuvre pour documenter les User Story, la documentation devrait être concise, suffisante, et nécessaire.

1.2.3 Rétrospectives

En développement Agile, une rétrospective est une réunion qui se tient à la fin de chaque itération pour discuter de ce qui est réussi, de ce qui peut être amélioré, et comment intégrer les améliorations tout en conservant ce qui est réussi dans les itérations futures. Les rétrospectives couvrent des éléments comme le processus, les personnes, l'organisation, le relationnel, et les outils. Des réunions de rétrospective, conduites régulièrement pendant les activités appropriées de suivi, sont critiques pour l'auto organisation et l'amélioration continue du développement et des tests.

Les rétrospectives peuvent aboutir à des décisions d'amélioration des tests, focalisées sur leur efficacité, productivité, sur la qualité des cas de test, ainsi que concerner la satisfaction de l'équipe. Elles peuvent également adresser la testabilité des applications, des user Story, des fonctionnalités, ou des interfaces système. Les analyses des causes racines des défauts peuvent conduire à des

© International Software Testing Qualifications Board

améliorations des tests et des développements. En général, les équipes devraient implémenter seulement quelques améliorations par itération. Ceci permet une amélioration continue à un rythme soutenu.

Le timing et l'organisation de la rétrospective dépend de la méthode Agile spécifiquement suivie. Les représentants Métier et l'équipe participent à chaque rétrospective en tant que participants, alors que le facilitateur organise et conduit la réunion. Dans certains cas, l'équipe peut inviter d'autres participants à la réunion.

Les testeurs devraient jouer un rôle important dans les rétrospectives. Les testeurs font partie de l'équipe et apportent leur vision propre. [ISTQB_FL_SYL], Section 1.5. Les tests se font à chaque sprint et contribuent de façon vitale au succès. Tous les membres, testeurs et non testeurs, peuvent apporter des éléments sur les activités de tests et autres.
Les rétrospectives doivent se dérouler dans un environnement professionnel de confiance mutuelle. Les caractéristiques d'une rétrospective réussie sont les mêmes que celles pour tout autre revue comme cela est discuté dans le syllabus niveau Fondation [ISTQB_FL_SYL], Section 3.2.

1.2.4 Intégration Continue

La livraison de l'incrément d'un produit requiert qu'il soit fiable, qu'il fonctionne, et qu'il soit intégré dans le logiciel à la fin de chaque sprint. L'intégration continue permet ce challenge en fusionnant tous les changements faits au logiciel et en intégrant tous les composants modifiés régulièrement, au moins une fois par jour. La gestion de configuration, la compilation, le Build du logiciel, son déploiement, et les tests sont inclus dans un processus simple, automatisé, et répétitif. Puisque les développeurs intègrent leur travail constamment, font des Builds constants, et testent constamment, les défauts dans le code sont détectés plus rapidement.

A la suite du codage des développeurs, du débogage et de la mise sous contrôle du code dans un entrepôt de code source partagé, le processus d'intégration continue inclut les activités automatisées suivantes:

- Analyse statique du code: Exécuter l'analyse statique de code et faire le reporting des résultats
- Compilation: Compiler et lier le code, générer les fichiers exécutables
- Tests unitaires: Exécuter les tests unitaires, vérifier la couverture du code et faire le reporting des résultats des tests
- Déploiement: Installer le Build dans un environnement de tests
- Test d'Intégration: Exécuter les tests d'intégration et faire le reporting des résultats
- Reporting (tableau de bord): Publier le statut de toutes les activités dans un endroit public visible or par email à l'équipe.

Un processus de construction et de test qui est fait sur une base journalière détecte les défauts d'intégration tôt et rapidement. L'intégration continue permet aux testeurs Agile d'effectuer les tests automatisés régulièrement—dans certains cas comme une partie du processus d'intégration continu lui-même —et envoie des feedbacks rapides à l'équipe sur la qualité du code. Ces résultats de test sont visibles à tous les membres de l'équipe spécialement quand des rapports automatisés sont intégrés au processus. Les tests de régression automatisés peuvent être continus tout au long de l'itération qui peut également fournir une construction pas à pas de grands systèmes intégrés le cas échéant. De bons tests de régression automatisés couvrent autant de fonctionnalités que possible, en incluant les user Story délivrées dans les itérations précédentes. Cela libère les testeurs Agile pour qu'ils concentrent leurs tests manuels sur les nouvelles fonctionnalités, les changements implémentés, et les tests de confirmation des défauts corrigés..

En plus des tests automatisés, les organisations utilisant l'intégration continue utilisent typiquement les outils pour implémenter un contrôle qualité continu. En plus de l'exécution des tests unitaires et

© International Software Testing Qualifications Board

d'intégration, ces outils peuvent exécuter des tests statiques et dynamiques additionnels, mesurer et catégoriser la performance, extraire et formater la documentation à partir du code source et faciliter les processus manuels d'assurance qualité. Cette application continue du contrôle qualité a pour but d'améliorer la qualité du produit et de réduire le temps pour le délivrer, en remplaçant la pratique traditionnelle de réalisation des contrôles qualité après que tout le développement soit réalisé.
Les outils de Build peuvent être reliés à des outils de déploiement automatiques, qui peuvent récupérer le Build correct du serveur d'intégration continue ou de Build, et de le déployer dans un ou plusieurs environnements de développement, de test, de démonstration, ou même de production. Cela réduit les erreurs et les délais associés en s'appuyant sur le personnel spécialisé ou les programmeurs pour installer les Release dans ces environnements.
L'intégration continue peut apporter les bénéfices suivants:

- Permettre la détection au plus tôt et faciliter l'analyse des causes racines des problèmes dus à l'intégration et à des changements conflictuels
- Donner un feedback régulier à l'équipe de développement sur comment le code fonctionne.
- Dans une même journée, avoir la même version du logiciel en test que celle en développement
- Réduire les risques de régression associés au raffinement du code du développeur dus à des re-tests rapides du code de base après chaque petit ensemble de changements.
- Assurer que le travail de développement journalier a des bases solides
- Montrer les progrès dans la fabrication du produit, encourager les développeurs et les testeurs
- Éliminer les risques prévisibles associés à l'intégration big-bang
- Fournir une disponibilité constante de logiciels exécutables dans le sprint à des fins d'essai, de démonstration, ou d'éducation
- Réduire les activités de test manuelles et répétitives
- Fournir un feedback rapide sur les décisions prises pour améliorer la qualité et les tests

Cependant, l'intégration continue n'est pas sans risques ni challenges:

- Les outils d'intégration continue doivent être mis en place et maintenus
- Le processus d'intégration continue doit être défini et établi
- L'automatisation des tests requiert des ressources additionnelles et peut être complexe à mettre en place
- Une couverture de test approfondie est essentielle pour que les avantages des tests automatisés soit réels
- Les équipes font parfois trop de confiance aux tests unitaires et font trop peu de tests systèmes et d'acceptation

L'intégration continue requiert l'utilisation d'outils incluant les outils de test, les outils d'automatisation du processus de Build, et les outils de contrôle de version.

1.2.5 Planification de release et d'itérations

Comme mentionné dans le syllabus Fondation [ISTQB_FL_SYL], planifier est une activité continuelle, ce qui est vrai aussi dans le cas des cycles de vie Agile. Pour les cycles de vie agile, il existe deux types de planification, la planification de la release et la planification de l'itération.

La planification de release va jusqu'à la release du produit, et est souvent faite quelques mois avant le début du projet. Le planning de release définit et redéfinit le backlog de produit, et peut impliquer de raffiner des User Story trop grosses en un ensemble de Story plus petites. Le planning de release fournit les bases pour une approche de tests et un plan de test couvrant toutes les itérations. Les plans de Release sont de haut niveau.

© International Software Testing Qualifications Board

Pendant la planification de release, les représentants Métier établissent et priorisent les User Story pour la release, en collaboration avec l'équipe (voir Section 1.2.2). Basés sur ces User Story, les risques projet et qualité sont identifiés et une estimation de l'effort de haut niveau est effectuée (voir Section 3.2).

Les testeurs sont impliqués dans la planification de release et apportent de la valeur ajoutée spécialement dans les activités suivantes:

- Définir des User Story testables, incluant des critères d'acceptation
- Participer aux analyses de risque projet et qualité
- Estimer l'effort de test associé aux User Story
- Définir les niveaux de test nécessaires
- Planifier les tests pour la release

Une fois la planification de release terminée, la planification d'itération pour la première itération commence. La planification d'itération va jusqu'à la fin d'une seule itération et est liée au backlog d'itération.

Pendant la planification de l'itération, l'équipe sélectionne les user Story depuis le backlog de release priorisées, élabore les user Story, effectue une analyse des risques pour les user Story, et estime le travail nécessaire pour chaque User Story. Si une User Story est trop vague, et que les essais pour la clarifier ont échoué, l'équipe peut refuser de l'accepter et utilise la User Story suivante dans l'ordre de priorisation. Les représentants Métier doivent répondre aux questions de l'équipe au sujet de chaque Story de façon à ce que l'équipe puisse comprendre ce qu'ils devraient implémenter et comment tester chaque Story.

Le nombre de Story sélectionné est basé sur la vélocité établie de l'équipe et la taille estimée des user Story sélectionnées. Une fois le contenu de l'itération finalisé, les user Story sont divisées en tâches qui seront prises en charge par les membres de l'équipe appropriés.

Les testeurs sont impliqués dans la planification des itérations et apportent particulièrement de la valeur ajoutée dans les activités suivantes:

- Participer à l'analyse détaillée des risques des user Story
- Déterminer la testabilité des user Story
- Créer les tests d'acceptation pour les user Story
- Diviser les user Story en tâches (particulièrement des tâches de test)
- Estimer l'effort de test pour toutes les tâches de test
- Identifier les aspects fonctionnels et non fonctionnels du système à tester
- Assister et participer à l'automatisation des tests à de multiples niveaux de test

Les plans de release peuvent changer en cours de projet, en incluant des changements sur les user Story individuelles dans le backlog de produit. Ces changements peuvent être dus à des facteurs internes ou externes. Les facteurs internes incluent les capacités à délivrer, la vélocité, et les problèmes techniques. Les facteurs externes incluent la découverte de nouveaux marchés et des opportunités, de nouveaux concurrents, ou des menaces liées au métier qui peuvent changer les objectifs de release et/ou les dates prévues. De plus, les plans d'itération peuvent changer pendant une itération. Par exemple, une User Story particulière qui était considérée relativement simple lors de l'estimation peut s'avérer plus complexe que prévue.

Ces changements peuvent s'avérer difficiles pour les testeurs. Pour planifier les tests, les testeurs doivent avoir une image globale de la release, et pour développer des tests, ils doivent avoir une base de test adéquate avec des oracles de test à chaque itération comme cela est discuté dans le syllabus Fondation [ISTQB_FL_SYL], Section 1.4. Les informations requises doivent être disponibles tôt pour

© International Software Testing Qualifications Board

le testeur, et pourtant les changements doivent être adoptés selon les principes Agile. Ce dilemme requiert des décisions réfléchies au sujet des stratégies de test et de la documentation des tests. Pour plus d'informations sur les challenges des tests Agile, voir [Black09], Chapitre 12.

La planification de release et d'itération devrait adresser la planification des tests tout comme la planification d'activités de développement. Les problèmes particuliers relatifs aux tests incluent :

- Le périmètre du test, l'étendue des tests pour ce périmètre, les objectifs de test, et les raisons de ces décisions.
- Les membres de l'équipe qui prennent en charge les activités de test
- Les environnements de test et les données de test nécessaires, quand elles sont nécessaires, et si aucun ajout ou changement à l'environnement de test et/ou aux données va survenir avant ou pendant le projet.
- Le calendrier, le séquencement, les dépendances, et les pré-requis pour les activités de test fonctionnels et non fonctionnels (Ex: Quelle fréquence d'exécution des tests de régression, quelles fonctionnalité dépendent de telle autre ou de données de test, etc), incluant comment les activités de test sont liées à et dépendent des activités de développement
- Les risques Qualité et projet à adresser (voir Section 3.2.1)
- De plus, la plus grande estimation par l'équipe de l'effort de test devrait considérer le temps et l'effort nécessaire pour réaliser les activités de test requises.

© International Software Testing Qualifications Board

2. Principes, Pratiques, et Processus fondamentaux Agile– 105 mins.

Termes
Build, test de vérification, élément de configuration, gestion de configuration

Objectifs d'apprentissage pour les principes, pratiques et processus de test Agile

2.1 Différences entre les tests traditionnels et les approches Agile

FA-2.1.1 (K2) Décrire les différences entre les activités de test dans les projets Agile et non Agile
FA-2.1.2 (K2) Décrire comment le développement et les activités de codage et de test sont intégrées dans les projets Agile
FA-2.1.3 (K2) Décrire le rôle du test indépendant dans les projets Agile.

2.2 Statut du test dans les projets Agile

FA-2.2.1 (K2) Décrire les techniques et les outils utilisés pour communiquer le statut des tests dans un projet Agile, incluant l'avancement des tests et la qualité du produit
FA-2.2.2 (K2) Décrire le processus d'évolution des tests au travers de multiples itérations et expliquer pourquoi l'automatisation des tests est importante pour gérer le risque de régression dans les projets Agile

2.3 Rôle et Compétences d'un testeur dans une équipe Agile

FA-2.3.1 (K2) Comprendre les compétences (personnelles, Domaine Métier, et en test) d'un testeur dans une équipe Agile
FA-2.3.2 (K2) Comprendre le rôle d'un testeur dans une équipe Agile

© International Software Testing Qualifications Board

2.1 Les différences des tests entre les approches traditionnelles et Agile

Comme décrit dans le syllabus Fondation [ISTQB_FL_SYL] et dans [Black09], les activités de test sont liées aux activités de développement, et donc les tests varient suivant les différents cycles de vie. Les testeurs doivent comprendre les différences entre tester dans un modèle de cycle de vie traditionnel (ex: séquentiel comme le modèle en V ou itératif comme RUP) et les cycles de vie Agile de façon à travailler efficacement et de façon efficiente. Les modèles Agile diffèrent dans la façon dont les activités de test et de développement sont intégrées, dans les produits d'activité des projets, les noms, les critères d'entrée et de sortie utilisés pour différents niveaux de test, l'utilisation d'outils, et la façon dont le test indépendant peut être utilisé efficacement.

Les testeurs devraient se souvenir que les organisations varient considérablement dans l'implémentation de leur cycle de vie. Des déviations des cycles de vie Agile idéaux (voir Section 1.1) peuvent être en fait des personnalisations intelligentes et une adaptation des pratiques. La capacité à s'adapter au contexte d'un projet donné, en incluant les pratiques de développement logiciel réellement suivies, est un facteur clé de succès pour les testeurs.

2.1.1 Activités de test et de développement

Une des différences principales entre les cycles de vie traditionnels et les cycles de vie Agile est l'idée d'avoir des itérations très courtes, chaque itération résultant dans un logiciel opérationnel qui délivre des éléments de valeur aux parties prenantes Métier. Au début du projet, il y a la période de planification de release. Elle est suivie par une suite séquentielle d'itérations. Au début de chaque itération, il y a une période de planification d'itération. Quand le périmètre de l'itération est établi, les user Story sélectionnées sont développées, intégrées dans le système, et testées. Ces itérations sont hautement dynamiques, avec un développement, une intégration, et des activités de test, qui ont lieu tout au long de chaque itération, avec un parallélisme et des chevauchements considérables. Les activités de test se déroulent tout au long de l'itération, et ne sont pas une activité finale.

Les testeurs, développeurs, et les parties prenantes Métier ont tous un rôle concernant le test, tout comme dans les cycles de vie traditionnels. Les développeurs exécutent des tests unitaires puisqu'ils développent les fonctionnalités à partir des user Story. . Les testeurs testent alors ces fonctionnalités. Les parties prenantes Métier testent également les Story pendant l'implémentation. Les parties prenantes Métier peuvent utiliser des cas de test écrits, mais ils peuvent simplement expérimenter la fonction de façon à fournir un feedback rapide à l'équipe de développement.

Dans certains cas, des itérations de renforcement ou de stabilisation se font périodiquement pour résoudre tous les défauts persistants et autres dettes techniques. Cependant, la meilleure pratique est qu'aucune fonctionnalité ne soit considérée comme terminée tant qu'elle n'a pas été intégrée et testée dans le système, [Goucher09]. Une autre bonne pratique est de régler les défauts restant de l'itération précédente au début de l'itération suivante, en tant que parties du backlog de cette nouvelle itération (Se référer à "Corriger les bugs d'abord"). Cependant, certains se plaignent que cette pratique entraine une situation ou l'ensemble du travail à faire dans l'itération est inconnu et qu'il est plus difficile d'estimer les fonctionnalités restantes qui pourront être terminées. A la fin de la suite d'itérations, il peut y avoir un ensemble d'activités pour obtenir un logiciel prêt à être livré, bien que dans certains cas, la livraison soit effectuée à la fin de chaque itération.

Quand le test basé sur les risques est utilisé comme une des stratégies de test, une analyse de risque de haut niveau est effectuée pendant la phase de planification de release, avec les testeurs qui conduisent souvent l'analyse. Cependant, les risques qualité spécifiques associés à chaque itération sont identifiés et évalués dans la planification d'itération. Cette analyse de risque peut influencer la séquence de développement tout comme la priorité et la profondeur de test des fonctionnalités. Elle influence également l'estimation de l'effort de test requis pour chaque fonctionnalité. (voir Section 3.2)

© International Software Testing Qualifications Board

Dans certaines pratiques Agile (ex: Extreme Programming), le travail en binôme est utilisé. Le travail en binôme peut impliquer que les testeurs travaillent à deux pour tester une fonctionnalité. Ce peut également impliquer que le testeur travaille collaborativement avec un développeur pour développer et tester une fonctionnalité. Le travail en binôme peut être difficile quand l'équipe de test est distribuée, mais les processus et les outils peuvent permettre le travail en binôme avec des équipes distribuées. Pour plus d'éléments associés avec le travail distribué, voir [ISTQB_ALTM_SYL], Section 2.8.

Les testeurs peuvent également servir comme coach tests et qualité dans l'équipe, en partageant la connaissance des tests et en prenant en charge l'assurance qualité dans l'équipe. Cela favorise un sentiment d'appropriation collective de la qualité du produit.

L'automatisation des tests à tous niveaux se fait dans beaucoup d'équipes Agile, et cela peut signifier que les testeurs passent du temps à créer, exécuter, surveiller et maintenir les tests automatisés et leurs résultats. A cause de la forte utilisation de l'automatisation des tests, un fort pourcentage de tests manuels sur les projets Agile tend à être réalisé en utilisant les techniques de tests basés sur l'expérience et les défauts comme les attaques logicielles, les tests exploratoires, et l'estimation d'erreurs (voir [ISTQB_ALTA_SYL], Sections 3.3 and 3.4, et [ISTQB_FL_SYL], Section 4.5). Alors que les développeurs se focalisent sur la création de tests unitaires, les testeurs devraient se focaliser sur la création de tests automatisés aux niveaux intégration, système, et intégration système. Cela conduit à une tendance pour les équipes Agile de favoriser les testeurs qui ont un fort background technique et d'automatisation de tests.

Un des principes de base Agile est que le changement peut arriver tout au long du projet. Par conséquent une documentation allégée du produit logiciel est favorisée dans les projets Agile. Les changements de fonctions existantes peuvent avoir des implications concernant les tests, particulièrement pour les tests de régression. L'utilisation de tests automatisés est une façon de gérer la quantité d'effort de test associé au changement. Cependant, il est important que le taux de changement n'excède pas la capacité de l'équipe projet à gérer les risques associés à ces changements.

2.1.2 Produits d'activité des projets

Les produits d'activité des projets qui intéressent directement les testeurs se décomposent en trois catégories:

1. Les produits d'activité orientés Métier qui décrivent le besoin, (ex: spécifications d'exigences) et l'utilisation (ex: Documentation utilisateur)
2. Les produits du développement qui décrivent comment le système est construit (ex: diagrammes d'entité-relation), qui sont mis en œuvre dans le système (ex: code), ou qui évaluent des parties de code individuelles (ex: test unitaires automatisés)
3. Les produits du test qui décrivent comment le système est testé (ex: stratégies et plans de test), qui testent le système (ex: tes manuels et automatisés), ou qui présentent les résultats des tests (ex: tableaux de bord des tests comme discuté dans la Section 2.2.1)

Dans un projet Agile typique, une pratique commune est d'éviter de produire de grandes quantités de documentation. A la place, le focus est fait sur le fait d'obtenir un logiciel opérationnel, avec des tests automatisés qui démontre la conformité aux exigences. (Cet encouragement à réduire la documentation s'applique seulement à la documentation qui n'apporte pas de valeur au client) Dans les projets Agile réussis, une balance est faite entre augmenter l'efficacité en réduisant la documentation, et fournir une documentation suffisante pour permettre les activités des Métiers, de test, de développement, et de maintenance. L'équipe doit prendre une décision pendant la planification de release pour définir quels produits d'activité sont requis et avec quel niveau de documentation.

© International Software Testing Qualifications Board

Les produit du travail orientés Métier dans les projets Agile incluent les user Story et les critères d'acceptation. Dans l'Agilité, les User Story représentent les spécifications d'exigences, et devraient expliquer comment le système doit se comporter, et ce par fonction ou caractéristique simple et cohérente. Une User Story devrait définir une fonctionnalité suffisamment petite pour être réalisée dans une simple itération. Des ensembles conséquents de fonctions connexes, ou un ensemble de sous fonctions qui engendrent une fonctionnalité complexe, peuvent être appelés "Epics". Les Epics peuvent inclure des user Story qui concernent différentes équipes de développement. Par exemple, une User Story peut décrire ce qui est requis au niveau API (middleware) alors qu'une autre décrit ce qui est nécessaire au niveau interface utilisateur (application). Ces ensembles peuvent être développés dans une série de sprints. Chaque Epic et ses user Story devraient avoir des critères d'acceptation associés.

Le travail type des développeurs dans les projets Agile inclut le code. Les développeurs Agile créent également des tests unitaires. Ces tests peuvent être créés après le développement du code. Dans certains cas, les développeurs créent des tests de façon incrémentale, avant d'écrire chaque portion de code, de façon à vérifier qu'il fonctionne comme prévu une fois le code écrit. Bien que cette approche soit appelée « Test First » ou « développement piloté par les tests », en réalité les tests sont plus une forme de spécifications de conception exécutables de bas niveau que des tests [Beck02].

Les produit d'activité des testeurs sur les projets Agile incluent les tests automatisés, tout autant que les documents tels que les plans de test, les catalogues de risques qualité, les tests manuels, les rapports de défauts, et les logs de résultats de test. Ces documents sont conçus pour être les plus légers possible ce qui est souvent également vrai pour ces documents dans les cycles de vie traditionnels. Les testeurs vont alors produire des métriques de test à partir des rapports de défauts et des logs de résultats de test, et cela met encore l'accent sur l'idée d'approche légère.

Dans certaines implémentations Agile, concernant particulièrement les projets et produits de sécurité critique, distribués, ou fortement complexes, une formalisation plus importante de ces produits de travail est requise. Par exemple, certaines équipes transforment les user Story et les critères d'acceptation en des spécifications d'exigences plus formelles. Les rapports de traçabilité verticaux et horizontaux peuvent être préparés pour satisfaire les auditeurs, les règlements, et autres exigences.

2.1.3 Niveaux de Test

Les niveaux de test sont les activités de test liées logiquement, souvent par la maturité ou la complétude de l'objet en test.

Dans les modèles de cycle de vie séquentiels, les niveaux de test sont souvent définis de façon à ce que le critère de sortie d'un niveau soit une partie du critère d'entrée du niveau suivant. Dans certains modèles itératifs, ces règles ne s'appliquent pas parce que les niveaux de test se chevauchent et que les spécifications d'exigences, les spécifications de conception, et les activités de développement peuvent se chevaucher avec les niveaux de tests.

Dans certains cycles de vie Agile, les chevauchements existent car des changements dans les exigences, la conception, peuvent se produire à tout moment d'une itération. Alors que Scrum, en théorie ne permet pas les changements dans les user Story après la planification d'itération, de tels changements surviennent parfois en pratique. Dans une itération, une User Story va suivre séquentiellement les activités de test suivantes:

- Les tests unitaires, typiquement faits par le développeur.
- Les tests d'acceptation des fonctionnalités, qui peuvent être découpés en deux activités.
    - Le test de vérification des fonctionnalités, qui est souvent automatisé, qui peut être fait par les développeurs ou les testeurs, et impliquent que le testeur suive les critères d'acceptation.

© International Software Testing Qualifications Board

- Le test de validation de la fonctionnalité, qui est habituellement manuel et peut impliquer des développeurs, des testeurs et des parties prenantes du Métier dans un travail collaboratif pour déterminer si la fonction est adaptée à l'utilisation, pour améliorer la visibilité de l'avancement fait, et pour recueillir un vrai feedback des parties prenantes Métier.

De plus, il y a souvent un processus parallèle de tests de régression qui se déroule tout au long de l'itération. Cela implique de rejouer les tests unitaires automatisés et les tests de vérification des fonctions de l'itération en cours et des précédentes, habituellement par un Framework d'intégration continue.

Dans certains projets Agile, il peut y avoir un niveau de tests système qui commence une fois que la première User Story est prête pour un tel test. Cela implique d'exécuter des tests fonctionnels, ou non fonctionnels pour la performance, fiabilité, utilisabilité, et d'autres types de test pertinents.

Les équipes Agile peuvent utiliser différentes formes de tests d'acceptation (en utilisant le terme comme expliqué dans le syllabus de niveau Fondation [ISTQB_FL_SYL]). Les alpha tests internes et les beta tests externes peuvent avoir lieu, soit à la fin de chaque itération, après l'achèvement de chaque itération, ou après une série d'itérations. Les tests d'acceptation utilisateur, les tests d'acceptation opérationnels, les tests d'acceptation légaux, et les tests d'acceptation contractuels peuvent également être déroulés, soit à la fin de chaque itération, soit après l'achèvement de chaque itération, ou encore après une série d'itérations.

2.1.4 Tests et gestion de configuration

Les projets Agile impliquent souvent un usage élevé d'outils d'automatisation pour développer les tests, et gérer le développement logiciel. Les développeurs utilisent les outils pour l'analyse statique, les tests unitaires et la couverture de code). Les développeurs vérifient continuellement leur code et les tests unitaires dans un système de gestion de configuration en utilisant des Build automatisés et des Framework de test. Ces frameworks permettent l'intégration continue de nouveaux logiciels dans le système, avec l'analyse statique et les tests unitaires joués répétitivement quand le nouveau logiciel est mis en configuration. [Kubaczkowski].

Ces tests automatisés peuvent également inclure des tests fonctionnels aux niveaux intégration et système. De tests automatisés fonctionnels peuvent être créés en utilisant des harnais de test fonctionnels, des outils de test fonctionnels d'interface homme machine open source, ou des outils commerciaux, et peuvent être intégrés dans les tests automatisés comme faisant partie du Framework d'intégration continue. Dans certains cas, à cause de leur durée, les tests fonctionnels sont séparés des tests unitaires et joués moins fréquemment. Par exemple, les tests unitaires peuvent être joués chaque fois qu'un nouveau logiciel est mis sous contrôle, alors que les tests fonctionnels plus longs sont joués seulement tous les certains nombres de jours.

Un des buts des tests automatisés est de confirmer que le Build fonctionne et est insatiable. Si un test automatisé est en échec, l'équipe devrait corriger le défaut sous-jacent à temps pour la prochaine mise sous contrôle du code. Cela requiert un investissement dans le reporting des tests temps réel pour fournir une bonne visibilité dans les résultats de test. Cette approche aide à réduire des cycles de "Build-installation-échec -reBuild-réinstallation" couteux et inefficaces qui peuvent arriver dans de nombreux projets traditionnels, puisque des changements qui cassent le Build ou causent l'échec de l'installation sont détectés rapidement.

Les tests automatisés et les outils de Build aident à gérer le risque de régression associé aux changements fréquents qui adviennent dans les projets Agile

© International Software Testing Qualifications Board

Cependant, la sur confiance dans les seuls tests unitaires automatisés pour gérer ces risques peut être un problème, car les tests unitaires ont souvent une efficacité de détection de défauts limitée [Jones11]. Les tests automatisés au niveau intégration et systèmes sont également requis.

2.1.5 Options d'organisation avec des tests indépendants

Comme discuté dans le Syllabus niveau Fondation [ISTQB_FL_SYL], les testeurs indépendants sont souvent plus efficaces pour trouver des défauts. Dans certaines équipes Agile, les développeurs créent de nombreux tests sous la forme de tests automatisés. Un ou plusieurs testeurs peuvent être intégrés dans l'équipe, et réaliser nombre des tâches de test. Cependant, étant donné la position du testeur dans l'équipe, il y a un risque de perte d'indépendance et d'évaluation objective.

D'autres équipes Agile restent complètement indépendantes, équipes de test séparées, et des testeurs affectés à la demande pendant les derniers jours de chaque sprint. Cela peut préserver l'indépendance, et ces testeurs peuvent fournir une évaluation objective, non biaisée du logiciel. Cependant, la pression due aux délais, les manques de compréhension dans les nouvelles fonctionnalités du produit, et les problèmes relationnels entre les parties prenantes Métier et les développeurs, entrainent souvent des problèmes avec cette approche.

Une troisième option est d'avoir une équipe de test indépendante et séparée où les testeurs sont affectés aux équipes Agile sur du long terme, en début de projet, ce qui leur permet de garder leur indépendance et de gagner une bonne compréhension du produit et en ayant un relationnel fort avec les autres membres d'équipe. De plus, l'équipe de test indépendante peut avoir des testeurs spécialisés en dehors de l'équipe Agile pour travailler sur du travail à long terme et/ou des activités en dehors de l'itération, comme le développement d'outils de test automatisés, pour effectuer des tests non-fonctionnels, pour créer et maintenir les environnements de test et les données, et pour prendre en charge des niveaux de test qui pourraient ne pas être adaptés à un sprint (ex: test d'intégration système).

2.2 Statuts du test dans les projets Agile

Les changements surviennent rapidement dans les projets Agile. Ces changements signifient que les statuts du test, l'avancement des tests, et la qualité du produit évolue constamment, et que les testeurs doivent trouver des moyens de transmettre l'information de façon à ce que l'équipe puisse prendre des décisions pour rester sur la bonne voie pour réussir avec succès chaque itération. De plus, les changements peuvent affecter des fonctionnalités existantes d'itérations précédentes. Donc, des tests automatisés et manuels doivent être mis à jour pour traiter efficacement les risques de régression.

2.2.1 Communiquer les statuts du test, l'avancement, et la qualité Produit

Les équipes Agile progressent en obtenant un logiciel opérationnel à la fin de chaque itération. Pour déterminer si l'équipe aura un logiciel opérationnel, Elles ont besoin de suivre l'avancement de chaque élément produit dans l'itération et la release. Les testeurs dans les équipes Agile utilisent différentes méthodes pour enregistrer la progression et le statut des tests, incluant l'automatisation des résultats des tests, la progression de ces tâches et des Story dans le tableau de taches Agile, et les Burndown Charts montrant la progression de l'équipe. Cela peut alors être communiqué au reste de l'équipe en utilisant des medias comme les tableaux de bord Wiki, les emails, tout comme verbalement pendant les Stand-Up Meetings. Les équipes Agile peuvent utiliser des outils qui génèrent automatiquement les rapports de statuts basés sur les résultats des tests et de l'avancement des tâches, qui mettent alors à jour des tableaux de bord comme les wikis ou les emails. Cette méthode de communication collecte également des métriques du processus de test, qui peuvent être utilisées dans l'amélioration du processus. Communiquer le statut du processus de test de cette

© International Software Testing Qualifications Board

manière permet aux testeurs de libérer du temps pour se focaliser sur la conception et l'exécution de cas de test supplémentaires.

Les équipes peuvent utiliser les Burndown Charts pour suivre la progression au travers de la release entière et dans chaque itération. Un Burndown Chart [Crispin08] présente la quantité de travail restant à faire par rapport au temps alloué à la release ou à l'itération.

Pour fournir une représentation à un instant donné, détaillée et visuelle du statut actuel de l'équipe intégrée, incluant les statuts du test, les équipes peuvent utiliser le tableau de taches Agile. Les cartes des Story, les tâches de développement, les taches de test, et autres taches créées pendant la planification d'itération (voir Section 1.2.5) sont marquées sur le tableau de tâches, en utilisant souvent des cartes de couleurs coordonnées, pour déterminer le type de tâche. Pendant l'itération, l'avancement est géré via le mouvement de ces tâches au travers du tableau dans des colonnes comme *à faire, Travail en cours*, *vérifié*, and *terminé*. Les équipes Agile peuvent utiliser des outils pour maintenir leurs cartes de Story et les tableaux de tâches Agile, qui peuvent automatiser la mise à jour des statuts et des tableaux de bord.

Les taches de Test sur le tableau de tâches sont liées aux critères d'acceptation définis pour les user Story. Quand les scripts automatisés, les tests manuels, et les tests exploratoires pour une tache de test atteignent un statut "Passé", la tâche est déplacée dans la colonne *Terminé* du tableau de bord. L'équipe intégrée passe en revue le statut de la tâche régulièrement, souvent pendant les stand-up meetings journaliers, pour s'assurer que les tâches se déplacent sur le tableau à un rythme acceptable. Si des tâches (en incluant les tâches de test) ne bougent pas ou trop lentement, l'équipe passe en revue et s'occupe des problèmes qui peuvent bloquer la progression de ces tâches.

Les stand-up meetings journaliers incluent tous les membres de l'équipe Agile en incluant les testeurs. Pendant cette réunion, ils communiquent leur statut en cours. L'ordre du jour de chaque membre est [Agile Alliance Guide]:

- Qu'est-ce que tu as terminé depuis la dernière réunion ?
- Qu'est-ce que tu prévois de terminer pour la prochaine réunion ?
- Comment comptes-tu avancer ?

Tous les problèmes qui peuvent bloquer la progression des tests sont communiqués pendant les stand-up meetings journaliers, comme cela, toute l'équipe est au courant des problèmes et peuvent les résoudre de la bonne façon.

Pour améliorer la qualité du produit complet, de nombreuses équipes Agile font des enquêtes de satisfaction des clients pour obtenir un feedback concernant le fait que le produit atteint les attentes des clients. Les équipes peuvent utiliser d'autres métriques similaires à celles utilisées dans les méthodologies de développement traditionnels, comme les taux de succès/échec, les taux de découverte des défauts, les résultats des tests de confirmation et de régression, la densité de défauts, les défauts trouvés et corrigés, la couverture des exigences, la couverture des risques, la couverture de code, et la modification de code pour améliorer la qualité du produit. Comme dans tous les cycles de vie, les métriques capturées et reportées devraient être pertinentes et aider à la prise de décision. Les métriques ne devraient pas être utilisées pour récompenser, punir, ou isoler des membres de l'équipe.

2.2.2 Gérer les risques de régression en faisant évoluer les cas de test manuels et automatisés

Dans un projet Agile, à la fin de chaque itération, le produit grossit. Donc, le périmètre des tests s'accroit également. Alors que le code change dans l'itération en cours, les testeurs ont également besoin de vérifier qu'aucune régression n'a été introduite dans les fonctionnalités qui ont été développées et testées dans les itérations précédentes. Le risque d'introduire de la régression dans les développements Agile est fortement dû à d'importantes évolutions de code (lignes de code

© International Software Testing Qualifications Board

ajoutées, modifiées, ou effacées d'une version à une autre). Puisque répondre au changement est un principe Agile clé, des changements peuvent également être fait à des fonctionnalités livrées précédemment de façon à satisfaire les besoins Métier. Pour maintenir la vélocité sans augmenter fortement la dette technique, il est critique que l'équipe s'investisse dans l'automatisation des tests à tous les niveaux de test aussi tôt que possible. Il est également critique que tous les tests capitalisés comme les tests automatisés, les cas de test manuels, les données de test, et autres artefacts de test soient mis à jour à chaque itération. Il est hautement recommandé que tous les test capitalisés soient maintenus dans un outil de gestion de configuration de façon à permettre le contrôle des versions, pour assurer un accès aisé à tous les membres de l'équipe , et assurer un support comme requis par les changements de fonctionnalités et préserver l'historique d'information des tests capitalisés.

Parce que la répétition complète de tous les tests est rarement possible, spécialement dans les projets Agile bien délimités dans le temps, les testeurs ont besoin d'allouer du temps à chaque itération pour faire la revue des cas de test manuels et automatisés des itérations précédente et celle en cours pour sélectionner les cas de test qui peuvent être candidats aux suites de tests de régression, et pour supprimer les cas de test qui ne sont plus pertinent. Les tests écrits dans les itérations précédentes pour vérifier des fonctionnalités spécifiques, peuvent avoir une faible valeur dans des itérations suivantes à cause de changements dans les fonctionnalités ou à cause de nouvelles fonctionnalités qui affectent le comportement de fonctionnalités précédentes

En passant en revue les cas de test, les testeurs devraient considérer leur pertinence pour l'automatisation. L'équipe a besoin d'automatiser autant de tests des itérations actuelles et précédentes que possible. Cela permet d'automatiser les tests de régression pour réduire les risques de régression avec moins d'effort que des tests de régression manuels pourraient requérir. Cette réduction de l'effort de test de régression libère du temps aux testeurs pour tester plus complètement les nouvelles fonctionnalités et fonctions de l'itération en cours.

Il est critique que les testeurs aient la possibilité d'identifier rapidement et de mettre à jour les cas de test des itérations précédentes et/ou des releases qui sont affectées par les changements faits dans l'itération en cours. Définir comment l'équipe conçoit, écrit, et stocke les cas de test devraient se passer pendant la planification de release. Les bonnes pratiques de conception de test et d'implémentation ont besoin d'être adoptées tôt et toujours appliquées. Les délais plus courts pour tester et les changements constants dans chaque itération augmenteront l'impact d'une pauvre conception de test et des pratiques d'implémentation.

L'utilisation de l'automatisation des tests, à tous les niveaux de test, permet aux équipes Agile de fournir un feedback rapide sur la qualité produit. Des tests automatisés bien écrits fournissent un document vivant du fonctionnement du système [Crispin08]. En vérifiant les tests automatisés et leurs résultats de test correspondants dans le système de gestion de configuration, aligné avec la gestion de version des produits construits, les équipes Agile peuvent faire la revue des fonctionnalités testées et les résultats de test pour tout Build à n'importe quel moment dans le temps.

Les tests unitaires automatisés sont exécutés avant que le code source soit vérifié dans le circuit principal du système de gestion de configuration pour assurer que les changements de code ne cassent pas le Build du logiciel construit. Pour réduire les cassures de Build, qui peuvent ralentir la progression de l'équipe intégrée, le code ne devrait pas être mis sous contrôle à moins que tous les tests automatisés ne passent. Les résultats des tests unitaires automatisés fournissent un feedback immédiat sur la qualité du code et du Build, mais pas sur la qualité du produit.

Les tests d'acceptation automatisés sont exécutés régulièrement comme une partie de l'intégration continue du système complet construit. Ces tests sont exécutés sur un système complet construit quotidiennement, mais ne sont généralement pas exécutés à chaque mise sous contrôle du code car ils prennent plus longtemps à exécuter que les tests unitaires automatisés et pourraient ralentir les mises sous contrôle du code. Les résultats des tests d'acceptation automatisés fournissent du

© International Software Testing Qualifications Board

feedback sur la qualité du produit concernant la régression depuis le dernier Build, mais ils ne fournissent pas de statut sur la qualité complète du produit.

Les tests automatisés peuvent être exécutés continuellement sur le système. Un sous ensemble initial de tests automatisés pour couvrir les fonctionnalités et les points d'intégration critiques du système devrait être créé immédiatement après qu'un nouveau Build soit déployé dans l'environnement de test. Ces tests sont communément connus comme des tests de vérification du Build. Les résultats des tests de vérification de Build fourniront un feedback instantané sur le logiciel après déploiement, de façon à ce que les équipes ne perdent pas de temps à tester un Build instable.

Les tests automatisés contenus dans l'ensemble des tests de régression sont généralement exécutés comme une partie principale de construction du Build journalier dans l'environnement d'intégration continue, et encore quand un nouveau Build est déployé dans l'environnement de test. Dès qu'un test de régression automatisé est en échec, l'équipe arrête et recherche les raisons de la défaillance du test. Les tests peuvent être en défaillance à cause de changements fonctionnels légitimes dans l'itération en cours, auquel cas les tests et/ou les User Story peuvent nécessiter d'être mis à jour pour refléter les nouveaux critères d'acceptation. Alternativement, les tests peuvent nécessiter d'être retirés si un autre test a été fait pour couvrir les changements. Cependant, si le test est en échec à cause d'un défaut, la bonne pratique pour l'équipe est qu'elle corrige le défaut avant d'avancer avec de nouvelles fonctionnalités.

En plus de l'automatisation des tests, les taches de test suivantes peuvent également être automatisées:

- Génération de données de Test
- Chargement de données de test dans les systèmes
- Déploiement de Build dans l'environnement de test
- Restauration d'un environnement de test (ex: la base de données ou les fichiers de données des sites web) à une base de référence (baseline)
- Comparaison de données de sortie

L'automatisation de ces tâches réduit les frais généraux et permet à l'équipe de passer du temps à développer et à tester des nouvelles fonctionnalités.

2.3 Rôles et compétences d'un testeur dans une équipe Agile

Dans une équipe Agile, les testeurs doivent collaborer de façon rapprochée avec tous les autres membres de l'équipe et avec les parties prenantes Métier. Cela entraine de nombreuses implications en termes des compétences qu'un testeur doit avoir et les activités qu'il réalise dans une équipe Agile.

2.3.1 Compétences d'un testeur Agile

Les testeurs Agile devraient avoir les toutes les compétences mentionnées dans le Syllabus niveau Fondation [ISTQB_FL_SYL]. En plus de ces compétences, un testeur dans une équipe Agile devrait être compétent en automatisation des tests, développement piloté par les tests, développement piloté par les tests d'acceptation, et en tests boite blanche, boite noire et en tests basés sur l'expérience.

Comme les méthodologies Agile dépendent grandement de la collaboration, de la communication, et de l'interaction entre les membres de l'équipe tout autant qu'avec les parties prenantes en dehors de l'équipe, les testeurs dans une équipe Agile devraient avoir de bonnes compétences inter personnelles. Les testeurs dans une équipe Agiles devraient:

- Etre positifs et orientés solution avec les membres de l'équipe et les parties prenantes
- Afficher un esprit critique, être orientés qualité, être septiques au sujet du produit

© International Software Testing Qualifications Board

- Acquérir activement de l'information des parties prenantes (plutôt que de se fier entièrement aux spécifications écrites)
- Evaluer précisément et reporter les résultats, l'avancement des tests, et la qualité produit
- Travailler efficacement pour définir des user Story testables, spécialement les critères d'acceptation, avec les représentants des clients et les parties prenantes
- Collaborer avec l'équipe, travailler en binôme avec les programmeurs et autres membres de l'équipe
- Répondre au changement rapidement, incluant le changement, ajoutant ou améliorant les cas de test
- Planifier et organiser leur propre travail

Une croissance continue des compétences, incluant celles des compétences inter personnelles, est essentielle pour tous les testeurs, incluant ceux des équipes Agiles.

2.3.2 Le Rôle d'un testeur dans une équipe Agile

Le rôle d'un testeur dans une équipe Agile inclut des activités qui génèrent et fournissent du feedback non seulement sur les statuts des tests, avancement des tests, et qualité produit, mais également sur la qualité du processus. En plus des activités décrites partout dans ce syllabus, ces activités incluent:

- Comprendre, implémenter, et mettre à jour la stratégie de test
- Mesurer et reporter la couverture de test au travers de toutes les dimensions de couverture applicables
- Assurer d'une utilisation correcte des outils de test
- Configurer, utiliser, et gérer les environnements de test et les données de test
- Reporter les défauts et travailler avec l'équipe pour les résoudre
- Coacher les autres membres de l'équipe sur les aspects pertinents du test
- Assurer que les tâches de test appropriées sont planifiées pendant les planifications de release et d'itération
- Collaborer activement avec les développeurs, les parties prenantes Métier, et les Product Owners pour clarifier les exigences, spécialement en termes de testabilité, consistance, et de complétude
- Participer pro-activement aux rétrospectives d'équipe, suggérer et implémenter des améliorations

Dans une équipe Agile, chaque membre de l'équipe est responsable de la qualité produit et joue un rôle en réalisant des tests liés aux tâches.

Les organisations Agile peuvent rencontrer des risques liés aux organisations de test:

- Les testeurs travaillant très proches des développeurs perdent leur mentalité propre de testeur
- Les testeurs deviennent tolérants ou silencieux au sujet de l'inefficience, de l'inefficacité, ou du faible niveau des pratiques de l'équipe
- Les testeurs ne peuvent pas garder le rythme avec des changements qui arrivent dans des itérations contraintes en temps

Pour mitiger ces risques, les organisations peuvent considérer différentes solutions pour préserver l'indépendance discutée dans la Section 2.1.5.

© International Software Testing Qualifications Board

3. Méthodes, Techniques, et outils pour les tests Agile – 480 mins.

Termes
critères d'acceptation, tests exploratoires, tests de performance, risques produit, risques qualité, tests de régression, approche de test, charte de test, estimation des test, automatisation de l'exécution de test, stratégie de test, développement piloté par les tests, Framework de tests unitaires

Objectifs d'apprentissage pour Méthodes, Techniques, et Outils des tests Agile

3.1 Méthodes de test Agile

FA-3.1.1 (K1) Rappeler les concepts du développement piloté par les tests, développement piloté par les tests d'acceptation, et développement piloté par le comportement
FA-3.1.2 (K1) Rappeler les concepts de la pyramide des tests
FA-3.1.3 (K2) Résumer les quadrants des tests et leur relation avec les niveaux et les types de test
FA-3.1.4 (K3) Pour un projet Agile donné, pratiquer le rôle d'un testeur dans une équipe Scrum

3.2 Évaluation des risques de qualité et estimation de l'effort de test

FA-3.2.1 (K3) Evaluer les risques qualité dans un projet Agile
FA-3.2.2 (K3) Estimer l'effort de test basé sur le contenu d'une itération et les risques qualité

3.3 Techniques dans les projets Agile

FA-3.3.1 (K3) Interpréter les informations pertinentes pour soutenir les activités de test
FA-3.3.2 (K2) Expliquer aux parties prenantes Métier comment définir des critères d'acceptation testables
FA-3.3.3 (K3) Pour une User Story donnée, écrire les cas de test de développement piloté par les tests d'acceptation
FA-3.3.4 (K3) Pour des comportements à la fois fonctionnels et non-fonctionnels, écrire des cas de test utilisant des techniques de conception de test boite noire basés sur des user Story données
FA-3.3.5 (K3) Effectuer des tests exploratoires pour soutenir les tests d'un projet Agile

3.4 Outils dans les projets Agile

FA-3.4.1 (K1) Rappeler les différents outils disponibles pour les testeurs associés à leur sujet et aux activités des projets Agile

© International Software Testing Qualifications Board

3.1 Méthodes de test Agile

Il existe certaines pratiques de test qui peuvent être suivies dans tous les projets de développement (agile ou non) pour produire des produits de qualité. Cela inclut d'écrire des tests en avance pour exprimer un comportement propre, se focaliser sur la prévention au plus tôt des défauts, la détection, et la suppression, et s'assurer que les types de test corrects sont exécutés au bon moment et comme partie du bon niveau de test. Ceux qui pratiquent l'Agile, s'attachent à introduire ces pratiques tôt.
Les testeurs dans les cycles de vie Agile, jouent un rôle clé en guidant l'utilisation de ces pratiques de test dans le cycle de vie.

3.1.1 Développement piloté par les tests, Développement piloté par les tests d'Acceptation, et Développement piloté par les tests de Comportement

Le développement piloté par les tests, les développements pilotés par les tests d'acceptation, et le développement piloté par les tests de comportement sont trois techniques complémentaires utilisées dans les équipes Agile pour prendre en charge les tests des différents niveaux de test. Chaque technique est un exemple des principes fondamentaux de test, le bénéfice des tests au plus tôt et des activités de QA, puisque les tests sont définis avant que le code ne soit écrit.

Développement piloté par les tests (Test Driven Development)
Le développement piloté par les tests (TDD) est utilisé pour développer le code en étant guidé par des cas de test automatisés. Le processus pour le développement piloté par les tests est:

- Ajouter un test qui reproduit ce que le programmeur conçoit du fonctionnement désiré d'une petite partie de code
- Exécuter le test, qui devrait être en échec, puisque le code n'existe pas
- Ecrire le code et exécuter le test en boucle jusqu'à ce que le test passe
- Refactoriser le code après que le test soit passé, ré-exécuter les tests pour assurer qu'il continue de passer après la refactorisation du code
- Répéter ce processus pour la petite partie de code suivante, en exécutant les tests précédents tout autant que les tests ajoutés

Les tests écrits sont principalement de niveau unitaire et sont focalisés sur le code, bien que les tests puissent également être écrits au niveau intégration ou système. Le développement piloté par les tests a gagné sa popularité par l'Extreme Programming [Beck02], mais est également utilisé dans d'autres méthodologies Agile et parfois dans les cycles de vie séquentiels.

Le développeur se focalise sur des résultats attendus clairement définis. Les tests sont automatisés et sont utilisés en intégration continue.

Développement piloté par les tests d'acceptation (Acceptance Test Driven Developement)
Le développement piloté par les tests d'acceptation [Adzic09]. Il définit des critères d'acceptation pendant la création des user Story (voir Section 1.2.2). Le développement piloté par les tests d'acceptation est une approche collaborative qui permet à chaque partie prenante de comprendre comment le composant logiciel doit se comporter et ce que les développeurs, les testeurs, et les représentants Métier ont besoin pour assurer ce comportement. Le processus de développement piloté par les tests d'acceptation est expliqué en Section 3.3.2.

Le développement piloté par les tests d'acceptation génère des tests réutilisables pour les tests de régression. Des outils spécifiques soutiennent la création et l'exécution de tels tests, souvent dans le processus d'intégration continue. Ces outils peuvent se connecter aux données et à la couche service de l'application, ce qui permet aux tests d'être exécutés au niveau système ou acceptation. Le développement piloté par les tests d'acceptation permet une correction rapide des défauts et la validation du comportement de la fonctionnalité. Cela aide à déterminer si les critères d'acceptation sont atteints pour la fonctionnalité.

© International Software Testing Qualifications Board

Développement piloté par les tests de comportement (Behavior-Driven Development)
Le Développement piloté par les tests de comportement [Chelimsky10], permet à un développeur de se focaliser sur le test du code en cours de développement en se focalisant sur le comportement attendu du logiciel, de façon à rendre les tests plus facile à comprendre par les autres membres de l'équipe et les parties prenantes .
Des Frameworks spécifiques de développement piloté par les tests de comportement peuvent être utilisés pour définir des critères d'acceptation basés sur le format *Given (*Etant donné) / *When* (quand) / *Then* (alors):

Etant donné un contexte initial,
Quand un évènement intervient,
Alors cela entraine des résultats.

A partir de ces exigences, le Framework de développement piloté par le comportement génère du code qui peut être utilisés par les développeurs pour créer des cas de test. Le développement piloté par le comportement aide le développeur à collaborer avec les autres parties prenantes, en incluant les testeurs, pour définir des tests unitaires focalisés sur les besoins Métier.

3.1.2 La Pyramide des tests

Un système logiciel peut être testé à différents niveaux. Typiquement les niveaux de test sont, du plus bas au plus haut, unitaires, intégration, système, et acceptation (voir [ISTQB_FL_SYL] Section 2.2). La pyramide des tests souligne qu'il y a un nombre important de tests aux niveaux les plus bas (en bas de la pyramide) et, plus le développement avance vers les niveaux hauts, plus le nombre de tests décroit (haut de la pyramide). Habituellement les niveaux de test unitaires et d'intégration sont automatisés et créés en utilisant des outils basés sur des API. Aux niveaux système et acceptation, les tests automatisés sont créés en utilisant des outils basés sur les IHM. Le concept de la pyramide des tests est basé sur le principe de la qualité logicielle et des tests au plus tôt (i.e., en éliminant les défauts aussi tôt que possible dans le cycle de vie).

3.1.3 Quadrants de Test, Niveaux de Test, et type de Test

Les quadrants des tests, définis par Brian Marick [Crispin08], permettent d'aligner les niveaux de tests avec les types de test appropriés dans la méthodologie Agile. Le modèle des quadrants de test, et ses variantes, aident à assurer que tous les types de test importants et tous les niveaux de test sont inclus dans le cycle de développement. Ce modèle fournit également une façon de différentier et de décrire les types de test à toutes les parties prenantes, incluant les développeurs, les testeurs, et les représentants Métier.

Dans les quadrants de test, les tests peuvent être orientés Métier (utilisateur) ou technologie (développeur). Certains tests concernent le travail fait par les équipes Agile et concernent le comportement du logiciel. D'autres tests peuvent vérifier le produit. Les tests peuvent être complètement manuels, complètement automatisés, une combinaison de manuels et automatisés, ou manuels mais assistés par des outils. Les quatre quadrants sont comme ci-dessous:

- Le Quadrant Q1 est de niveau unitaire, orienté technologie, et concerne les développeurs. Ce quadrant contient des tests unitaires. Ces tests devraient être automatisés et dans le processus d'intégration continue.
- Le Quadrant Q2 est au niveau système, orienté Métier, et confirme le comportement du produit. Ce quadrant contient des tests fonctionnels, des tests basés sur les exemples, les tests des Story, des prototypes d'expérience utilisateur, et des simulations. Ces tests vérifient les critères d'acceptation, et peuvent être manuels ou automatisés. Ils sont souvent créés pendant le développement des User Story et ainsi améliorent la qualité des Story. Ils sont utilisés lors de la création des suites de tests automatisés.

© International Software Testing Qualifications Board

- Le Quadrant Q3 est au niveau système ou acceptation utilisateur, orienté Métier, et contient des tests qui critiquent le produit, en utilisant des scenarios et des données réalistes. Ce quadrant contient les tests exploratoires, les scenarios, les flots de processus, les tests d'utilisabilité, d'acceptation utilisateur, les alpha test et les beta tests. Ces tests sont souvent manuels et sont orientés utilisateur.
- Le Quadrant Q4 est au niveau système ou acceptation opérationnelle, orienté technologie, et contient des tests qui critiquent le produit. Ce quadrant contient des tests de performance, de charge, de stress, et d'évolutivité, de sécurité de maintenabilité, de gestion de la mémoire, de compatibilité et d'interopérabilité, de migration de données, d'infrastructure, et de tests de restauration. Ces tests sont souvent automatisés.

Durant toute itération, des tests de certains ou de tous les quadrants peuvent être requis. Les quadrants de test s'appliquent aux tests dynamiques plutôt qu'aux tests statiques.

3.1.4 Le Rôle d'un Testeur

Au travers de ce syllabus, des références générales ont été faites aux méthodes et techniques Agile, ainsi que sur le rôle d'un testeur dans différents cycles de vie Agile. Cette sous-section concerne spécifiquement le rôle d'un testeur dans un projet suivant un cycle de vie Scrum [Aalst13].

Travail d'équipe

Le travail d'équipe est un principe fondamental du développement Agile. Agile met en avant l'approche de l'équipe intégrée constituée de développeurs, testeurs, et des représentants Métier qui travaillent ensemble. Les éléments suivant sont les meilleures pratiques organisationnelles et comportementales des équipes Scrum:

- Transversalité: Chaque membre de l'équipe apporte un ensemble de compétences différentes à l'équipe. L'équipe travaille ensemble sur la planification de la stratégie de test, la spécification des tests, l'exécution des tests, l'évaluation des tests, et le reporting des résultats de test.
- Auto-organisation: L'équipe peut être constituée seulement de développeurs, mais, comme noté dans la Section 2.1.5, idéalement il y aurait un, deux, ou plus de testeurs.
- Co-location: Les testeurs sont assis avec les développeurs et le Product Owner.
- Collaboration: Les testeurs collaborent avec les membres de leur équipe, d'autres équipes, les parties prenantes, le Product Owner, et le Scrum master.
- Habilités: Les décisions techniques concernant la conception et les tests sont faites par l'équipe dans son ensemble (développeurs, testeurs, et Scrum master), en collaboration avec le Product Owner et d'autres équipes si nécessaire.
- Engagés: Le testeur s'engage à questionner et à évaluer les comportements du produit et ses caractéristiques, en ce qui concerne les attentes et les besoins des clients et des utilisateurs.
- Transparents: L'avancement du développement et des tests est visible sur le tableau de tâches Agile (voir Section 2.2.1).
- Crédibles: Les testeurs doivent assurer la crédibilité de la stratégie de test, dans son implémentation, et exécution, sinon les parties prenantes n'auront pas confiance dans les résultats des tests. Cela se fait souvent en fournissant de l'information sur le processus de test aux parties prenantes.
- Ouverts au feedback: Le feedback est un aspect important de succès d'un projet, spécialement dans les projets Agile. Les rétrospectives permettent aux équipes d'apprendre de leurs succès et échecs.
- Résistants: Le test doit être capable de répondre aux changements, comme toutes les autres activités des projets Agile.

Ces bonnes pratiques maximisent la probabilité de tester avec succès dans les projets Scrum.

© International Software Testing Qualifications Board

Sprint Zéro

Le Sprint zéro est la première itération du projet dans laquelle de nombreuses activités de préparation ont lieu (voir Section 1.2.5). Le testeur collabore avec l'équipe sur les activités suivantes durant cette itération:

- Identifier le périmètre du projet (Ex: le backlog de produit)
- Créer une architecture du système initiale et des prototypes de haut niveau
- Planifier, acquérir, et installer les outils nécessaires (ex: pour la gestion des tests, la gestion des défauts, l'automatisation des tests, et l'intégration continue)
- Créer une stratégie de test initiale pour tous les niveaux de test concernant le périmètre des tests (entre autres sujets): les risques techniques, les types de test (voir Section 3.1.3), et les objectifs de couverture
- Faire une analyse initiale de risques qualité (voir Section 3.2.1)
- Définir des métriques de test pour mesurer l'avancement du processus de test, l'avancement des tests dans le projet, et la qualité du produit
- Spécifier la définition de "terminé"
- Créer le Task Board (voir Section 2.2.1)
- Définir quand continuer ou stopper les tests avant de délivrer le système au client.

Le Sprint zéro donne la direction vers quoi le test a besoin de réaliser et comment les tests doivent le réaliser au cours des sprints.

Intégration

Dans les projets Agile, l'objectif est de délivrer au client de la valeur sur une base continue (de préférence à chaque sprint). Pour permettre ceci, la stratégie d'intégration devrait considérer à la fois la conception et le test.

Pour permettre à une stratégie de test continue de délivrer des fonctionnalités et des caractéristiques, il est important d'identifier toutes les dépendances entre les fonctions sous-jacentes et les fonctionnalités.

Planification des tests

Puisque les tests sont complètement intégrés dans l'équipe Agile, la planification des tests devrait commencer pendant la session de planification de release et mise à jour pendant chaque sprint. La planification des tests pour la release et chaque sprint devrait concerner les points discutés en Section 1.2.5

La planification de sprint résulte en un ensemble de tâches à mettre dans le tableau des tâches, ou chaque tâche devrait avoir une durée de un à deux jours. De plus, tout problème lié au test devrait être suivi conforme à la check liste ci-dessus pour maintenir un flux constant de test.

Pratiques de Test Agile

De nombreuses pratiques peuvent être utiles pour les testeurs d'une équipe Scrum, dont certaines incluent:

- Binômes: Deux membres d'équipe (ex: un testeur et un développeur, deux testeurs, ou un testeur et un Product Owner) sont assis ensemble à une station de travail pour réaliser des tâches de test ou autres dans le Sprint.

Conception de test incrémentale: Les cas de test et les chartes de test sont créés progressivement à partir des user Story et autres bases de test, en commençant avec de simples tests et en avançant avec de plus complexes.

© International Software Testing Qualifications Board

Mind-mapping: Le Mind-mapping est un outil utile pour tester [Crispin08]. Par exemple, les testeurs peuvent utiliser le mind-mapping pour identifier quelles sessions de test réaliser, pour afficher les stratégies de test, et décrire les données de test.

Ces pratiques sont en addition à d'autres pratiques, discutées dans ce syllabus et dans le chapitre 4 du syllabus niveau fondation [ISTQB FL SYL].

3.2 Evaluer les risques Qualité et estimer l'effort de Test

Un objectif typique de test de tous projets, Agiles ou traditionnels, est de réduire le risque de problème de qualité Produit un niveau acceptable avant de faire la release. Les testeurs dans les projets Agile peuvent utiliser les mêmes types de techniques utilisées sur les projets traditionnels pour identifier les risques qualité (ou risques Produit), évaluer les niveaux de risque associés, estimer l'effort requis pour réduire suffisamment ces risques, et alors mitiger ces risques au travers de la conception, l'implémentation et l'exécution des tests. Cependant, étant donné les itérations courtes et le taux de changement dans les projets Agile, quelques adaptations à ces techniques sont requises.

3.2.1 Evaluer les risques de qualité sur les Projets Agile

Un de nombreux enjeux dans les tests est la sélection, l'affectation, et la priorisation correcte des conditions de test. Cela inclut déterminer la quantité appropriée d'effort à allouer de façon à couvrir chaque condition avec des tests, et en séquençant les tests qui en résultent de façon à optimiser l'efficacité et l'efficience des tests à faire. L'identification, l'analyse et les stratégies de mitigation des risques, peut être utilisée par les testeurs des équipes Agile pour aider à déterminer un nombre de cas de tests acceptable à exécuter, bien que des contraintes et des variables qui interagissent peuvent requérir des compromis.

Le risque est la possibilité d'un résultat ou d'un évènement négatif ou non désiré. Le niveau de risque est trouvé en évaluant la probabilité de l'occurrence du risque et de son impact. Quand l'effet principal du problème potentiel est sur la qualité du produit, les problèmes potentiels sont appelés risques qualité ou risques produit. Quand l'effet principal du problème potentiel concerne le succès du projet, les problèmes potentiels sont appelés risques projet ou risques de planification [Black07] [vanVeenendaal12].

Dans les projets Agile, l'analyse des risques qualité se fait à deux moments.

- Planification de release: les représentants Métier qui connaissent les fonctionnalités de la release fournissent une vue générale des risques de haut niveau, et l'équipe complète, incluant le testeur(s), peut participer à l'identification et l'évaluation des risques.
- Planification d'itération: l'équipe intégrée identifie et évalue les risques qualité.

Des exemples de risques qualité pour un système incluent:

- Les calculs incorrects dans des rapports (un risque fonctionnel lié à la précision)
- Des réponses lentes aux entrées utilisateur (un risque non-fonctionnel lié à l'efficience et aux temps de réponse)
- Des difficultés à comprendre les écrans et les champs (un risque non fonctionnel relatif à la capacité d'utilisation et à la capacité de compréhension)

Comme mentionné plus tôt, une itération commence avec la planification d'itération, qui aboutit à l'estimation des tâches sur le tableau de tâches. Ces tâches peuvent être priorisées en parties basées sur le niveau de qualité du risque associé à la tâche. Les tâches associées avec un risque plus élevé devraient commencer tôt et impliquer le moins d'effort de test. Les tâches associées avec des risques plus faibles devraient commencer plus tard et impliquer moins d'effort de test.

© International Software Testing Qualifications Board

Un exemple de comment le processus d'analyse des risques qualité dans un projet Agile peut se dérouler pendant la planification d'itération est décrit dans les étapes suivantes:

1. Rassembler les membres de l'équipe Agile, incluant le(s) testeur(s)
2. Lister tous les éléments du backlog pour l'itération en cours (ex., sur un tableau de tâches)
3. Identifier les risques qualité associés à chaque élément, en considérant toutes les caractéristiques qualités pertinentes
4. Evaluer chaque risque identifié, ce qui inclut deux activités: catégoriser le risque et déterminer son niveau de risque basé sur l'impact et la probabilité de défaut
5. Déterminer la portée du test proportionnellement au niveau de risque
6. Sélectionner le(s) technique(s) de test appropriées) pour mitiger chaque élément de risque basé sur le risque, le niveau de risque, et les caractéristiques qualité pertinentes.

Le testeur conçoit ainsi, implémente, et exécute des tests pour mitiger les risques. Cela inclut la totalité des caractéristiques, comportements, caractéristiques qualité, et attributs qui affectent la satisfaction du client, de l'utilisateur, et des parties prenantes.

Au travers du projet, l'équipe devrait rester à l'écoute de nouvelles informations qui peuvent changer l'ensemble des risques et/ou le niveau de risque associé aux risques qualité connus. Un ajustement périodique de l'analyse du risque produit, qui résulte en ajustements pour les tests, devrait se faire. Les ajustements incluent d'identifier les nouveaux risques, de réévaluer les niveaux de risque existants, et d'évaluer la productivité des activités de mitigation des risques.

Les risques Qualité peuvent être également mitigés avant que l'exécution des tests ne commence. Par exemple, si des problèmes avec les user Story sont trouvés pendant l'identification des risques, l'équipe projet peut revoir en profondeur les User Story comme actions de mitigation des risques.

3.2.2 Estimer l'effort de test basé sur le contenu et les risques

Durant la planification de release, l'équipe Agile estime l'effort requis pour terminer la release. L'estimation concerne également l'effort de test. Une technique d'estimation commune utilisée dans les projets Agile est le planning Poker, une technique basée sur le consensus. Le Product Owner ou le client lit une User Story aux évaluateurs. Chaque évaluateur a un jeu de cartes avec des valeurs similaires à la séquence de Fibonacci (EX 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89…) ou toute autre choix de progression. (Ex: Les tailles de T-shirt allant de extra-petit à extra-large). Les valeurs représentent le nombre de points de Story, de jours d'effort, ou d'autres unités que l'équipe utilise pour estimer. La séquence de Fibonacci est recommandée car les nombres de la séquence reflètent l'incertitude qui augmente proportionnellement avec la taille de l'histoire. Une estimation haute signifie habituellement que la Story n'est pas bien comprise ou devrait être divisée dans de multiples Story plus petites.
Les évaluateurs discutent de la fonctionnalité, et posent des questions au Product Owner tant qu'il y en a besoin. Les aspects comme le développement et l'effort de test, la complexité de la Story, et le périmètre du test jouent un rôle dans l'estimation. Donc il est conseillé d'inclure le niveau de risque d'un élément du backlog, en addition à la priorité spécifiée par le Product Owner avant que la session de planning poker ne soit initialisée. Quand la fonctionnalité a été complètement discutée, chaque évaluateur sélectionné personnellement une carte pour représenter son estimation. Toutes les cartes sont ainsi révélées en même temps. Si tous les évaluateurs sélectionnent la même valeur, cela devient l'estimation. Si non, les évaluateurs discutent les différences entre les estimations après quoi la session de poker est répétée jusqu'à ce qu'un accord soit atteint, soit par consensus, ou en appliquant des règles (ex: utiliser la valeur médiane ou le score le plus haut) pour limiter le nombre de sessions. Ces discussions assurent une estimation fiable de l'effort nécessaire pour réaliser les éléments du backlog produit requis par le Product Owner et aide à améliorer la connaissance collective de ce qui doit être fait [Cohn04].

© International Software Testing Qualifications Board

3.3 Techniques dans les projets Agile

De nombreuses techniques et niveaux de test qui s'appliquent aux projets traditionnels peuvent également être appliqués aux projets Agile. Cependant, pour les projets Agile, il faut considérer des différences dans les techniques de test, terminologies, et documentation qui devraient être pris en considération.

3.3.1 Critères d'acceptation, adéquation de la couverture, et autres informations pour les Tests

Les projets Agile précisent les exigences initiales, comme les user Story dans un backlog priorisés en début de projet. Les exigences initiales sont courtes et suivent habituellement un format prédéfini (voir Section 1.2.2). Les exigences Non-fonctionnelles, comme l'utilisabilité, la performance, sont également importantes et peuvent être spécifiées comme des user Story uniques ou connectées aux autres user Story fonctionnelles. Les exigences Non-fonctionnelles peuvent suivre un format prédéfini ou standard, comme [ISO25000], ou un standard industriel spécifique.

Les user Story sont utilisées comme des premières bases de test importantes. D'autres bases de test possibles incluent:

- L'expérience de projets précédents
- Les fonctions existantes, fonctionnalités, et caractéristiques qualité du système
- Le code, l'architecture et la conception
- Les profils utilisateur (contexte, configuration système, et comportement de l'utilisateur)
- L'information sur les défauts de projets existants ou passés
- Une catégorisation des défauts dans une taxonomie des défauts
- Les standards applicables (ex., [DO-178B] pour les logiciels avioniques)
- Les risques Qualité (voir Section 3.2.1)

Durant chaque itération, les développeurs créent le code qui implémente les fonctions et les fonctionnalités décrites dans les User Story, avec les caractéristiques qualité pertinentes, et ce code est vérifié et validé via les tests d'acceptation. Pour être testables, les critères d'acceptation doivent adresser les éléments suivant quand pertinent [Wiegers13]:

- Comportement Fonctionnel: Le comportement observable de l'extérieur avec les actions utilisateur comme entrées de ce qui est fait avec certaines configurations.
- Caractéristiques Qualité: Comment le système réalise le comportement spécifié. Les caractéristiques peuvent également être nommées attributs qualité ou exigences no-fonctionnelles. Les caractéristiques qualités communes sont la performance, la fiabilité, la capacité d'utilisation, etc.
- Scénarios (cas d'utilisation): Une séquence d'actions entre un acteur externe (souvent un utilisateur) et le système, de façon à atteindre un but spécifique ou accomplir une tâche Métier.
- Règles Métier: Activités qui peuvent seulement être réalisées dans le système sous certaines conditions définies par des procédures externes et des contraintes (ex., Les procédures utilisées par une compagnie d'assurance pour traiter les demandes d'assurance).
- Interfaces externes: Description des connexions entre le système à développer et le monde extérieur. Les interfaces externes peuvent être divisées en différents types (interface utilisateur, interfaces vers d'autres systèmes, etc.).
- Contraintes: Toute contrainte de conception et d'implémentation qui va restreindre les options pour le développeur. Les composants avec logiciel embarqué doivent souvent respecter des contraintes physiques comme la taille, le poids, et les interfaces de connexion.

© International Software Testing Qualifications Board

- Définitions de données: Les clients peuvent décrire le format, les types de données, les valeurs autorisées, et les valeurs par défaut pour une donnée dans la composition d'une structure complexe de données Métier (ex., le code postal d'un adresse postale).

En plus des User Story et de leurs critères d'acceptation associés, d'autres informations sont pertinentes pour le testeur, incluant:

- Comment le système est supposé fonctionner et être utilisé
- Les interfaces systèmes qui peuvent être utilisées/accédées pour tester le système
- Si l'aide apportée par les outils actuels est suffisante
- Si le testeur a suffisamment de connaissances et de compétences pour réaliser les tests nécessaires

Les testeurs vont souvent découvrir un besoin d'informations supplémentaires (ex: La couverture de code) au travers des itérations et devraient travailler collaborativement avec le reste de l'équipe Agile pour obtenir cette information. La pertinence des informations fait partie de la détermination pour savoir si une activité particulière peut être considérée comme terminée. Ce concept de définition de "terminé" est critique dans les projets Agile, et s'applique de différentes façons comme cela est discuté dans les sous-sections suivantes.

Niveaux de Test

Chaque niveau de test a sa propre définition de "terminé". La liste suivante donne des exemples qui peuvent être pertinents pour les différents niveaux de test..

- Test unitaires
  - 100% de couverture des décisions quand c'est possible, avec des revues soignées des chemins infaisables
  - Analyses Statiques effectuées sur tout le code
  - Aucun défaut majeur non résolu
  - Pas de dette technique connue et inacceptable restant dans la conception et le code [Jones11]
  - Tout le code, les tests unitaires, et les résultats de tests unitaires revus
  - Tous les tests unitaires automatisés
  - Les caractéristiques importantes sont dans les limites acceptées (ex., performance)
- Tests d'intégration
  - Toutes les exigences fonctionnelles testées, incluant des tests positifs et négatifs, avec le nombre de tests basé sur la taille, la complexité et les risques
  - Toutes les interfaces entre les composants testées
  - Tous les risques Qualité couverts en accord avec la portée des tests
  - Tous les défauts trouvés sont rapportés
  - Pas de défaut majeur non résolu (priorisés en fonction des risques et de l'importance)
  - Tous les tests de régression automatisés, quand c'est possible, avec tous les tests automatisés stockés dans un entrepôt commun
- Tests Systèmes
  - Tests de bout en bout des user Story, des fonctionnalités et des fonctions
  - Profils utilisateurs couverts
  - Les caractéristiques qualité les plus importantes couvertes (Ex: performance, robustesse, sécurité)
  - Tests effectués dans de(s) environnement(s) iso production, incluant tous les matériels et logiciels pour les configurations supportées, dans la mesure du possible
  - Tous les risques qualité couverts en accord avec l'étendue des tests

© International Software Testing Qualifications Board

- Tous les tests de régression automatisés, quand possible, avec tous les tests automatisés stockés dans le même entrepôt commun
- Tous les défauts reportés
- Pas de défauts majeurs non résolus (priorisés en fonction des risques et de l'importance)

User Story

La définition de "Terminé" pour les User Story peut être déterminée par les critères suivant:

- Les user Story sélectionnées pour l'itération sont complètes en accord avec le thème du produit, comprises par l'équipe, et ont des critères d'acceptation détaillés et testables
- Tous les éléments de la User Story sont spécifiés et revus, incluant la réalisation des tests d'acceptation des User Story
- Les tâches nécessaires pour implémenter et tester les User Story sélectionnées ont été identifiées et estimées par l'équipe

Fonctionnalité

La définition de "Terminé" des fonctionnalités, qui peuvent être décomposées en de multiples User Story ou EPIC peut inclure:

- Toutes les user Story constituantes, avec des critères d'acceptation, sont définies et approuvées par le client
- La conception est complète, sans dette technique connue
- Le code est complet, sans dette technique connue ou de raffinement non fini
- Les tests unitaires ont été réalisés et ont atteint le niveau défini de couverture
- Les tests d'Intégration et les tests systèmes pour la fonctionnalité ont été réalisés en accord avec critères de couverture définis
- Aucun défaut majeur ne reste à corriger
- La documentation fonctionnelle est complète, ce qui peut inclure les notes de release, les manuels utilisateurs, et les fonctions d'aide en ligne

Itération

Le niveau suivant de critère concerne l'itération (sprint), dont voici ci-dessous un exemple:

- Toutes les fonctionnalités de l'itération sont prêtes et testées individuellement conformément au niveau de critère fonctionnel
- L'intégration de toutes les fonctionnalités et caractéristiques pour l'itération complétées et testées
- La documentation écrite, revue et approuvée

A ce point, le logiciel est potentiellement livrable car l'itération a été complétée avec succès, mais toutes itérations ne donnent pas une release comme résultat.

Release

La définition de "Terminé" pour une release, qui peut être divisée en de multiples itérations peut inclure:

- Couverture
  - Tous les éléments de base des tests pertinents pour tous les contenus de la release ont été couverts par des tests.
  - L'adéquation de la couverture est déterminée par ce qui est nouveau ou changé, sa complexité et taille, et les risques de défaut associés.
- Qualité

© International Software Testing Qualifications Board

- L'intensité des défauts (Ex: combien de défauts sont trouvés par jour ou par transaction),
- la densité de défauts (Ex: le nombre de défauts trouvés comparé au nombre de User Story, l'effort, et/ou aux attributs Qualité), et le nombre estimé de défauts restant étant dans des limites acceptables.
- Les conséquences des défauts corrigés et résiduels (ex., la sévérité et la priorité) sont compris et acceptables.
- Les niveaux de risques résiduels associés avec chaque risque qualité identifié sont compris et acceptables.
- La tendance des défauts et le nombre estimé de défauts restants dans le système

- Temps – Si la date de livraison prédéterminée est atteinte, considérer le fait d'être releasé ou pas en fonction des considérations Métier.
- Coûts – Les coûts estimés du cycle de vie devraient être utilisés pour calculer le retour sur investissement du système délivré (i.e., les coûts calculés du développement et de la maintenance devraient être considérablement plus bas que le total des ventes espéré du produit). La part la plus importante des coûts du cycle de vie provient souvent de la maintenance une fois le produit releasé, du fait du nombre de défauts laissés en production.

3.3.2 Appliquer le Développement piloté par les tests d'acceptation (ATDD)

Le développement piloté par les tests d'acceptation est une approche « test en premier ». Les cas de test sont créés avant d'implémenter la User Story. Les cas de test sont créés par l'équipe Agile incluant les développeurs, les testeurs, et les représentants Métier [Adzic09] et peuvent être manuels ou automatisés.

La première étape est un atelier de spécification, ou la User Story est analysée, discutée et écrite par les développeurs, testeurs, et représentants Métier. Toute incomplétude, ambigüité, ou erreurs dans la User Story sont corrigées Durant ce processus.

La prochaine étape est de créer les tests. Cela peut être fait par l'ensemble de l'équipe ou individuellement par le testeur. Dans tous les cas, une personne indépendante comme un représentant Métier valide les tests. Les tests dont des exemples qui décrivent les caractéristiques des User Story. Ces exemples vont aider l'équipe à implémenter correctement la User Story. Puisque les exemples et les tests sont les mêmes, ces termes sont souvent interchangeables. Le travail commence avec des exemples basiques et des questions ouvertes.

Typiquement, les premiers tests sont les tests des chemins positifs confirmant le comportement correct sans exception ou condition d'erreur, comprenant la séquence d'activités exécutée telle que souhaitée. Une fois les chemins positifs testés, l'équipe devrait également écrire des tests des chemins négatifs et couvrir des attributs non-fonctionnels (ex: performance, utilisabilité). Les tests sont exprimés de façon à ce que chaque partie prenante soit capable de comprendre, contenant des séquences en langage naturel impliquant la précondition nécessaire, et s'il y en a, les données d'entrées et les résultats associés.

Les exemples doivent couvrir toutes les caractéristiques de la User Story et ne pas rajouter des éléments à la Story. Cela signifie qu'il ne devrait pas exister d'exemple qui décrit un aspect de la User Story non documenté dans la Story elle-même. De plus deux exemples ne devraient pas décrire les mêmes caractéristiques de la User Story.

3.3.3 Conception des tests boite noire Fonctionnels et Non-Fonctionnels

Dans les tests Agile, de nombreux tests sont créés par les testeurs en parallèle des activités de programmation. Lorsque les développeurs programment en se basant sur les User Story et les critères d'acceptation, les développeurs créent les tests, basés sur les User Story et leurs critères

© International Software Testing Qualifications Board

d'acceptation. (Certains tests, comme les tests exploratoires et d'autres tests basés sur l'expérience, sont créés ultérieurement, pendant l'exécution des tests, comme expliqué dans la Section 3.3.4). Les testeurs peuvent appliquer les techniques de conception des tests boite noire traditionnelles, comme les partitions d'équivalence, l'analyse des valeurs limites, les tables de décision, et les tests de transition d'état pour créer ces tests.
Par exemple, l'analyse des valeurs limites peut être utilisée pour sélectionner les valeurs de test quand le client est limité en nombre d'élément qui peuvent être sélectionnés pour acheter.

Dans de nombreuses situations, les exigences non-fonctionnelles peuvent être documentées comme des User Stories. Les techniques de conception boite noire (comme l'analyse des valeurs limites) peut également être utilisées pour créer des tests pour des caractéristiques non-fonctionnelles. La User Story peut contenir des exigences de performance ou de fiabilité. Par exemple, une exécution donnée ne peut pas excéder une limite de temps ou un nombre d'opérations peut être en échec avant un certain nombre de fois.
Pour plus d'information sur l'utilisation des techniques de conception des tests boite noire, voir le syllabus niveau fondation [ISTQB_FL_SYL] et le syllabus analyste de test au niveau avancé [ISTQB_ALTA_SYL].

3.3.4 Les tests exploratoires et les tests Agile

Les tests exploratoires sont importants en Agile, dus aux limites de temps disponibles pour l'analyse des tests et le niveau limité des détails dans les User Story. De façon à obtenir les meilleurs résultats, les tests exploratoires devraient être combinés avec d'autres techniques basées sur l'expérience comme faisant partie de la stratégie de test réactive, mélangée avec d'autres stratégies de test comme les tests analytiques basés sur les risques, tests analytiques basés sur les exigences, test basés sur les modèles, et tests de non régression.
Les stratégies de test et les stratégies de test combinées sont discutées dans le syllabus niveau fondation [ISTQB_FL_SYL].

Dans les tests exploratoires, la conception des tests et l'exécution des tests se font au même moment, guidés par une charte de test préparée. Une charte de test fournit les conditions de test à couvrir pendant une session limitée dans le temps. Pendant les tests exploratoires, les résultats des tests les plus récents guident les suivants. Les mêmes techniques boite blanche et boite noire utilisées pour concevoir les tests préconçus, peuvent être utilisées pour concevoir les tests.

Une charte de tests peut inclure les éléments suivant:

- Acteur: Utilisateur attendu du système
- Sujet: Le thème de la charte incluant quel objectif particulier l'acteur veut réaliser, par exemple les conditions de test
- Setup: Ce qui doit être mis en place pour commencer l'exécution des tests
- Priorité: Importance relative de la charte, base sur la priorité des User Story associées ou du niveau de risqué
- Référence: Spécifications (Ex: User Story), risques, ou autres sources d'information
- Data: Tous les data nécessaires pour prendre en charge la charte
- Activités: Une liste d'idées de ce que l'acteur peut vouloir faire avec le système (Ex: "Log sur le système comme "super utilisateur") et ce qu'il serait intéressant de tester (à la fois les tests positifs et négatifs)
- Notes d'oracle: Comment évaluer le produit pour déterminer des résultats corrects (Ex pour capturer ce qui survient sur l'écran et le comparer à ce qui est écrit dans le manuel utilisateur)
- Variations: Actions alternatives et évaluations pour compléter les idées décrites derrière les activités.

© International Software Testing Qualifications Board

Pour gérer les tests exploratoires, une méthode appelée gestion par des sessions de test peut être utilisée. Une session est définie comme un période ininterrompue de test qui peut prendre de 60 à 120 minutes.

Les sessions de test incluent les éléments suivants:

- Session d'enquête (pour apprendre comment il fonctionne)
- Session d'analyse (évaluation de la fonctionnalité ou des caractéristiques)
- Profondeur de couverture (cas exotiques, scenarios, interactions)

La qualité des tests dépend de la capacité du testeur à poser les questions pertinentes sur ce qu'il teste. Les exemples incluent les suivants:

- Qu'est ce qui est le plus important à découvrir concernant le système?
- De quelle façon le système peut-il tomber en échec ?
- Qu'est ce qui se passe si.....?
- Qu'est ce qui se passera quand.....?
- Est-ce que les besoins utilisateur, exigences, et souhaits sont comblés?
- Est-ce qu'il est possible d'installer les systèmes (et de le supprimer si nécessaire) dans tous les circuits de mise à jour supportés?

Durant l'exécution des tests, le testeur utilise la créativité, l'intuition, les facultés cognitives et les compétences des testeurs pour trouver des défauts possibles dans le produit. Les testeurs ont également besoin d'avoir une bonne connaissance et une bonne compréhension du logiciel en test, du domaine Métier, de comment le logiciel est utilisé, et comment déterminer quand le système tombe en échec.

Un ensemble d'heuristiques peuvent être appliquées lors des tests. Une heuristique peut guider le testeur sur comment réaliser les tests et évaluer les résultats [Hendrickson]. Les exemples incluent:

- Les limites
- CRUD (Créer (Create), Lire(Read), Mettre à jour(Update), Effacer(Delete))
- Variations de configuration
- Interruptions (ex., se déconnecter, éteindre, ou rebooter)

Il est important que le testeur documente le processus autant que possible. Autrement, il serait difficile de revenir en arrière et montrer comment un problème a été découvert. La liste suivante fournit des exemples d'information qui peuvent être utiles pour documenter:

- Couverture de Test: Quelles données d'entrée ont été utilisées, combien a été couvert, et combien reste à tester
- Notes d'évaluation*:* Le*s* observations durant les tests faisant que le système et les fonctionnalités en test semblent être stables, est-ce que tous les défauts ont été trouvés, qu'est ce qui est planifié à l'étape suivante en accord avec les observations courantes et toute autre liste d'idées.
- Listes de Risques/Stratégie: Quels risques ont été couverts et lesquels restent parmi les plus importants, est-ce que la stratégie initiale a été suivie, est-ce qu'elle a besoin de changements
- Points en suspens, questions, et anomalies: tout comportement non attendu, toutes question concernant l'efficacité de l'approche, toute préoccupation au sujet des idées /tentatives de test, environnements de test, données de test, non compréhensions de la fonction, script de test ou le système en test
- Comportement actuel*:* enregistrer le comportement réel du système qui a besoin d'être sauvé (vidéo, captures d'écran, fichiers de données de sortie)

L'information enregistrée devrait être capturée et/ou résumée sous la forme d'outils de gestion des statuts (ex: outils de gestion des tests, outils de gestion des tâches, le tableau des tâches), de façon à

© International Software Testing Qualifications Board

ce qu'il soit facile aux parties prenantes de comprendre le statut en cours pour tous les tests qui ont été réalisés.

3.4 Outils dans les projets Agile

Les outils décrits dans le Syllabus niveau Fondation [ISTQB_FL_SYL] sont pertinents et utilisés par les testeurs des équipes Agiles. Tous les outils ne sont pas utilisés de la même façon et certains outils sont plus pertinents pour les projets Agile qu'ils ne le sont dans les projets traditionnels.

Par exemple, alors que les outils de gestion des tests, outils de gestion des exigences, outils de gestion des incidents (outils de suivi des défauts) peuvent être utilisés par les équipes Agiles, des équipes Agile optent pour un outil tout en un (ex: gestion du cycle de vie de l'application ou gestion de tâches) qui fournit des fonctionnalités pertinentes pour le développement Agile, comme les tableaux de tâches, les Burndown Charts, et les User Story. Les outils de gestion de configuration sont importants pour les testeurs des équipes Agile à cause du nombre élevé de tests automatisés à tous les niveaux et le besoin de stocker et de gérer les artéfacts de test automatisés associés.

En plus des outils décrits dans le Syllabus niveau Fondation [ISTQB_FL_SYL], les testeurs sur les projets Agile peuvent utiliser également les outils décrits dans les sous-sections suivantes. Ces outils sont utilisés par l'équipe intégrée pour assurer la collaboration d'équipe et le partage d'informations, qui sont des pratiques clés des pratiques Agile.

3.4.1 Outils de gestion des tâches et de suivi

Dans certains cas, les équipes Agile utilisent des tableaux physiques pour les Story et les tâches (ex., tableaux blancs, tableau en liège) pour gérer et suivre les user Story, les tests, et autres tâches au travers de chaque sprint. D'autres équipes utiliseront des applications de gestion du cycle de vie et des logiciels de gestion de tâches, incluant les tableaux de bord électroniques. Ces outils sont utilisés pour ce qui suit:

- Enregistrer les Story et leurs tâches de développement et de test, pour assurer que rien n'est perdu durant un sprint
- Capturer les estimations des membres de l'équipe sur leurs tâches et automatiquement calculer l'effort requis pour implémenter une Story, pour rendre efficace une session de planification
- Les tâches de développement associées et les tâches de test pour une même Story, pour fournir une vue complète de l'effort de l'équipe pour implémenter la Story
- Agréger les mises à jour des développeurs et des testeurs sur le statut de la tâche quand ils finissent leur travail, fournir automatiquement un instantané calculé du statut de chaque Story, de l'itération, et de la release entière
- Fournir une représentation visuelle (via des métriques, graphiques et tableaux de bord) de l'état en cours de chaque User Story, itération, et release, permettant à toutes les parties prenantes, incluant les membres des équipes distribuées géographiquement, de vérifier rapidement les statuts
- Intégrer avec des outils de gestion de configuration, qui peuvent permettre des mises sous contrôle automatisées du code et des Builds, et, dans certains cas, les mises à jour des statuts automatisés des tâches

3.4.2 Outils de communication et de partage d'information

En plus des e-mails, des documents, et de la communication verbale, les équipes Agiles utilisent souvent trois types d'outils additionnels pour la communication et le partage de l'information: wikis, messageries instantanées, et partage d'écran.

© International Software Testing Qualifications Board

Les Wikis permettent à l'équipe de construire et de partager une base de connaissance en ligne concernant des aspects variés du projet, incluant les éléments suivant:

- Diagrammes de fonctionnalités Produit, discussion sur les fonctionnalités, diagrammes de prototypes, photos de discussions sur tableau blanc, et autres informations
- Outils et/ou techniques pour développer et tester identifiées comme utiles par les autres membres de l'équipe
- Métriques, graphiques, et tableaux de bord des statuts du produit, qui sont particulièrement utiles quand le wiki est intégré avec d'autres outils comme le serveur de Build et le système de gestion de tâches, puisque l'outil peut mettre à jour les statuts du produit automatiquement
- Les messageries instantanées, téléconférences audio, et outils de chat vidéo apportent les bénéfices suivants:
    - Permettre une communication directe en temps réel entre les membres de l'équipe, en particulier les équipes distribuées
    - Impliquer les équipes distribuées dans les stand up meetings
    - Réduire les notes de téléphone en utilisant la technologie VOIP, supprimer les coûts des contraintes qui pourraient réduire la communication des membres de l'équipe dans des situations distribuées
- Les outils de partages d'écran et de capture fournissent les bénéfices suivants:
    - Dans les équipes distribuées, des démonstrations du produit, des revues de code, et même le travail en binôme peuvent avoir lieu.
    - Capturer les démonstrations du produit à la fin de chaque itération, qui peuvent être placées sur les wikis des équipes

Ces outils devraient être utilisés en complément et renforcer, pas remplacer, la communication en face à face dans les Équipe Agiles.

3.4.3 Build de logiciel et outils de distribution

Comme discuté plus tôt dans ce syllabus, les Builds et les déploiements journaliers de logiciels sont une pratique clé dans les équipes Agiles. Cela requiert l'utilisation d'outils d'intégration continue et des outils de distribution.

L'utilisation, les bénéfices, et les risques de ces outils a été décrite plus tôt dans la section 1.2.4.

3.4.4 Outils de gestion de configuration

Dans les équipes Agile, les outils de gestion de configuration peuvent être utilisés pour ne pas seulement stocker le code source et les tests automatisés, mais les tests manuels et les autres produits des tests sont souvent stockés dans le même entrepôt que le code source du produit. Cela donne de la traçabilité sur quelles versions du logiciels ont été testées avec quelles versions particulières des tests, et permettent des changements rapides sans perdre l'historique. Les types principaux de systèmes de contrôle de version incluent les systèmes de contrôle centralisés de sources et les systèmes de contrôle de version distribués. La taille de l'équipe, sa structure sa localisation, et les exigences d'intégration avec d'autres outils détermineront quel système de contrôle de version est le bon pour un projet Agile en particulier.

3.4.5 Outils de conception, d'implémentation, et d'exécution des tests

- Certains outils sont utiles aux testeurs Agile à des points particuliers du processus de test logiciel. Alors que dans leur majorité, ces outils ne sont pas nouveaux ou spécifiques à l'Agile, ils fournissent des capacités importantes étant donné les changements rapides des projets Agile.
- Outils de conception de test: L'utilisation d'outils comme les mind-maps est devenue plus populaire pour concevoir rapidement et définir les tests d'une nouvelle fonctionnalité.

© International Software Testing Qualifications Board

- Outils de gestion des cas de test: Ce type d'outil utilisé en Agile peut être une partie de l'outil de gestion du cycle de vie de l'application de l'équipe intégrée ou l'outil de gestion des tâches.
- Données de test, préparation et génération des données: Les outils qui génèrent les données qui peuplent une base de données applicative sont très bénéfiques quand beaucoup de données et de combinaisons de données sont nécessaires pour tester l'application. Ces outils peuvent également aider à redéfinir la structure de la base de données alors que le produit subit des changements pendant un projet Agile et raffine les scripts pour générer les données. Cela permet une mise à jour rapide des données de test quand des changements surviennent. Certains outils de préparation des données de test utilisent les sources de données de production comme matière première, et utilisent des scripts pour supprimer ou anonymiser des données sensibles. D'autres outils de préparation des données de test peuvent aider en validant des données d'entrées et de résultats de grandes tailles.
- Outils de chargement de données de test: Une fois les données générées pour le test, elles ont besoin d'être chargées dans l'application. L'entrée manuelle de données est souvent chronophage et facteur d'erreurs, mais les outils de chargement de données existent pour rendre le processus fiable et efficace. En fait, de nombreux outils de génération de données incluent un composant de chargement de données intégré. Autrement, le chargement en vrac à partir d'un système de gestion de base de données est également possible.
- Outils d'exécution de tests automatisés: Ce sont les outils d'exécution de tests qui sont les plus en phase avec les tests Agile. Des outils spécifiques sont disponibles sous forme commerciale ou open source qui supportent les approches « test en premier », comme le développement piloté par le comportement, le développement piloté par les tests d'acceptation. Ces outils permettent aux testeurs et aux équipes Métier d'exprimer le comportement du système attendu dans des tableaux ou en langage naturel utilisant des mots clé.
- Outils de tests exploratoires: Les outils qui capturent et enregistrent les activités réalisées sur une application pendant les sessions de tests exploratoires sont bénéfiques pour le testeur et le développeur, puisqu'ils enregistrent les actions prises. Cela est utile quand un défaut est détecté, puisque les actions faites avant la défaillance ont été capturées et peuvent être utilisées pour rapporter le défaut aux développeurs. L'enregistrement des étapes effectuées lors d'une session de tests exploratoires montre un bénéfice quand le test est finalement inclus dans les suites de tests de régression automatisés.

3.4.6 Outils de Cloud Computing et de virtualisation

La virtualisation permet à une simple ressource physique (serveur) d'opérer comme de nombreuses ressources plus petites et séparées. Quand des machines virtuelles ou des instances de cloud sont utilisées, les équipes ont un plus grand nombre de serveurs disponibles pour eux pour le développement et les tests. Cela peut aider à annuler les délais associés avec l'attente de serveurs physiques. Provisionner un nouveau serveur ou en restaurer un est plus efficace grâce aux capacités à faire un instantané de la plupart des outils de virtualisation. Certains outils de gestion des tests utilisent maintenant les technologies de virtualisation pour faire des instantanés des serveurs au moment où une faute est détectée, ce qui permet aux testeurs de partager l'instantané avec les développeurs pour investiguer les fautes.

© International Software Testing Qualifications Board

4. Références

4.1 Standards

[DO-178B] RTCA/FAA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992.
[ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE), 2005.

4.2 Documents ISTQB et CFTL

[ISTQB_ALTA_SYL] ISTQB Syllabus niveau avancé Analyste de test, Version 2012
[ISTQB_ALTM_SYL] ISTQB Syllabus niveau avancé Test Manager, Version 2012
[ISTQB_FA_OVIEW] ISTQB Fondation Level Agile Testeur Overview, Version 1.0
[ISTQB_FL_SYL] ISTQB Syllabus niveau Fondation, Version 2011

4.3 Livres

[Aalst13] Leo van der Aalst and Cecile Davis, “TMap NEXT® in Scrum,” ICT-Books.com, 2013.
[Adzic09] Gojko Adzic, “Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing,” Neuri Limited, 2009.
[Anderson13] David Anderson, “Kanban: Successful Evolutionary Change for Your Technology Business,” Blue Hole Press, 2010.
[Beck02] Kent Beck, “Test-driven Development: By Example,” Addison-Wesley Professional, 2002.
[Beck04] Kent Beck and Cynthia Andres, “Extreme Programming Explained: Embrace Change, 2e” Addison-Wesley Professional, 2004.
[Black07] Rex Black, “Pragmatic Software Testing,” John Wiley and Sons, 2007.
[Black09] Rex Black, “Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 3e,” Wiley, 2009.
[Chelimsky10] David Chelimsky et al, “The RSpec Book: Behavior Driven Development with Rspec, Cucumber, and Friends,” Pragmatic Bookshelf, 2010.
[Cohn04] Mike Cohn, “User Story Applied: For Agile Software Development,” Addison-Wesley Professional, 2004.
[Crispin08] Lisa Crispin and Janet Gregory, “Agile Testing: A Practical Guide for Testers and Agile Teams,” Addison-Wesley Professional, 2008.
[Goucher09] Adam Goucher and Tim Reilly, editors, “Beautiful Testing: Leading Professionals Reveal How They Improve Software,” O'Reilly Media, 2009.
[Jeffries00] Ron Jeffries, Ann Anderson, and Chet Hendrickson, “Extreme Programming Installed,” Addison-Wesley Professional, 2000.
[Jones11] Capers Jones and Olivier Bonsignour, “The Economics of Software Quality,” Addison-Wesley Professional, 2011.
[Linz14] Tilo Linz, “Testing in Scrum: A Guide for Software Quality Assurance in the Agile World,” Rocky Nook, 2014.
[Schwaber01] Ken Schwaber and Mike Beedle, “Agile Software Development with Scrum,” Prentice Hall, 2001.
[vanVeenendaal12] Erik van Veenendaal, “The PRISMA approach”, Uitgeverij Tutein Nolthenius, 2012.
[Wiegers13] Karl Weigers and Joy Beatty, “Software Requirements, 3e,” Microsoft Press, 2013

© International Software Testing Qualifications Board

4.4 Terminologie Agile

Les termes qui existent dans le glossaire CFTL/ISTQB sont identifiés au début de chaque chapitre. Pour les termes Agile habituels, nous nous sommes alignés sur les ressources internet ci-dessous communément acceptées pour fournir des définitions :

http://guide.Agilealliance.org/
http://searchlogicielquality.techtarget.com
http://whatis.techtargetcom/glossary
http://www.scrumalliance.org/

Nous encourageons les lecteurs à vérifier ces sites s'ils trouvent des termes Agile non familiers dans ce document. Ces liens étaient actifs au moment de la release de ce document.

4.5 Autres Références

Les références suivantes pointent des informations disponibles sur Internet et ailleurs. Même si ces références ont été vérifiées au moment de la publication de ce syllabus, l'ISTQB ne peut pas être tenu responsable si des références ne sont plus disponibles.

[Agile Alliance Guide] Différents contributeurs, http://guide.Agilealliance.org/.
[Agilemanifesto] Différents contributeurs, www.agilemanifesto.org.
[Hendrickson]: Elisabeth Hendrickson, "Acceptance test driven development," testobsessed.com/2008/12/acceptance-test-driven-development-atdd-an-overview.
[INVEST] Bill Wake, "INVEST in Good Story, and SMART Tasks," xp123.com/articles/invest-in-good-Story-and-smart-tasks.
[Kubaczkowski] Greg Kubaczkowski and Rex Black, "Mission Made Possible," www.rbcs-us.com/images/documents/Mission-Made-Possible.pdf.
[Scrum Guide] Ken Schwaber and Jeff Sutherland, editors, "The Scrum Guide," www.scrum.org.
[Sourceforge] Différents contributeurs, www.sourceforge.net.

© International Software Testing Qualifications Board

5. Index

Agile software development, 12
Agile task boards, 25
amélioration du processus, 25
analyse des causes racines, 14
analyse du risque qualité, 33
approche de test, 16, 29
automatisation de l'exécution des tests, 29
automatisation des tests, 17, 21, 24, 25, 26, 27, 33
backlog de produit, 12, 13, 14, 16, 17, 33
backlog de sprint, 12, 17
bases de test, 17, 36
burndown charts, 24, 25, 42
carte de story, 14
charte de tests, 29, 40
Concept des 3C, 14
conception des tests incrémentale, 33
contrôle de version, 43
critères d'acceptation, 13, 14, 17, 22, 25, 27, 28, 29, 30, 31, 37, 38, 40
dette technique, 20, 26
développement piloté par les tests, 22, 29
élément de configuration, 19
epics, 22
estimation des tests, 29
framework de test unitaires, 29
gestion de configuration, 19, 26, 42, 43
incrément, 12
intégration continue, 12, 15, 16, 23, 26, 27, 30, 31, 33, 43
INVEST, 14
Kanban, 13
mind-mapping, 34
modèle des quadrants des tests, 31
oracle de test, 17
outil de génération de données, 44
outil de préparation des données de tests, 44
planification d'itération, 16, 17, 20, 22, 25, 28, 34, 42
planification de release, 14, 16, 17, 20, 26, 33, 34
Product Owner, 13
produits d'activité des projets, 21
programmation tests en premier, 12
pyramide des tests, 29, 31
quadrants des tests, 31
raffinement du backlog, 12
regression testing, 15, 21, 26, 29, 30
rétrospective, 14, 15, 32
risque produit, 29, 35
risque qualité, 17, 22, 29, 39
Scrum, 12, 13, 22, 32, 45
Scrum Master, 12
sprint, 12
stand-up meeting journalier, 25
stand-up meetings, 24, 25
stratégie de test, 28, 29, 32, 33
tableau de tâches Agile, 24
Tableau Kanban, 13
taxonomie de défauts, 29
test de vérification du build, 19, 27
test pilotés par les tests d'acceptation, 39
tests d'acceptation, 17, 23, 26, 38
tests d'acceptation, 16
tests d'utilisation, 29, 32
tests de maintenabilité, 29
tests de performance, 29
tests de sécurité, 29, 32
tests en binômes, 33
tests exploratoires, 21, 29, 32, 40, 41
timeboxing, 12, 13
transparence, 12
user stories, 13, 14, 15, 16, 17, 20, 22, 25, 28, 33, 35, 36, 37, 38, 40, 42
user story, 14, 17, 22, 23, 27, 30, 31, 35, 39, 42
vélocité, 17, 26
XP Voir Extreme Programming

© International Software Testing Qualifications Board